DrillBit

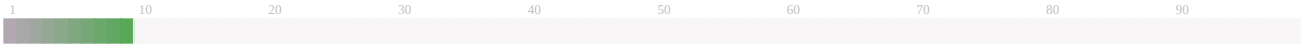
The Report is Generated by DrillBit Plagiarism Detection Software

Submission Information

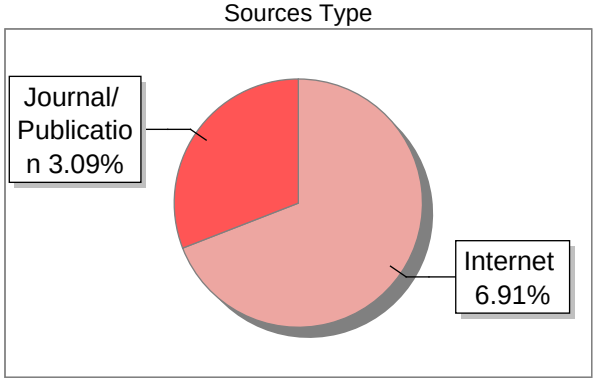
Author Name	MTSOU
Title	CSM-6252
Paper/Submission ID	3643407
Submitted by	librarian@mtsou.edu.in
Submission Date	2025-05-21 12:05:19
Total Pages, Total Words	63, 10340
Document type	Others

Result Information

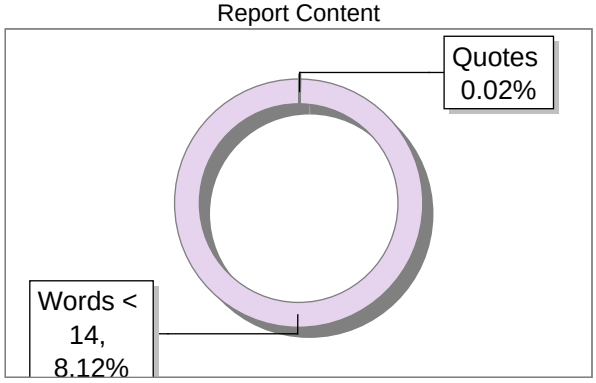
Similarity **10 %**



Sources Type



Report Content



Exclude Information

Quotes	Excluded
References/Bibliography	Excluded
Source: Excluded < 14 Words	Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	Yes

A Unique QR Code use to View/Download/Share Pdf File

DrillBit

DrillBit Similarity Report

10	22	A
SIMILARITY %	MATCHED SOURCES	GRADE

A-Satisfactory (0-10%)
B-Upgrade (11-40%)
C-Poor (41-60%)
D-Unacceptable (61-100%)

LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
1	admissions.keralauniversity.ac.in	1	Publication
2	cbseacademic.nic.in	1	Publication
3	skymark.in	1	Internet Data
4	fastercapital.com	1	Internet Data
5	www.blockchain-council.org	1	Internet Data
6	www.imensosoftware.com	1	Internet Data
7	www.jnu.ac.in	1	Internet Data
8	www.linkedin.com	1	Internet Data
9	new.kuk.ac.in	<1	Publication
10	www.geeksforgeeks.org	<1	Internet Data
11	bcastudyguide.com	<1	Internet Data
12	www.tutorialspoint.com	<1	Internet Data
13	www.linkedin.com	<1	Internet Data
14	alistapart.com	<1	Internet Data

15	mu.ac.in	<1	Publication
16	IEEE 2012 Innovative Parallel Computing (InPar) - San Jose, CA, USA by	<1	Publication
17	www.geeksforgeeks.org	<1	Internet Data
18	pdfcookie.com	<1	Internet Data
19	thepostindia.co.in	<1	Internet Data
20	home.iiserb.ac.in	<1	Publication
21	oorwin.com	<1	Internet Data
22	www.linkedin.com	<1	Internet Data

CSM -6252 DAA & WEB PROGRAMMINGLAB

COURSE INTRODUCTION

Welcome to the **DAA & Web Programming Lab**, a key component of the computer science curriculum. This lab is designed to equip students with essential skills in both **algorithm design** and **web development**. It seamlessly blends the theoretical underpinnings of algorithm analysis with the practicalities of web programming, offering a thorough learning experience that prepares students for the diverse challenges of the tech industry.

Design and Analysis of Algorithms (DAA) Segment

In this part of the lab, students will explore the core principles of algorithmic strategies, including **divide and conquer**, **dynamic programming**, and **greedy algorithms**. A strong focus is placed on comprehending and evaluating the efficiency of algorithms using **Big O notation**, which is vital for optimizing performance in large-scale applications. Students will also gain hands-on experience implementing various **data structures** like arrays, linked lists, and trees, solidifying their theoretical understanding with practical application.

Web Programming Segment

The web programming portion focuses on building robust and interactive web applications. Students will learn essential web technologies such as **HTML**, **CSS**, **JavaScript**, and server-side scripting languages. The curriculum covers both **front-end** and **back-end development**, enabling students to construct full-stack web applications. Emphasis will be placed on best practices in web development, user experience design, and security considerations to ensure the creation of high-quality web solutions.

Throughout the lab, students will participate in **hands-on projects** and **collaborative exercises**, fostering both individual and team-based problem-solving skills. By integrating the study of algorithms with practical web development, this course aims to provide a well-rounded education that bridges the gap between theoretical computer science and real-world applications. Whether your aspirations lie in algorithmic research or web development, this lab will prove invaluable in building a strong foundation for your future career.

OVERVIEW

This book presents the DAA & Web Programming Lab, an integrated approach to mastering two core areas of computer science: algorithm design and web development.

In the Design and Analysis of Algorithms (DAA) section, you'll delve into fundamental algorithmic strategies like divide and conquer, dynamic programming, and greedy algorithms. A key focus will be on evaluating algorithm efficiency using Big O notation.

The Web Programming segment covers essential web technologies, including HTML, CSS, JavaScript, and server-side scripting. This will empower you to build comprehensive full-stack web applications.

Through a combination of hands-on projects and collaborative exercises, you'll gain practical experience and sharpen your problem-solving abilities, preparing you for the diverse challenges of the tech industry. This lab bridges the gap between theoretical knowledge and real-world application, providing a solid foundation for future careers in both algorithm research and web development.

What You'll Learn:

Here's a breakdown of what you'll gain from this lab:

- **Core Algorithmic Techniques:** Master fundamental strategies such as divide and conquer, dynamic programming, and greedy algorithms.
- **Algorithm Efficiency Analysis:** Learn to evaluate algorithm performance using Big O notation.
- **Data Structure Implementation:** Get practical experience implementing essential data structures like arrays, linked lists, and trees.
- **Web Technology Mastery:** Become proficient in key web technologies, including HTML, CSS, and JavaScript.
- **Full-Stack Web Development:** Develop server-side scripts to create complete full-stack web applications.
- **Web Development Best Practices:** Understand best practices in web development and user experience design.
- **Web Security Principles:** Learn important security considerations in web programming.
- **Hands-on Project Experience:** Gain practical experience through engaging projects and collaborative exercises.
- **Enhanced Problem-Solving:** Improve your problem-solving abilities in both algorithmic and web development contexts.
- **Teamwork and Communication:** Develop essential teamwork and communication skills crucial for success in the tech industry.

How You'll Learn:

In the DAA & Web Programming Lab, you'll learn through a dynamic blend of interactive lectures and hands-on projects. This approach ensures that theoretical concepts are immediately reinforced with practical application. Structured lab exercises will guide you step-by-step through implementing algorithms and building web applications. Meanwhile, collaborative work will sharpen your teamwork and communication skills. You'll receive valuable feedback through regular code reviews and assessments, helping you refine your coding practices and improve solution efficiency.

For deeper insights into complex topics, you'll have access to workshops, tutorials, and peer learning sessions. Plus, instructor office hours offer personalized support. To further enhance your learning, online resources, including video tutorials and forums, are readily available, ensuring you gain a comprehensive understanding of both algorithm design and web development.

Who Is This BookFor:

This book is for computer science students and professionals eager to enhance their understanding of both algorithm design and web development. It's perfect if you're looking for a balanced approach that merges theoretical knowledge with practical skills.

Whether you're a beginner aiming to build a solid foundation in both fields, or an experienced developer looking to refine your expertise, this book offers valuable insights and hands-on experience. It's also well-suited for anyone preparing for technical interviews or striving to improve their problem-solving abilities and coding efficiency in real-world applications.

Conclusion:

By the end of this book, you'll have a strong foundation in both **algorithm design** and **web development**. The DAA & Web Programming Lab offers a comprehensive learning experience, perfectly blending the theoretical aspects of algorithms with the practical skills needed for web development.

Through engaging **hands-on projects**, **collaborative exercises**, and detailed **coding reviews**, you'll gain a solid understanding of how to design efficient algorithms and create dynamic web applications. This course will not only boost your **problem-solving abilities** and **technical proficiency** but also prepare you for the diverse challenges within the tech industry.

Whether you're looking to deepen your academic knowledge or excel in your professional career, the skills you acquire in this lab will be incredibly valuable for your future success in the constantly evolving field of computer science.

SESSION 1 DESIGN AND ANALYSIS OF ALGORITHMS LAB

Basic Structure

1.0 INTRODUCTION

This session dives into **problem-solving strategies**, particularly how they apply to computer programming. We'll specifically explore the **Design and Analysis of Algorithms (DAA) Lab**, a vital part of computer science education that provides hands-on experience in creating efficient and effective algorithms.

In this lab, students thoroughly investigate algorithm design, learning to approach problem-solving with a structured and analytical mindset. A core focus is on understanding various algorithmic strategies such as **divide and conquer, dynamic programming, greedy algorithms, and backtracking**. Through practical exercises, students gain a deeper appreciation for the theoretical concepts covered in lectures, seeing firsthand how these principles are applied in real-world scenarios.

Emphasis on Algorithm Efficiency

A key aspect of this lab is its strong emphasis on **algorithm efficiency**. Students learn to evaluate algorithm performance using **Big O notation**, which offers a high-level

understanding of an algorithm's time and space complexity. This analysis is critical for determining whether algorithms are feasible for large-scale problems, where even small inefficiencies can lead to significant performance bottlenecks. By comparing different algorithms for the same problem, students develop the ability to select the most appropriate solution based on context and constraints, thereby enhancing their problem-solving skills and algorithmic thinking.

Implementation of Data Structures

Another crucial element of the DAA Lab is the **implementation of various data structures**. Understanding the interplay between algorithms and data structures is essential, as the choice of data structures can profoundly impact an algorithm's efficiency and simplicity. Students will implement and manipulate structures like **arrays, linked lists, trees, graphs, and hash tables**, learning how these can be leveraged to optimize algorithm performance. This hands-on experience solidifies their understanding of theoretical concepts and prepares them for more advanced topics in computer science.

Collaboration and Practical Application

Collaboration and experimentation are highly encouraged in the lab, fostering a learning environment where students can discuss ideas and troubleshoot challenges together. This collaborative approach helps develop vital communication and teamwork skills, which are crucial in professional settings. Additionally, the lab often includes projects and assignments that require students to design, implement, and test their algorithms, providing a comprehensive learning experience that bridges the gap between theory and practice.

Overall, the Design and Analysis of Algorithms Lab is an essential part of the computer science curriculum, equipping students with the practical skills and analytical tools necessary for tackling complex computational problems. This unit will explain the fundamentals of the Design and Analysis of Algorithms Lab to you.

1.1 OBJECTIVES

After completing this unit, you will be able to:

- Learn fundamental principles of algorithm design.
- Evaluate algorithm efficiency using Big O notation.
- Implement core algorithms to reinforce theoretical concepts.
- Utilize various data structures to optimize performance.
- Enhance problem-solving skills with structured thinking.
- Compare algorithms to choose the most efficient solutions.
- Develop teamwork and effective communication skills.

- Apply theoretical knowledge in practical scenarios.
- Encourage innovation and experimentation in algorithm design.
- Prepare for advanced computer science topics and professional development.

1.2 IMPLEMENTATION OF SORTING ALGORITHMS

Sorting algorithms are a cornerstone of computer science, offering essential methods for organizing data into a specific sequence. For students in a Design and Analysis of Algorithms Lab, comprehending and implementing these algorithms is vital. This section will elaborate on the implementation of various sorting algorithms, focusing on their core principles, practical code examples, and performance analysis.

1.2.1 Bubble Sort

Bubble Sort is a straightforward, comparison-based algorithm. It works by repeatedly iterating through a list, comparing adjacent elements, and swapping them if they are not in the desired order. This iterative process continues until the entire list is sorted.

Code Example (Python):

```
defbubble_sort(arr):
    n = len(arr)
fori in range(n):
for j in range(0, n-i-1):
ifarr[j] >arr[j+1]:
arr[j], arr[j+1] = arr[j+1], arr[j]
returnarr
```

Example

```
arr = [64, 34, 25, 12, 22, 11, 90]
sorted_arr = bubble_sort(arr)
print("Sorted array:", sorted_arr)
```

1.2.2 Selection Sort

Selection Sort divides the input list into two parts: the sublist of items already sorted and the sublist of items remaining to be sorted. It repeatedly selects the smallest (or largest, depending on the order) element from the unsorted sublist and moves it to the sorted sublist.

```
defselection_sort(arr):
        n = len(arr)
    fori in range(n):
```

```
min_idx = i
for j in range(i+1, n):
ifarr[j] <arr[min_idx]:
min_idx = j
arr[i], arr[min_idx] = arr[min_idx], arr[i]
returnarr
```

Example

```
arr = [64, 25, 12, 22, 11]
sorted_arr = selection_sort(arr)
print("Sorted array:", sorted_arr)
```

1.2.3 Insertion Sort

Insertion Sort constructs the final sorted array by inserting one element at a time into its correct position. While simple, it's significantly less efficient for large datasets compared to more advanced algorithms such as Quicksort, Heapsort, or Merge Sort.

```
definsertion_sort(arr):
fori in range(1, len(arr)):
key = arr[i]
    j = i-1
while j >= 0 and key <arr[j]:
arr[j + 1] = arr[j]
    j -= 1
arr[j + 1] = key
returnarr
```

Example

```
arr = [12, 11, 13, 5, 6]
sorted_arr = insertion_sort(arr)
print("Sorted array:", sorted_arr)
```

1.2.4 Merge Sort

Merge Sort operates on the principle of divide and conquer. It works by splitting the input array into two equal halves, recursively sorting each half, and then combining the two sorted halves back into a single, sorted array.

Example

```
defmerge_sort(arr):
        iflen(arr) > 1:
mid = len(arr) // 2
```

```
L = arr[:mid]
R = arr[mid:]
merge_sort(L)
merge_sort(R)
i = j = k = 0
whilei<len(L) and j <len(R):
if L[i] < R[j]:
arr[k] = L[i]
i += 1
else:
arr[k] = R[j]
j += 1
k += 1
whilei<len(L):
arr[k] = L[i]
i += 1
k += 1
while j <len(R):
arr[k] = R[j]
j += 1
k += 1
returnarr
```

Example

```
arr = [12, 11, 13, 5, 6, 7]
sorted_arr = merge_sort(arr)
print("Sorted array:", sorted_arr)
```

1.2.5 Quick Sort

Quick Sort is a highly efficient sorting algorithm that relies on the concept of partitioning. It works by selecting a pivot element from the array. Then, it rearranges the other elements so that all elements smaller than the pivot are placed to its left, and all elements larger than the pivot are moved to its right. This partitioning step effectively divides the array into two sub-arrays. The Quick Sort process is then recursively applied to these smaller sub-arrays until the entire array is sorted.

```
defquick_sort(arr):
iflen(arr) <= 1:
```

```
returnarr
else:
pivot = arr[len(arr) // 2]
left = [x for x in arr if x < pivot]
middle = [x for x in arr if x == pivot]
right = [x for x in arr if x > pivot]
returnquick_sort(left) + middle + quick_sort(right)
```

Example

```
arr = [3, 6, 8, 10, 1, 2, 1]
sorted_arr = quick_sort(arr)
print("Sorted array:", sorted_arr)
```

1.3 IMPLEMENTATION OF SEARCHING ALGORITHMS

Searching algorithms are fundamental to computer science, serving the crucial purpose of locating specific elements within various data structures. Implementing these algorithms offers students a deeper insight into their foundational principles, efficiency, and real-world utility. This section will detail the implementation of core searching algorithms, providing code examples and an analysis of their performance.

1.3.1 Linear Search

Linear Search is a simple and direct algorithm. It operates by sequentially examining each element in a list until the target element is discovered or the end of the list is reached.

Example

```
deflinear_search(arr, x):
fori in range(len(arr)):
ifarr[i] == x:
returni
return -1
```

Example

```
arr = [2, 3, 4, 10, 40]
x = 10
result = linear_search(arr, x)
if result != -1:
```

```
print(f"Element found at index {result}")
else:
print("Element not found")
```

1.3.2 Binary Search

Binary Search is a highly efficient algorithm for finding an element in a sorted array. It works by repeatedly dividing the search interval in half, comparing the target value to the middle element, and narrowing the search range accordingly.

Example

```
defbinary_search(arr, x):
left, right = 0, len(arr) - 1
while left <= right:
mid = (left + right) // 2
ifarr[mid] == x:
return mid
elifarr[mid] < x:
left = mid + 1
else:
right = mid - 1
return -1
```

Example

```
arr = [2, 3, 4, 10, 40]
x = 10
result = binary_search(arr, x)
if result != -1:
print(f"Element found at index {result}")
else:
print("Element not found")
```

1.3.3 Jump Search

Jump Search is an algorithm designed for sorted arrays that aims to reduce the number of comparisons needed to find an element. It achieves this by taking "jumps" of fixed steps through the array. Once the block containing the target element is identified (or the block where it should be), a linear search is then performed only within that specific block.

Example

```
import math
```

```
defjump_search(arr, x):
    n = len(arr)
step = int(math.sqrt(n))
prev = 0
whilearr[min(step, n) - 1] < x:
prev = step
step += int(math.sqrt(n))
ifprev>= n:
return -1

fori in range(prev, min(step, n)):
ifarr[i] == x:
returni
return -1
```

Example

```
arr = [0, 1, 2, 4, 5, 7, 9, 10, 12]
x = 10
result = jump_search(arr, x)
if result != -1:
print(f"Element found at index {result}")
else:
print("Element not found")
```

1.3.4 Interpolation Search

Interpolation Search offers an enhancement over Binary Search, particularly when dealing with **uniformly distributed data**. Instead of simply dividing the search space in half, it intelligently estimates the probable position of the desired value by considering the values present at the boundaries of the current search interval.

Example

```
definterpolation_search(arr, x):
low = 0
high = len(arr) - 1
while low <= high and arr[low] <= x <= arr[high]:
if low == high:
ifarr[low] == x:
return low
```

```
return -1
pos = low + ((high - low) // (arr[high] - arr[low]) * (x - arr[low]))
ifarr[pos] == x:
return pos
ifarr[pos] < x:
low = pos + 1
else:
high = pos - 1
return -1
```

Example

```
arr = [10, 12, 13, 16, 18, 19, 20, 21, 22, 23]
x = 18
result = interpolation_search(arr, x)
if result != -1:
print(f"Element found at index {result}")
else:
print("Element not found")
```

1.3.5 Exponential Search

Exponential Search is an algorithm that finds the range where the element may be present and then performs Binary Search within that range. It is particularly useful for unbounded or infinite lists.

Example

```
defbinary_search(arr, left, right, x):
while left <= right:
mid = (left + right) // 2
ifarr[mid] == x:
return mid
elifarr[mid] < x:
left = mid + 1
else:
right = mid - 1
return -1
defexponential_search(arr, x):
ifarr[0] == x:
return 0
```

```
n = len(arr)
i = 1
whilei< n and arr[i] <= x:
i = i * 2
returnbinary_search(arr, i // 2, min(i, n-1), x)
```

Example

```
arr = [2, 3, 4, 10, 40]
x = 10
result = exponential_search(arr, x)
if result != -1:
print(f"Element found at index {result}")
else:
print("Element not found")
```

1.4 IMPLEMENTATION OF SIMPLE ALGORITHMS

In this section of the **Design and Analysis of Algorithms Lab**, we'll dive into implementing fundamental algorithms. This includes **Euclid's algorithm** for finding the Greatest Common Divisor (GCD), **polynomial evaluation using Horner's method**, various **exponentiation algorithms**, and basic **sorting techniques**.

Key Algorithms Covered

- **Euclid's Algorithm for GCD:** You'll learn how to efficiently compute the greatest common divisor of two numbers by repeatedly applying the modulus operation. This iterative process finds the largest number that divides two integers without leaving a remainder.
- **Horner's Method for Polynomial Evaluation:** This method optimizes polynomial computation by significantly reducing the number of multiplicative operations through nested multiplication and addition.
- **Exponentiation Algorithms:** We'll explore efficient ways to compute powers, utilizing either recursive or iterative approaches.
- **Simple Sorting Algorithms (e.g., Selection Sort):** You'll study algorithms like Selection Sort, which sorts an array by iteratively identifying the smallest element and placing it in its correct position. This demonstrates foundational principles of sorting methodologies and how to incrementally sort an entire array.

Learning Outcomes

Each algorithm will be implemented and thoroughly analyzed. This hands-on approach will deepen your understanding of **algorithmic efficiency**, **performance characteristics**, and

their **applicability in solving computational challenges**. This section focuses on equipping you with foundational algorithms that serve as essential building blocks in computational problem-solving.

1.5 TASK SCHEDULING ALGORITHMS

Within the Design and Analysis of Algorithms Lab, a task scheduling problem is framed as an optimization challenge. The primary goal is to maximize profit by carefully selecting and scheduling tasks so they are completed within their specified deadlines.

The problem involves identifying a subset of tasks from a given collection and arranging them in a sequence that respects each task's deadline, all while aiming for the highest possible total profit. This constitutes a maximization optimization problem, inherently constrained by the absolute requirement that all chosen tasks must be finished by their designated deadlines. The lab focuses on investigating algorithms and strategies designed to efficiently solve these types of optimization problems, ensuring strict adherence to deadline constraints.

1.6 HUFFMAN'S CODING ALGORITHMS

In the Design and Analysis of Algorithms Lab, Huffman coding is explored as a greedy algorithm specifically designed for data compression. This method aims to efficiently reduce data size, particularly for sequences of characters with varying frequencies.

Huffman coding typically achieves compression rates of 70% to 80%. The process begins by assessing the frequency of each character within the dataset. Based on these frequencies, a Huffman tree is constructed to generate an optimal binary representation for every character. This technique, also known as variable-length coding, assigns shorter codes to characters that appear more frequently and longer codes to those that appear less often. This strategic assignment ultimately minimizes the total storage or transmission space required for the data.

1.7 DIVIDE AND CONQUER ALGORITHMS

In the Design and Analysis of Algorithms Lab, the Divide and Conquer approach is a core strategy. It involves recursively breaking down a complex problem into smaller, more manageable sub-problems. This division continues until each sub-problem is simple enough to be solved directly. Each of these sub-problems represents a smaller, more accessible piece of the original problem's complexity.

Once solved, the solutions to these individual sub-problems are then combined or merged

to construct the complete solution for the initial, larger problem. This methodical approach effectively uses recursion to efficiently tackle intricate problems by dissecting them into simpler parts and then synthesizing the results for a comprehensive solution.

1.8 SINGLE SOURCE SHORTEST PATH ALGORITHMS

In the Design and Analysis of Algorithms Lab, Dijkstra's algorithm is utilized to address the single-source shortest path problem, with the key constraint that all edge weights must be non-negative. This algorithm shares some methodological similarities with Prim's algorithm.

Dijkstra's algorithm operates by consistently choosing paths that are locally optimal at each step, prioritizing immediate efficiency. It iteratively explores vertices and updates the shortest known path from the source vertex to every other vertex. This progressive refinement ensures that the shortest paths are accurately determined for all vertices, ultimately providing an efficient solution to the shortest path problem in graphs where edge weights are non-negative.

1.9 MINIMUM COST SPANNING TREE

In the Design and Analysis of Algorithms Lab, a connected subgraph (S) of a graph (G(V, E)) is defined as a spanning tree if it encompasses all the vertices of (G) and has the minimum possible total weight of edges from (G). A crucial characteristic of a spanning tree is that it must be acyclic, meaning it contains no cycles.

Among all the possible spanning trees that can be formed from a graph, the one with the absolute lowest total edge weight is known as the Minimum Spanning Tree (MST). This concept is fundamental in optimizing network design, as it ensures that all vertices are connected using the least possible edge weight, while also preventing the formation of redundant loops or cycles within the network.

1.10 IMPLEMENTATION OF BINOMIAL COEFFICIENT PROBLEM

In the Design and Analysis of Algorithms Lab, implementing the binomial coefficient problem is a crucial exercise. It helps in understanding both combinatorial mathematics and dynamic programming. The binomial coefficient, often written as $ \binom{n}{k} $ or $ C(n, k) $, tells us the number of ways to choose $ k $ elements from a set of $ n $ elements without considering the order.

This problem can be solved using a recursive method based on Pascal's identity, with base

cases $ \binom{n}{0} = 1 $ and $ \binom{n}{n} = 1 $. While this recursive approach is intuitive, it becomes highly inefficient for large values of $ n $ and $ k $ due to its exponential time complexity.

Dynamic Programming for Efficiency

To overcome this inefficiency, the lab explores dynamic programming as a more effective solution. By storing intermediate results in a two-dimensional array, the dynamic programming approach significantly reduces the time complexity to $ O(nk) $ and the space complexity to $ O(k) $ if further optimized.

This method systematically builds the solution from the base cases, ensuring that each subproblem is solved only once. The implementation involves initializing a table where each entry represents $ \binom{i}{j} $, and then filling the table using the recursive relation $ \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} $. This hands-on practice helps students grasp the power of dynamic programming in optimizing combinatorial problems and reinforces their understanding of algorithmic efficiency.

1.11 EXPERIMENTS

1. Develop a program to determine the **operation count** for a given pseudocode.
2. Implement **Bubble Sort** for any provided list of numbers.
3. Implement **Insertion Sort** for any provided list of numbers.
4. Write a program to perform **Quick Sort** on a given list of integer values.
5. Create a program to find the **maximum and minimum** values within a given set of integers.
6. Implement **Merge Sort** on two given lists of integer values.
7. Develop a program to perform **Binary Search** on a given set of integer values, demonstrating both recursive and non-recursive approaches.
8. Write a program to solve the **knapsack problem** using the **greedy method**.
9. Implement **Prim's Algorithm** to find the minimum cost spanning tree.
10. Implement **Kruskal's Algorithm** to find the minimum cost spanning tree.
11. Create a program to solve the **single-source shortest path problem** for a given graph.
12. Write a program to find a solution for the **job sequencing with deadlines problem**.
13. Develop a program to solve the **all-pairs shortest path problem**.
14. Implement a program to solve the **N-QUEENS problem**.
15. Write a program to find a solution for the **sum of subsets problem** for a given set of distinct numbers.

Experiment No.1: Develop a program to determine the operation count for a given pseudocode.

```
 #include<stdio.h>
#include<conio.h>
void main()
{
int count=0,sum=0,n,i,a[50];
clrscr();
count=count+1;
printf("\n Enter the n value:");
scanf("%d",&n);
count=count+1;
printf("\n Enter %d values to sum:",n);
for(i=0;i<n;i++)
{
count=count+1;
scanf("%d",&a[i]);
}
count=count+1;
for(i=0;i<n;i++)
  {
count=count+1;
sum=sum+a[i];
count=count+1;
}
count=count+1;
printf("\n The of %d values is:%d and count is=%d",n,sum,count);
getch();
}
```

Output:

```
Enter the n value:5

 Enter 5 values to sum:1 2 3 4 5
 The of 5 values is:15 and count is=19
```

Experiment No.2: Implement Bubble Sort for any provided list of numbers.

```
#include<stdio.h>
#include<conio.h>
voidbubblesort(int[],int);
void display(int[],int);
int main()
{
        int a[20],n,i;
        clrscr();
        printf("\n Enter the number of elements in array are:");
        scanf("%d",&n);
        printf("\n Enter %d elements in the array:",n);
        for(i=0;i<n;i++)
        scanf("%d", &a[i]);
        bubblesort(a,n);
        printf("\n The sorted elements in the array are:");
        display(a,n);
        getch();
        return();
        }
        voidbubblesort(int a[],int n)
        {
        inti,j,temp,excg=0;
        int last=n-1;
        for(i=0;ia[j+1])
        {
        temp=a[j];
        a[j]=a[j+1];
        a[j+1]=temp;
        excg++;
        }
        }
        }
        if(excg==0)
        return ;
        else
```

```
last=last-1;
}
Void display (int a[], int n)
{
inti;
for (i=0;i<n;i++)
printf("%d\t", a[i]);
}
```

Output:

```
Enter the number of elements in array are:7

 Enter 7 elements in the array:7 8 9 5 4 2 1

 The sorted elements in the array are:1     2   4   5   7   8   9
```

Experiment No.3: Implement Insertion Sort for any provided list of numbers.

```
#include<stdio.h>
#include<conio.h>
voidinssort(int[],int);
void display(int[],int);
int main()
{
int a[20],n,i;
clrscr();
printf("\n Enter the number of elements in array are:");
scanf("%d",&n);
printf("\n Enter %d elements in the array:",n);
for(i=0;i<n;i++)
scanf("%d", &a[i]);
inssort(a,n);
printf("\n The sorted elements in the array are:");
display(a,n);
getch();
return 0;
}
voidinssort(int a[],int n)
```

```
{
inti,j,index=0;
for(i=1;i<n;i++)
{
Index=a[i];
j=1;
while((j>0)&&(a[j-1]>index))
{
a[j]=a[j-1];
j--;
}
a[j]=index;
}
}
void display(int a[],int n)
{
inti;
for(i=0;i<n;i++)
{
Printf("%d\t",a[i]);
}
}
```

Output:

```
Enter the number of elements in array are:7

 Enter 7 elements in the array:33 56 98 12 34 9 4

 The sorted elements in the array are:4  9  12  33  34  56  98
```

Experiment No.4: Write a program to perform Quick Sort on a given list of integer values.

```
#include<stdio.h>
#include<conio.h>
voidqsort(int [],int,int);
```

```
int partition(int [],int,int);
voidqsort(int a[],intfirst,int last)
{
int j;
if(first<last)
{
j=partition(a,first,last+1);
qsort(a,first,j-1);
qsort(a,j+1,last);
}
}
int partition(int a[], int first, int last)
{
int v=a[first];
inti=first;
int j=last;
int temp=0;
do
{
do
{
i++;
}
while(a[i]<v);
do
{
j--;
}
while(a[j]>v);
if(i<j)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
while(i>j);
```

```
a[first]=a[j];
a[j]=v;
return j;
}
int main()
{
int a[40],i,n;
clrscr();
printf("\n Enter the no of elements (size):");
scanf("%d",&n);
printf("\n Enter the ELements to sort:");
for(i=0;i<n;i++)
scanf("%d", &a[i]);
qsort(a,0,n-1);
printf("\n The ELements after sorting are:");
for(i=0;i<n;i++)
{
Printf("%d\t", a[i]);
}
getch();
return();
}
```

Output:

```
Enter the no of elements (size):6

 Enter the ELements to sort:98 45 21 34 90 43

 The ELements after sorting are:21  34  43  45  90  98
```

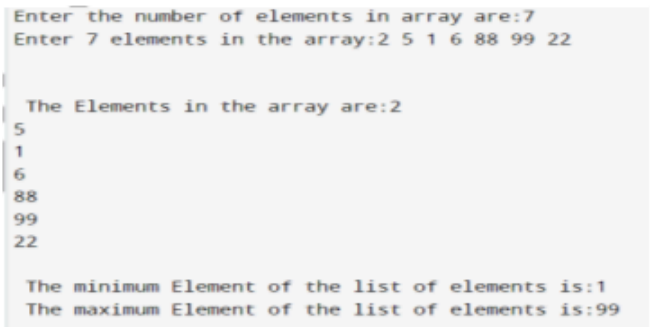
Experiment No.5: Create a program to find the maximum and minimum values within a given set of integers.

```
#include<stdio.h>
#include<conio.h>
voidminmax(int,int,int,int);
inti,j,a[50],n,fmax,fmin;
int main()
```

```
{
clrscr();
printf("\n Enter the number of elements in array are:");
scanf("%d",&n);
printf("\n Enter %d elements in the array:",n);
for(i=0;i<n;i++)
printf(“%d\n”, a[i]);
//fmax=fmin=a[0];
minmax(0, n-1, a[0],a[0]);
printf("\n The minimum Element of the list of elements is:%d",fmin);
printf("\n The maximum Element of the list of elements is:%d",fmax);
getch();
return 0;
}
voidminmax(inti,intj,intmax,int min)
{
intgmax,gmin,hmax,hmin;
gmax=hmax=max;
gmin=hmin=min;
if(i==j)
{
fmax=fmin=a[i];
}
else if(i==(j-1))
{
If(a[i]>a[j])
{
fmax=a[i];
fmin=a[j];
}
else
{
fmax=a[j];
fmin=a[i];
}
}
else
```

```
{
int mid=(i+j)/2;
minmax(i,mid,a[i],a[i]);
gmax=fmax;
gmin=fmin;
minmax(mid+1,j,a[mid+1],a[mid+1]);
hmax=fmax;
hmin=fmin;
if(gmax>hmax)
{
fmax=gmax;
}
else
{
fmax=hmax;
}
if(gmin>hmin)
{
fmin=hmin;
}
else
{
fmin=gmin;
}
}
}
```

Output:



Experiment No.6: Implement Merge Sort on two given lists of integer values.

```
#include<stdio.h>
#include<conio.h>
void merge(int[],int,int,int);
voidmergesort(int[], int,int);
void merge(int a[25], int low, int mid, int high)
{
int b[25],h,i,j,k;
h=low;
i=low;
j=mid+1;
while((h<=mid)&&(j<=high))
{
If(a[h]<a[j])
  {
b[i]=a[h];
h++;
 }
else
 {
b[i]=a[j];
j++;
}
if(h>mid)
{
for(k=j;k<=high;k++)
{
b[i]=a[k];
i++;
}
}
else
{
for(k=h;k<=mid;k++)
{
b[i]=a[k];
```

```
i++;
}
}
for(k=low;k<=high;k++)
{
a[k]=b[k];
}
}
voidmergesort(int a[25],intlow,int high)
{
int mid;
if(low<high)
{
mid=(low+high)/2;
mergesort(a,low,mid);
mergesort(a,mid+1,high);
merge(a, low,mid,high);
}
}
void main()
{
int a[25],i,n;
clrscr();
printf("\n Enter the size of the elements to be sorted:");
scanf("%d",&n); printf("\n Enter the elements to sort:");
for(i=0;i<n;i++)
scanf(“d”, &a[i]);
printf("\n The Elements before sorting are:");
for(i=0;i<n;i++)
printf(“%d\t”, a[i]);
mergesort(a, 0, n-1);
printf("\n The elements after sorting are:");
for(i=0;i<n;i++)
printf(“%d\t”, a[i]);
getch();
  }
```

Output:

```
Enter the size of the elements to be sorted:7

Enter the elements to sort:33 44 77 22 11 0 9

The Elements before sorting are:33 44  77  22  11  0   9
The elements after sorting are:0   9   11  22  33  44  77
```

Experiment No.7: Develop a program to perform Binary Search on a given set of integer values, demonstrating both recursive and non-recursive approaches.

```
#include<stdio.h>
#include<conio.h>
voidbubblesort(int[],int);
intbinsrch(int[],int,int,int);
void display(int[],int);
inti,j;
int main()
{
int a[20],n,key,pos=-1;
clrscr();
printf("\n Enter the number of elements in array are:");
scanf("%d",&n);
printf("\n Enter %d elements in the array:",n);
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("\n Enter the element to be searched:");
scanf("%d",&key);
bubblesort(a,n);
printf("\n The sorted elements in the array are:");
display(a,n);
pos=binsrch(a,key,0,n-1);
if(pos!=-1)
printf("\n The Element %d is found in position %d",key,pos);
else
printf("\n Element not found");
getch();
return 0;
```

```
}
intbinsrch(int a[],intkey,intlow,int high)
{
int mid;
while(low<=high)
{
mid=(low+high)/2;
if(keya[mid])
high=mid-1;
else if(key>a[mid])
low=mid+1;
else
return mid;
}
return -1;
}
voidbubblesort(int a[],int n)
{
inti,j,temp,excg=0;
int last=n-1;
for(i=0;j<last;j++)
  {
If(a[j]>a[j+1])
   {
temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
excg++;
}
}
}
if(excg==0)
return ;
else
last=last-1;
}
void display(int a[],int n)
```

```
{
inti;
for(i=0;i<n;i++)
printf("%d\t", a[i]);
  }
```

Output:

1.
```
Enter the number of elements in array are:7
Enter 7 elements in the array:1 7 9 3 5 2 0
Enter the element to be searched:7
The sorted elements in the array are:0  1   2   3   5   7   9
The Element 7 is found in position 5
```

2.
```
Enter the number of elements in array are:7
Enter 7 elements in the array:3 5 7 9 1 2 6
Enter the element to be searched:55

 The sorted elements in the array are:1 2   3   5   6   7   9
 Element not found
```

Experiment No.8: Write a program to solve the knapsack problem using the greedy method.

```
#include<stdio.h>
#include<conio.h>
voidreadf();
void knapsack(int,int);
voiddsort(int n);
void display(int);
int p[20],w[20],n,m;
double x[20],d[20],temp,res=0.0,sum=0.0;
voidreadf()
{
intm,n,i;
printf("\n Enter the no of Profits and weights:");
scanf("%d",&n);
printf("\n Enter the Maximum Capacity of the Knapsack:");
```

```
scanf("%d",&m);
printf("\n Enter %d profits of the weights:",n);
for(i=0;i<n;i++)
scanf(“%d”, &p[i]);
printf(“\n Enter %d Weight:”, n);
for(i=0;i<n;i++)
scanf(“%d”, &w[i]);
for(i=0;i<n;i++);
d[i]=(double)p[i]/w[i];
dsort(n);
knapsack(m,n);
display(n);
}
voiddsort(int n)
{
inti,j,t;
for(i=0;i<n;j++)
{
If(d[j]<d[j+1]);
d[j+1]=temp;
t=p[j];
p[j]=p[j+1];
p[j+1]=t;
t=w[j];
w[j]=w[j+1];
w[j+1]=t;
}
}
}
}
void display(int n)
{
inti,m;
printf("\n The Required Optimal solution is:\n");
printf("Profits Weights Xvalue\n");
for(i=0;i<n;i++)
{
```

```
printf("%d\t%d\t%f\n",p[i],w[i],x[i]);
sum=sum+(p[i]*x[i]);
res=res+(w[i]*x[i]);
}
printf("\n The Total Resultant Profit is:%f",sum);
printf("\n The total resultant Weight into the knapsack is:%f",res);
}
void knapsack(intm,int n)
{
inti,cu=m;
for(i=0;i<n;i++)
{
inti,cu=m;
for(i=0;i<n;i++)
{
X[i]=0.0;
}
for(i=0;i<n,i++)
{
if(w[i]<=n)
{
If(w[i]<cu)
 {
 X[i]=1.0;
cu=cu-w[i];
 }
else
break;
 }
If(i<=n)
 {
x[i]=(double)cu/w[i];
}
}
int main()
 {
clrscr();
```

```
readf();
getch();
return 0;
}
```

Output:

```
 Enter the no of Profits and weights:3

 Enter the Maximum Capacity of the Knapsack:20

 Enter 3 profits of the objects:25 24 15

 Enter 3 Weights:18 15 10

 The Required Optimal solution is:
Profits Weights Xvalue
24      15      1.000000
15      10      0.500000
25      18      0.000000

 The Total Resultant Profit is:31.500000
 The total resultant Weight into the knapsack is:20.000000
```

Experiment No.9: Implement Prim's Algorithm to find the minimum cost spanning tree.

```
#include<stdio.h>
#include<conio.h>
intn,cost[10][10],temp,nears[10];
voidreadv();
voidprimsalg();
voidreadv()
{
inti,j;
printf("\n Enter the No of nodes or vertices:");
scanf("%d",&n);
printf("\n Enter the Cost Adjacency matrix of the given graph:");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if((cost[i][j]==0) && (i!=j))
{
```

```
cost[i][j]=999;
}
}
}
}
voidprimsalg()
{
intk,l,min,a,t[10][10],u,i,j,mincost=0;
min=999;
for(i=1;i<=n;i++)              //To Find the Minimum Edge E(k,l)
{
for(u=1;u<=n;u++)
{
if(i!=u)
{
if(cost[i][u]<min)
 {
min=cost[i][u];
 k=i;
 l=u;
    }
  }
 }
}
t[1][1]=k;
t[1][2]=l;
printf("\n The Minimum Cost Spanning tree is...");
printf("\n(%d,%d)-->%d",k,l,min);
for(i=1;i<=n;i++)
{
if(i!=k)
{
if(cost[i][l]<cost[i][k])
{
nears[i]=l;
}
else
```

```
{
nears[i]=k;
}
}
}
nears[k]=nears[l]=0;
mincost=min;
for(i=2;i<=n-1;i++)
{
j = findnextindex(cost,nears);
t[i][1]=j;
t[i][2]=nears[j];
printf("\n(%d,%d)-->%d",t[i][1],t[i][2],cost[j][nears[j]]);
mincost=mincost+cost[j][nears[j]];
nears[j]=0;
for(k=1;k<=n;k++)
{
if(nears[k]!=0 && cost[k][nears[k]]>cost[k][j])
{
nears[k]=j;
}
}
}
printf("\n The Required Mincost of the Spanning Tree is:%d",mincost);
}
intfindnextindex(int cost[10][10],int nears[10])
{
int min=999,a,k,p;
for(a=1;a<=n;a++)
{
p=nears[a];
if(p!=0)
{
if(cost[a][p]<min)
  {
   Min=cost[a][p];
   K=a;
```

```
  }
  }
  }
  Return k;
  }
void main()
  {
clrscr();
readv();
primsalg();
  }
```

Output:

```
 Enter the No of nodes or vertices:7

 Enter the Cost Adjacency matrix of the given graph:0 28 0 0 0 10 0
28 0 16 0 0 0 14
0 16 0 12 0 0 0
0 0 12 0 22 0 18
0 0 0 22 0 25 24
10 0 0 0 25 0 0
0 14 0 0 24 0 0

 The Minimum Cost Spanning tree is...
(1,6)-->10
(5,6)-->25
(4,5)-->22
(3,4)-->12
(2,3)-->16
(7,2)-->14
 The Required Mincost of the Spanning Tree is:99
```

Experiment No.10: Implement Kruskal's Algorithm to find the minimum cost spanning tree.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
inti,j,k,a,b,u,v,n,ne=1;
intmin,mincost=0,cost[9][9],parent[9];
int find(int);
intuni(int,int);
void main()
{
clrscr();
printf("\n\tImplementation of Kruskal's algorithm\n");
```

```
printf("\nEnter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the cost adjacency matrix:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
} }
printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
{
for(i=1,min=999;i<=n;i++)
{
for(j=1;j <= n;j++)
{
if(cost[i][j] < min)
{
min=cost[i][j];
a=u=i;
b=v=j;
}
}
}
u=find(u);
v=find(v);
if(uni(u,v))
{
printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
mincost +=min;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
getch();
```

```
}
int find(inti)
{
while(parent[i])
i=parent[i];
returni;
}
intuni(inti,int j)
{
if(i!=j)
{
parent[j]=i;
return 1;
}
return 0;
}
```

Output:

```
          Implementation of Kruskal's algorithm

 Enter the no. of vertices:7

 Enter the cost adjacency matrix:
 0 28 0 0 0 10 0
 28 0 16 0 0 0 14
 0 16 0 12 0 0 0
 0 0 12 0 22 0 18
 0 0 0 22 0 25 24
 10 0 0 0 25 0 0
 0 14 0 0 24 0 0
 The edges of Minimum Cost Spanning Tree are
 1 edge (1,6) =10
 2 edge (3,4) =12
 3 edge (2,7) =14
 4 edge (2,3) =16
 5 edge (4,5) =22
 6 edge (5,6) =25

         Minimum cost = 99
```

Experiment No.11: Create a program to solve the single-source shortest path problem for a given graph.

```
#include<stdio.h>
#include<conio.h>
voidreadf();
void SP();
```

```
int cost[20][20],dist[20],s[20];
intn,u,min,v,w;
voidreadf()
{
inti,j;
printf("\n Enter the no of vertices:");
scanf("%d",&n);
printf("\n Enter the Cost of vertices:");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
}
}
void SP()
{
inti,j;
printf("\n Enter the source vertex:");
scanf("%d",&v);
for(i=1;i<=n;i++)
{
s[i]=0;
dist[i]=cost[v][i];
}
s[v]=1;
dist[v]=0;
for(i=2;i<=n;i++)
{
min=dist[i];
for(j=2;j<=n;j++)
{
if(s[j]==0)
{
```

```
if(min>dist[j])
{
min=dist[j];
u=j;
}
}
}
s[u]=1;
for(w=1;w<=n;w++)
{
if(cost[u][w]!=0 && s[w]==0)
{
if(dist[w]>(dist[u]+cost[u][w]))
{
dist[w]=dist[u]+cost[u][w];
}
}
}
}
printf("\n From the Source vertex %d",v);
for(i=1;i<=n;i++)
printf("\n%d->%d",i,dist[i]);
}
void main()
{
clrscr();
readf();
SP();
getch();
}
```

Output:

```
Enter the no of vertices:6
Enter the Cost of vertices:
0 50 45 10 0 0

0 0 10 15 0 0

0 0 0 0 30 0

20 0 0 0 15 0

0 20 35 0 0 0

0 0 0 0 3 0
```

```
 Enter the source vertex:1

 From the Source vertex 1
1->0
2->45
3->45
4->10
5->25
6->999
```

Experiment No.12: Write a program to find a solution for the **job sequencing with deadlines problem**.

```
#include<stdio.h>
#include<conio.h>
intjobseq();
voidpsort();
inttp,j[10],d[10],p[10],n;
void main()
{
inti,k;
clrscr();
printf("\n Enter the n'o of jobs:");
scanf("%d",&n);
printf("\n Enter the %d Deadlines for the jobs:",n);
for(i=1;i<=n;i++)
scanf("%d",&d[i]);
printf("\n Enter the Profits required for jobs:");
```

```
for(i=1;i<=n;i++)
scanf("%d",&p[i]);
psort();
for(i=1;i<=k;i++)
{
tp=tp+p[j[i]];
printf("%d-->",j[i]);
}
printf("\n Profits:%d",tp);
getch();
}
intjobseq()
{
inti,k,q;
d[0]=0;
j[0]=0;
j[1]=1;
k=1;
for(i=2;i<=n;i++)
{
int r=k;
while((d[j[r]]>d[i]) && (d[j[r]]!=r))
r=r-1;
if((d[j[r]]<=d[i]) && (d[i]>r))
{
for(q=k;q>=r+1;q--)
{
j[q+1]=j[q];
}
j[r+1]=i;
k=k+1;
}
}
return k;
}
voidpsort()
{
```

```
int i,k,temp1;
for(i=1;i<=n;i++)
{
for(k=1;k<=n-i;k++)
{
if(p[k]<p[k+1])
{
temp1=p[k];
p[k]=p[k+1];
p[k+1]=temp1;
temp1=j[k];
j[k]=j[k+1];
j[k+1]=temp1;
temp1=d[k];
d[k]=d[k+1];
d[k+1]=temp1;
}
}
}
}
```

Output:

1.
```
Enter the n'o of jobs:9

 Enter the 9 Deadlines for the jobs:5 3 2 2 3 4 7 7 5
 Enter the Profits required for jobs:30 25 23 20 18 18 16 15 10
 The Required Solution is:3-->4-->2-->6-->1-->7-->8-->
 Profits:147
```

2.
```
Enter the n'o of jobs:5
Enter the 5 Deadlines for the jobs:2 2 1 1 3
Enter the Profits required for jobs:100 90 70 50 40

 The Required Solution is:1-->2-->5-->
 Profits:230
```

Experiment No.13: Develop a program to solve the all-pairs shortest path problem.

```
#include<stdio.h>
#include<conio.h>
voidreadf();
voidamin();
int cost[20][20],a[20][20];
inti,j,k,n;
voidreadf()
{
printf("\n Enter the no of vertices:");
scanf("%d",&n);
printf("\n Enter the Cost of vertices:");
for(i=0;i<n;j++)
{
scanf(“”,&cost[i][j]);
if(cost[i][j]==0 && (i!=j))
cost[i][j]=999;
a[i][j]=cost[i][j];
}
} }
Void main()
{
for(k=0;k<n;k++)
{
for(i=0:i<n;i++)
{
for(j=0;j<n;J++)
{
if(a[i][k]+a[k][j])
{
a[i][j]=a[i][k]+a[k][j];
} }
}
}
printf("\n The All pair shortest path is:");
for(i=0;i<n;i++)
```

```
{
printf(“\n”);
for(j=0;j<n;j++)
{
Printf(“%d\t”,a[i][j]);
}
} }
void main()
{
clrscr();
readf();
amin();
getch();
}
```

Output:

```
Enter the no of vertices:3
Enter the Cost of vertices:
0 4 11
6 0 2
3 0 0
The All pair shortest path is:
0   4   6
5   0   2
3   7   0
```

Experiment No.14: Implement a program to solve the N-QUEENS problem.

```
#include<stdio.h>
#include<conio.h>
 #include<math.h>
voidreadv();
voidnqueen(int,int);
int place(int,int);
int x[25],count=0;
voidreadv()
{
int n;
printf("\n Enter the no of Queens to be placed:");
```

```
scanf("%d",&n);
printf("\n The Places in which the %d Queens are toplaced in the %dx%dChessBoard is:",n,n);
nqueen(1,n);
printf("\n The No of Solutions for the %d Queens Problem are:%d",n,count);
}
voidnqueen(intk,int n)
{
inti,j;
for(i=1;i<=n;i++)
{
if(place(k,i))
{
x[k]=i;
if(k==n)
 {
count++;
if(count%10 == 0)
getch();
printf("\n");
for(j=1;j<=n;j++)
{
printf("%d\t",x[j]);
 }
}
else
{
nqueen(k+1,n);
 } }
} }
int place(intk,inti)
{
int j;
for(j=1;j<=k-1;j++)
{
if((x[j]==i)||(abs(x[j]-i)==abs(j-k)))
{
return 0;
```

```
} }
return 1;
}
void main()
{
clrscr();
readv();
getch();
}
```

Output:

```
Enter the no of Queens to be placed:4

 The Places in which the 4 Queens are to placed in the 4 x 4 ChessBoard is:
2  4   1   3
3  1   4   2
 The No of Solutions for the 4 Queens Problem are:2
```

Experiment No.15: Write a program to find a solution for the sum of subsets problem for a given set of distinct numbers.

```
#include<stdio.h>
#include<conio.h>
void Sum ofSub(int,int,int);
int x[25],n,m=0,sum=0,w[25];
voidreadf()
{
inti;
printf("\n Enter the no of values in the set:");
scanf("%d",&n);
printf("\n Enter the %d weights of the values in the set:",n);
for(i=1;i<=n;i++)
{
scanf("%d",&w[i]);
sum=sum+w[i];
x[i]=0;
}
printf("\n Enter the required sum of the values in the subset:");
scanf("%d",&m);
```

```
printf("\n The Total sum of the weights is:%d",sum);
SumofSub(0,1,sum);
}
voidSumOfSub(ints,intk,int r)
{
inti,j;
x[k]=1;
if(sum>=m)
{
if(s+w[k]==m)
{
printf("\n");
for(j=1;j<=n;j++)
{
printf("%d\t",x[j]);
}
printf("\n-->");
for(j=1;j<=k;j++)
{
if(x[j] == 1)
printf("%d\t",w[j]);
} }
else
if(s+w[k]+w[k+1]<=m)
SumofSub(s+w[k],k+1,r-w[k]);
if((s+r-w[k]>=m) && (s+w[k+1]<=m))
{
x[k]=0;
SumofSub(s,k+1,r-w[k]);
} }
else
{
printf("\n No Solutions Available because sum of all weights is %d less than required sum
%d",sum,m);
} }
void main()
{
```

```
clrscr();
readf();
getch();
}
```

Output:

```
Enter the no of values in the set:6
Enter the 6 weights of the values in the set:5 10 12 13 15 18

 Enter the required sum of the values in the subset:30

 The Total sum of the weights is:73
1   1   0   0   1   0
-->5    10  15
1   0   1   1   1   0
-->5    12  13
0   0   1   0   0   1
-->12   18
```

SESSION 2 WEB DESIGN LAB

Basic Structure

2.0 INTRODUCTION

This session introduces you to the vital role of **Web design labs** in bridging the gap between theoretical knowledge and practical application in web development. These labs offer a hands-on environment where students and professionals can actively experiment with various web design principles and technologies. By engaging in real-world projects and exercises, participants gain a deeper understanding of web design intricacies, including **user interface (UI)** and **user experience (UX) design**, **responsive layouts**, and **accessibility considerations**. This experiential learning approach ensures individuals are not only well-versed in the latest web design trends but also capable of implementing them effectively.

Tools and Resources for Web Design

In a web design lab, learners have access to a diverse array of tools and resources that facilitate website creation and testing. These range from graphic design software like Adobe Photoshop and Illustrator to robust coding environments such as Visual Studio Code. Additionally, web design labs often provide access to popular frameworks and libraries like Bootstrap and jQuery, which streamline the development process. By leveraging these resources, participants cultivate a comprehensive skill set encompassing both design aesthetics and technical proficiency, enabling them to build visually appealing and highly functional websites.

The Power of Collaboration

Collaboration is another crucial aspect of web design labs. Working in a collaborative setting empowers participants to share ideas, offer constructive feedback, and learn from each other's experiences. This teamwork is indispensable in web design, as diverse

perspectives often lead to more innovative and user-friendly solutions. Web design labs frequently simulate real-world scenarios where designers, developers, and clients must collaborate to achieve a common goal. This collaborative practice effectively prepares individuals for the dynamic and often team-oriented nature of the web design industry.

Staying Current with Web Technologies

Furthermore, web design labs play a pivotal role in keeping pace with the rapidly evolving landscape of web technologies. With new tools, techniques, and standards emerging constantly, staying updated is paramount for anyone involved in web design. Web design labs provide a structured environment for continuous learning and experimentation, ensuring participants remain at the forefront of the industry. By regularly engaging with the latest advancements, learners can develop adaptable skills and maintain their competitiveness in the job market.

In essence, web design labs are crucial for developing practical skills, fostering collaboration, and staying current with technological advancements in web design.

2.1 OBJECTIVES

After the completing this unit, you will be able to:

1. Develop a comprehensive understanding of UI and UX design principles.
2. Gain proficiency in using graphic design software and coding environments.
3. Master the implementation of responsive web design techniques.
4. Learn to create accessible and inclusive websites.
5. Utilize frameworks and libraries to streamline the web development process.
6. Enhance problem-solving skills through real-world web design projects.
7. Foster collaboration and teamwork among participants.
8. Stay updated with the latest web design trends and technologies.
9. Build a portfolio of professional-quality web design projects.
10. Prepare for a successful career in the web design industry by developing practical, hands-on experience.

2.2 INTRODUCTION TO WEB DESIGN

Web design is both an art and a science, focused on crafting websites that are not only visually appealing but also highly functional and user-friendly. At its core, this discipline relies on three fundamental technologies: **HTML, CSS, and JavaScript**. HTML (Hypertext Markup Language) establishes the structural foundation of a web page, defining its content and overall layout. CSS (Cascading Style Sheets) then enhances this structure by introducing styling elements like colors, fonts, and sophisticated layouts, all contributing to a captivating user experience. JavaScript, on the other hand, injects interactivity and dynamic behavior into web pages, making them responsive and engaging. Together, these technologies empower designers to build comprehensive web experiences that effectively meet the diverse needs of both users and clients.

Key Components of Web Design

a). Structure and Semantics with HTML

HTML serves as the essential building block of web design, providing the basic structure and semantic meaning of a web page. It utilizes a system of tags and attributes to organize content into fundamental elements such as headings, paragraphs, lists, links, and images. Understanding the correct usage of these HTML elements is crucial for creating well-structured and accessible web pages. The concept of **semantic HTML**, which involves using tags that clearly convey the meaning of the enclosed content, is particularly important for improving search engine optimization (SEO) and overall accessibility.

b). Styling and Layout with CSS

CSS plays a pivotal role in transforming the raw structure provided by HTML into a visually appealing and intuitive user interface. It allows designers extensive control over the presentation of web pages, including typography, color schemes, spacing, and overall layout. CSS is also vital for **responsive design**, ensuring that web pages seamlessly adapt to various screen sizes and devices, providing a consistent user experience. Key CSS concepts like the **box model, positioning, and flexbox/grid layouts** offer powerful tools for creating sophisticated and adaptable designs.

c). Interactivity and Dynamics with JavaScript

JavaScript brings interactivity and dynamic content to web pages, enabling features that significantly enhance user engagement and functionality. By manipulating the **Document Object Model (DOM)**, JavaScript can update content in real-time, respond to user events, validate forms, and create animations without requiring a full page reload. A solid grasp of JavaScript fundamentals, including variables, functions, and event handling, is essential for developing interactive web applications. Furthermore, advanced JavaScript libraries and frameworks such as jQuery, React, and Angular extend its capabilities, allowing for the creation of highly complex and performant web applications.

Integrating HTML, CSS, and JavaScript

Effective web design demands a seamless and harmonious integration of HTML, CSS, and JavaScript. These technologies work in concert to deliver a cohesive user experience, from the underlying structure and visual styling to interactive elements and dynamic functionality. In a web design lab setting, students and professionals actively practice this integration through hands-on projects that mimic real-world scenarios. This practical approach not only solidifies theoretical knowledge but also cultivates vital problem-solving skills and creative thinking, both of which are essential for a successful career in web design. By mastering the combined use of HTML, CSS, and JavaScript, designers can build robust and engaging websites that cater to diverse user needs.

2.2.1 Understanding Web Design Principles

Web design is both an art and a science, focused on crafting websites that are not only visually appealing but also highly functional and user-friendly. At its core, this discipline relies on three fundamental technologies: **HTML, CSS, and JavaScript**. HTML (Hypertext Markup Language) establishes the structural foundation of a web page, defining its content and overall layout. CSS (Cascading Style Sheets) then enhances this structure by introducing styling elements like colors, fonts, and sophisticated layouts, all contributing to a captivating user experience. JavaScript, on the other hand, injects interactivity and dynamic behavior into web pages, making them responsive and engaging. Together, these technologies empower designers to build comprehensive web experiences that effectively meet the diverse needs of both users and clients.

a). Structure and Semantics with HTML

HTML serves as the essential building block of web design, providing the basic structure and semantic meaning of a web page. It utilizes a system of tags and attributes to organize content into fundamental elements such as headings, paragraphs, lists, links, and images. Understanding the correct usage of these HTML elements is crucial for creating well-structured and accessible web pages. The concept of **semantic HTML**, which involves using tags that clearly convey the meaning of the enclosed content, is particularly important for improving search engine optimization (SEO) and overall accessibility.

b). Styling and Layout with CSS

CSS plays a pivotal role in transforming the raw structure provided by HTML into a visually appealing and intuitive user interface. It allows designers extensive control over the presentation of web pages, including typography, color schemes, spacing, and overall layout. CSS is also vital for **responsive design**, ensuring that web pages seamlessly adapt to various screen sizes and devices, providing a consistent user experience. Key CSS concepts like the **box model, positioning, and flexbox/grid layouts** offer powerful tools for creating sophisticated and adaptable designs.

c). Interactivity and Dynamics with JavaScript

JavaScript brings interactivity and dynamic content to web pages, enabling features that significantly enhance user engagement and functionality. By manipulating the **Document Object Model (DOM)**, JavaScript can update content in real-time, respond to user events, validate forms, and create animations without requiring a full page reload. A solid grasp of JavaScript fundamentals, including variables, functions, and event handling, is essential for developing interactive web applications. Furthermore, advanced JavaScript libraries and frameworks such as jQuery, React, and Angular extend its capabilities, allowing for the creation of highly complex and performant web applications.

d). Integrating HTML, CSS, and JavaScript

Effective web design demands a seamless and harmonious integration of HTML, CSS, and JavaScript. These technologies work in concert to deliver a cohesive user experience, from the underlying structure and visual styling to interactive elements and dynamic functionality. In a web design lab setting, students and professionals actively practice this integration through hands-on projects that mimic real-world scenarios. This practical approach not only solidifies theoretical knowledge but also cultivates vital problem-solving skills and creative thinking, both of which are essential for a successful career in web design. By mastering the combined use of HTML, CSS, and JavaScript, designers can build robust and engaging websites that cater to diverse user needs.

2.2.2 Importance of Web Design in the Digital Age

Web design is incredibly important in today's digital world. It's about creating websites that not only look great but are also easy to use and effective.

a). Enhancing User Engagement and Experience

A well-designed website is crucial for grabbing and holding user attention. It significantly boosts **user engagement** by providing an intuitive and enjoyable Browse experience. Features like simple navigation, fast loading times, and mobile responsiveness ensure users can access content smoothly across all their devices. By combining HTML for structure, CSS for styling, and JavaScript for interactivity, designers can create web pages that not only attract visitors but keep them engaged, reducing bounce rates and increasing the time they spend on the site.

b). Building Brand Identity and Credibility

Often, a website is the very first interaction a business has with potential customers. This makes its design vital for establishing a strong **brand identity** and **credibility**. Using consistent colors, fonts, and graphics that align with brand guidelines helps create a cohesive and professional appearance. Furthermore, incorporating interactive elements and modern design trends through CSS and JavaScript can convey innovation and attention to detail. A polished and functional website fosters trust and credibility, making visitors more inclined to become customers.

c). Impact on SEO and Online Visibility

Web design also heavily influences a site's **search engine optimization (SEO)** and overall **online visibility**. Correctly using HTML tags, such as headers, alt texts for images, and meta descriptions, helps search engines understand the site's content and context, leading to better indexing and higher rankings. CSS ensures the site is visually appealing and loads quickly, both of which are factors search engines consider when ranking pages. Additionally, JavaScript can improve user experience by enabling features

that boost user interaction and time spent on the site, indirectly enhancing SEO performance.

d). Facilitating Business Growth and Accessibility

In the digital marketplace, a well-designed website is a powerful engine for **business growth**. It serves as a comprehensive platform for marketing, sales, and customer engagement. By ensuring the site is **accessible** to a wide audience, including individuals with disabilities, businesses can reach more potential customers. HTML provides the foundational structure that screen readers and other assistive technologies rely on, while CSS and JavaScript can be used to enhance accessibility features without compromising the user experience. This inclusivity not only expands market reach but also ensures compliance with legal standards, contributing to a positive brand reputation.

By mastering the principles of HTML, CSS, and JavaScript, web designers can create sites that are not just visually stunning and highly functional, but also powerful tools for engagement, branding, SEO, and business growth. In a web design lab, learners can develop these essential skills through practical, hands-on experience, preparing them for the dynamic and ever-evolving field of web design.

2.3 TOOLS AND TECHNOLOGIES IN WEB DESIGN

Web design is both an art and a science, focused on crafting websites that are not only visually appealing but also highly functional and user-friendly. At its core, this discipline relies on three fundamental technologies: **HTML, CSS, and JavaScript**. HTML (Hypertext Markup Language) establishes the structural foundation of a web page, defining its content and overall layout. CSS (Cascading Style Sheets) then enhances this structure by introducing styling elements like colors, fonts, and sophisticated layouts, all contributing to a captivating user experience. JavaScript, on the other hand, injects interactivity and dynamic behavior into web pages, making them responsive and engaging. Together, these technologies empower designers to build comprehensive web experiences that effectively meet the diverse needs of both users and clients.

The Core Technologies of Web Design

a). HTML (Hypertext Markup Language)

HTML serves as the foundational technology in web design, providing the structure and content of a web page. It uses a system of tags and attributes to define various elements, such as headings, paragraphs, links, images, and multimedia content. By organizing content into a logical structure, HTML ensures that web pages are accessible and easy to navigate. Moreover, **semantic HTML**, which involves using tags that convey the meaning of the content, enhances search engine optimization (SEO) and accessibility,

making websites more discoverable and usable by a wider audience. Understanding HTML is crucial for any web designer, as it forms the basis upon which all other web technologies are built.

b). CSS (Cascading Style Sheets)

CSS is the technology that brings style and visual appeal to web pages. It allows designers to control the presentation of HTML elements, including layout, colors, fonts, and spacing. CSS enables the creation of **responsive designs** that adapt to different screen sizes and devices, ensuring a consistent user experience across desktops, tablets, and smart phones. Key features of CSS include the **box model**, which defines the space around elements, and advanced layout techniques such as **Flexbox and CSS Grid**, which provide powerful tools for creating complex and adaptable designs. By separating content (HTML) from presentation (CSS), designers can maintain cleaner code and more flexible design options.

c). JavaScript

JavaScript adds interactivity and dynamic functionality to web pages, making them more engaging and user-friendly. It allows designers to create real-time updates, form validations, interactive maps, animations, and other features that enhance user experience. JavaScript operates on the **client side**, meaning it runs directly in the user's browser, which reduces server load and increases the speed of web applications. Understanding JavaScript fundamentals, such as variables, functions, and event handling, is essential for creating interactive web applications. Additionally, modern JavaScript frameworks and libraries, such as React, Angular, and Vue.js, provide powerful tools for building complex, scalable, and maintainable web applications.

Essential Tools and Environments for Web Design

d). Integrated Development Environments (IDEs) and Tools

Effective web design requires the use of various tools and environments that streamline the development process. **Integrated Development Environments (IDEs)** like Visual Studio Code, Sublime Text, and Atom provide a robust platform for writing, testing, and debugging code. These tools often include features such as syntax highlighting, code completion, and version control integration, which enhance productivity and and reduce errors. Additionally, **graphic design software** like Adobe Photoshop and Illustrator are essential for creating and manipulating visual elements. **Version control systems** like Git enable collaborative work by tracking changes and managing code versions. These tools, when used in conjunction with HTML, CSS, and JavaScript, empower designers to create high-quality, professional web designs efficiently.

Specialized Design and Development Tools

1.1.1 Graphic Design Software

In addition to HTML, CSS, and JavaScript, **graphic design software** plays a crucial role in the web design process. Tools such as Adobe Photoshop, Illustrator, and Sketch are indispensable for creating and editing visual elements like logos, icons, and custom graphics. These programs allow designers to craft high-quality visuals that enhance the overall look and feel of a website. Photoshop is particularly useful for manipulating images and creating detailed graphics, while Illustrator excels in vector design, making it ideal for logos and scalable graphics. Sketch, popular among web designers, offers robust features for UI and UX design, including prototyping and collaboration tools. Proficiency in these graphic design tools enables designers to produce visually compelling and cohesive designs, ensuring that every element of a website aligns with the intended user experience and branding.

1.1.2 Coding Environments and Languages

A robust **coding environment** is essential for efficient web design and development, providing the tools and resources needed to write, test, and debug code effectively. Integrated Development Environments (IDEs) like **Visual Studio Code, Sublime Text, and Atom** are popular choices among web designers. These environments offer features such as syntax highlighting, code completion, and version control integration, which streamline the coding process and reduce errors. Additionally, familiarity with languages beyond HTML, CSS, and JavaScript, such as PHP, Python, and SQL, can enhance a designer's ability to build more complex and dynamic web applications. Mastery of these coding environments and languages is critical for web designers, allowing them to efficiently develop, test, and deploy high-quality web solutions.

2.3.3 Frameworks and Libraries

Frameworks and libraries are vital tools in web design, significantly enhancing the efficiency and capabilities of HTML, CSS, and JavaScript. Frameworks like **Bootstrap and Foundation** provide pre-designed UI components and responsive grid systems, allowing designers to quickly create uniform and mobile-friendly layouts. Libraries such as **jQuery** simplify JavaScript by offering easy-to-use functions for common tasks like DOM manipulation and event handling. More advanced libraries and frameworks like **React, Angular, and Vue.js** enable the development of complex, single-page applications with efficient data binding and state management. By leveraging these frameworks and libraries, web designers can streamline development processes, ensure consistency across projects, and focus more on creating unique and engaging user experiences.

In a web design lab setting, learners have the opportunity to master these tools and technologies through hands-on practice and real-world projects. This practical experience

is crucial for developing the technical skills and creative problem-solving abilities needed to succeed in the dynamic field of web design. By becoming proficient in HTML, CSS, JavaScript, and the various tools that support them, designers can create websites that are not only visually appealing but also functional, responsive, and user-friendly.

2.4 INTRODUCTION TO HTML

Understanding Websites and Web Pages

a). Websites and Web Pages

A **website** is essentially a collection of interconnected web pages, typically hosted on the same server. It functions as a cohesive unit of information managed by an individual, group, or organization. The **homepage** usually acts as the primary entry point to the site. Each individual **web page** within a website is a document crafted using **HTML (Hypertext Markup Language)** and becomes accessible on the internet when accessed via its unique web address. HTML is the foundational technology for building these pages, providing the structure and content that browsers interpret and display.

b). Types of Web Pages

i). Static Web Pages

Static web pages are delivered to the user exactly as they are stored on the web server. Their content is fixed and doesn't change in response to user interactions. These pages are simple to create and host, as they consist of basic HTML files without any server-side processing. Static web pages are ideal for content that doesn't require frequent updates or user input.

ii). Dynamic Web Pages

In contrast, **dynamic web pages** are generated in real-time by web applications driven by server-side software or client-side scripting. These pages can change content dynamically based on user interactions, such as form submissions or database queries. Dynamic web pages enhance the user experience by providing personalized and interactive content that evolves over time.

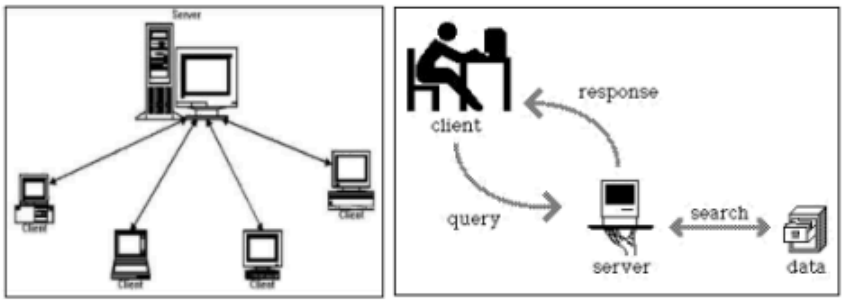
Web Browsers and the Client-Server Model

c). Browsers and Their Types

A **web browser** is a software application used to retrieve, present, and navigate information on the World Wide Web. Major web browsers include Google Chrome, Firefox, Microsoft Edge (formerly Internet Explorer), Opera, and Safari. Each browser interprets and displays HTML, CSS, and JavaScript in slightly different ways, making **cross-browser compatibility** an important consideration in web design.

d). Client-Server Model

The **client-server model** is a computing architecture that divides tasks between **servers** (providers of resources or services) and **clients** (requesters of those resources or services). In this model, clients and servers communicate over a network. Servers host web applications or data, while clients access these resources by initiating requests for content or services. This model is fundamental to web operations, enabling efficient and scalable interactions between users and web applications.



The Role of a Web Server

A **web server** can refer to either the physical hardware (a computer) or the software application that delivers web content over the Internet. While hosting websites is their primary function, web servers also support other applications such as online gaming, data storage, and enterprise software solutions. Web servers are crucial in web design because they store and serve the HTML, CSS, and JavaScript files that constitute web pages.

Different Categories of Web Pages

a). Advocacy Web Pages

Advocacy web pages are created and sponsored by organizations to influence public opinion or promote specific causes. These sites typically have URLs ending in .org and aim to raise awareness, support policy changes, or mobilize public action.

b). Business and Marketing Web Pages

Business and marketing web pages are designed by commercial enterprises to sell products or market services. These pages often feature e-commerce functionality, promotional content, and customer service information. Their URLs usually end in .com, reflecting their commercial nature.

c). News Web Pages

News web pages provide timely information about current events, issues, and developments. These sites are maintained by news organizations and journalists, offering updates on local, national, and international news.

d). Informational Web Pages

Informational web pages include reports, research findings, and educational content. These pages are often hosted by educational institutions, research organizations, or government entities, with URLs typically ending in .edu or .gov. They aim to provide accurate and detailed information on various topics.

e). Personal Web Pages

Personal web pages are created by individuals for personal use, such as blogs, portfolios, or online resumes. These pages often have a URL that includes a tilde (~) and reflect the interests and activities of the individual creator.

2.5 CASCADING STYLE SHEET (CSS)

Cascading Style Sheets (CSS) are an essential technology in web design, providing the means to control the appearance and layout of HTML elements on a web page. CSS enables designers to create visually appealing and cohesive designs by defining styles for elements like text, images, and containers. By separating content from presentation, CSS allows for more flexible and maintainable web designs. This separation means developers can change a website's look and feel without altering the underlying HTML

structure, which streamlines the design process and ensures consistency across multiple pages.

Responsive Design with CSS

One of the key advantages of CSS is its ability to create **responsive web designs**. Responsive design ensures that web pages look and function well on a variety of devices, from desktops to tablets and smartphones. **CSS media queries** allow designers to specify different styles for different screen sizes, orientations, and resolutions, creating a seamless user experience across devices. This adaptability is crucial in today's digital landscape, where users access websites from an ever-growing array of devices.

Powerful Layout Tools

CSS also offers powerful layout tools that enable designers to create complex and adaptive designs. **Flexbox and CSS Grid** are two advanced CSS features that provide flexible and efficient ways to arrange elements on a web page. Flexbox allows for the creation of one-dimensional layouts, making it easy to align and distribute space among items in a container. CSS Grid, on the other hand, supports two-dimensional layouts, allowing for more intricate and precise arrangements of elements. These tools empower designers to craft modern, visually appealing, and highly functional layouts that enhance the overall user experience.

Mastering CSS in the Lab

In a web design lab, mastering CSS is essential for creating professional and effective web pages. Through hands-on practice and real-world projects, learners can develop their skills in using CSS to style and lay out web pages. By understanding and applying CSS principles, designers can ensure their web pages are not only aesthetically pleasing but also responsive, accessible, and user-friendly. This proficiency in CSS, combined with knowledge of HTML and JavaScript, forms a solid foundation for successful web design and development.

2.6 JAVA SCRIPT HTML DOM

The **JavaScript HTML DOM (Document Object Model)** is a critical concept in web design and development, enabling dynamic interaction with the content and structure of web pages. The DOM essentially represents a web page as a hierarchical tree of objects, where each node corresponds to an HTML element. JavaScript can then manipulate these objects to dynamically change a web page's content, structure, and style. This manipulation allows developers to create interactive and responsive user interfaces, significantly enhancing the user experience by enabling real-time updates and interactions without requiring a full page reload.

Accessing and Modifying HTML Elements

One of the key features of the JavaScript HTML DOM is its ability to access and modify HTML elements and their attributes. Using methods such as getElementById, getElementsByClassName, and querySelector, developers can select specific elements and make changes to their content, style, or properties. This capability is essential for tasks like form validation, dynamic content updates, and creating interactive components such as image sliders, modal windows, and more. By leveraging the DOM, developers can create rich, interactive experiences that respond directly to user input and behavior.

Understanding Event Handling

Event handling is another crucial aspect of the JavaScript HTML DOM. Events are actions or occurrences that happen in the browser, such as clicks, key presses, or page loads. JavaScript can "listen" for these events and execute specific code in response, allowing for dynamic and interactive functionality. For example, a button click can trigger a function that changes the text of a paragraph, submits a form, or fetches data from a server. Understanding and implementing event handling is fundamental for creating responsive and interactive web applications, as it enables real-time interaction between the user and the web page.

Mastering DOM in the Lab

In a web design lab, mastering the JavaScript HTML DOM is essential for building dynamic and engaging web applications. Through practical exercises and projects, learners can develop their skills in manipulating the DOM to create interactive user interfaces. By understanding how to access and modify HTML elements, handle events, and update the DOM dynamically, designers can significantly enhance the functionality and responsiveness of their web pages. This expertise, combined with a strong knowledge of HTML and CSS, provides a comprehensive foundation for modern web development, enabling the creation of sophisticated and user-friendly web applications.

2.7 EXPERIMENTS

1. Design a page having suitable background colour and text colour with title “My First Web Page” using all the attributes of the Font tag.
2. Create a HTML document giving details of your [Name, Age], [Address, Phone] and [Register Number, Class] aligned in proper order using alignment attributes of Paragraph tag.
3. Write HTML code to design a page containing some text in a paragraph by giving suitable heading style.

4. Create a page to show different character formatting (B, I, U, SUB, SUP) tags. viz : log b m p = p logb m.
5. Write HTML code to create a Web Page that contains an Image at its centre.
6. Design a webpage featuring an image positioned on the left side. Configure this image so that clicking on it opens a completely different webpage in the browser.
7. Develop a webpage that utilizes anchor tags to create links to external websites. Ensure you use the appropriate HTML attributes to facilitate these external navigations.
8. Create a single webpage with multiple distinct sections. Implement internal links within this page so that when a user clicks on a link, the browser scrolls smoothly to the corresponding section on the *same* page.
9. Write a HTML code to create a web page with pink color background and display moving message in red color.
10. Create a web page, showing an ordered list of all second semester courses (Subjects) Procedure.
11. Create a web page, showing an unordered list of names of all the Diploma Programmes (Branches) in your institution.
12. Create a HTML document containing a nested list showing a content page of any book.
13. Create the following table in HTML with Dummy Data:

Reg. Number	**Student Name**	**Year/Semester**	**Date of Admission**

14. Create a web page which divides the page in two equal frames and place the audio and video clips in frame-1 and frame-2 respectively.

FRAME – I	**FRAME – II**

15. Create a web page which should generate following output:

FRAME-1	**FRAME-2**
	FRAME-3

CENTRE FOR DISTANCE AND ONLINE EDUCATION

MANGALAYATAN
UNIVERSITY

Learn Today to Lead Tomorrow

Extended NCR, 33rd Milestone, Aligarh-Mathura
Highway, Beswan, Aligarh, UP-202146

www.mangalayatan.in, www.mude.ac.in
cdoe@mangalayatan.edu.in