SSBT's College of Engineering and Technology , Bambhori, Jalgaon
Department of Computer Applications
Lab on Design and Analysis of an Algorithms

List

Practical No	Title
1.	Write a program for creating max./min. heap using INSERT.
2.	Write a program for creating max./min. heap using ADJUST/HEAPIFY.
3.	Write a program to implement union and find operation.
4.	Write a program to find minimum and maximum form a given array.
5.	Write a program for searching element form given array using binary search for n=1000,2000,3000 find exact time of execution.
6.	Write a program for sorting given array in ascending/descending order with n=1000,2000,3000
7.	find exact time of execution using Heap sort , Merge sort, Quick sort
8.	Write a program for matrix multiplication using Strassen's matrix multiplication.
9.	Write a program to find solution of Knapsack instant.
10.	Write a program to find shortest path using single source shortest path.
11.	Write a program to find Minimum-Cost Spanning Trees (Prim's & Kruskal's Algorithm).
12.	Write a program to find shortest path using all pair path.
13.	Write a program to find longest common subsequence.
14.	Write a program to implement breadth first traversal .
15.	Write a program to implement depth first traversal.
16.	Write a program to find all solutions for 8-queen problem using backtracking.

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 01

DOP: **DOC:**
Roll No:

Title: Write a program for creating max./min. heap using INSERT

1. **Objective:** To understand and Implement heap sort.
2. **Algorithm :**

For max_heap:

```
Begin
  Declare function max_heap ()
    Declare j, t of the integer datatype.
    Initialize t = a[m].
    j = 2 * m;
    while (j <= n) do
      if (j < n && a[j+1] > a[j]) then
        j = j + 1
      if (t > a[j]) then
        break
      else if (t <= a[j]) then
        a[j / 2] = a[j]
        j = 2 * j
    a[j/2] = t
    return
End.
```

For build_maxheap:

```
Begin
  Declare function build_maxheap(int *a,int n).
    Declare k of the integer datatype.
    for(k = n/2; k >= 1; k--)
      Call function max_heap(a,k,n)
End.
```

3. **Sample Code :**

```
#include <iostream>
using namespace std;
void max_heap(int *a, int m, int n) {
```

```
  int j, t;
  t = a[m];
  j = 2 * m;
  while (j <= n) {
    if (j < n && a[j+1] > a[j])
      j = j + 1;
    if (t > a[j])
      break;
    else if (t <= a[j]) {
      a[j / 2] = a[j];
      j = 2 * j;
    }
  }
  a[j/2] = t;
  return;
}
void build_maxheap(int *a,int n) {
  int k;
  for(k = n/2; k >= 1; k--) {
    max_heap(a,k,n);
  }
}
int main() {
  int n, i;
  cout<<"enter no of elements of array
";
  cin>>n;
  int a[30];
  for (i = 1; i <= n; i++) {
    cout<<"enter elements"<<" "<<(i)<<endl;
    cin>>a[i];
  }
  build_maxheap(a,n);
  cout<<"Max Heap
";
  for (i = 1; i <= n; i++) {
    cout<<a[i]<<endl;
  }
}
```

Output:

```
enter no of elements of array5
enter elements 1
10
enter elements 2
2
enter elements 3
99
enter elements 4
0
enter elements 5
57
Max Heap99
57
10
0
2
```

4. **Conclusion:**

We have perform in this practical to demonstrate understand and Implement heap sort

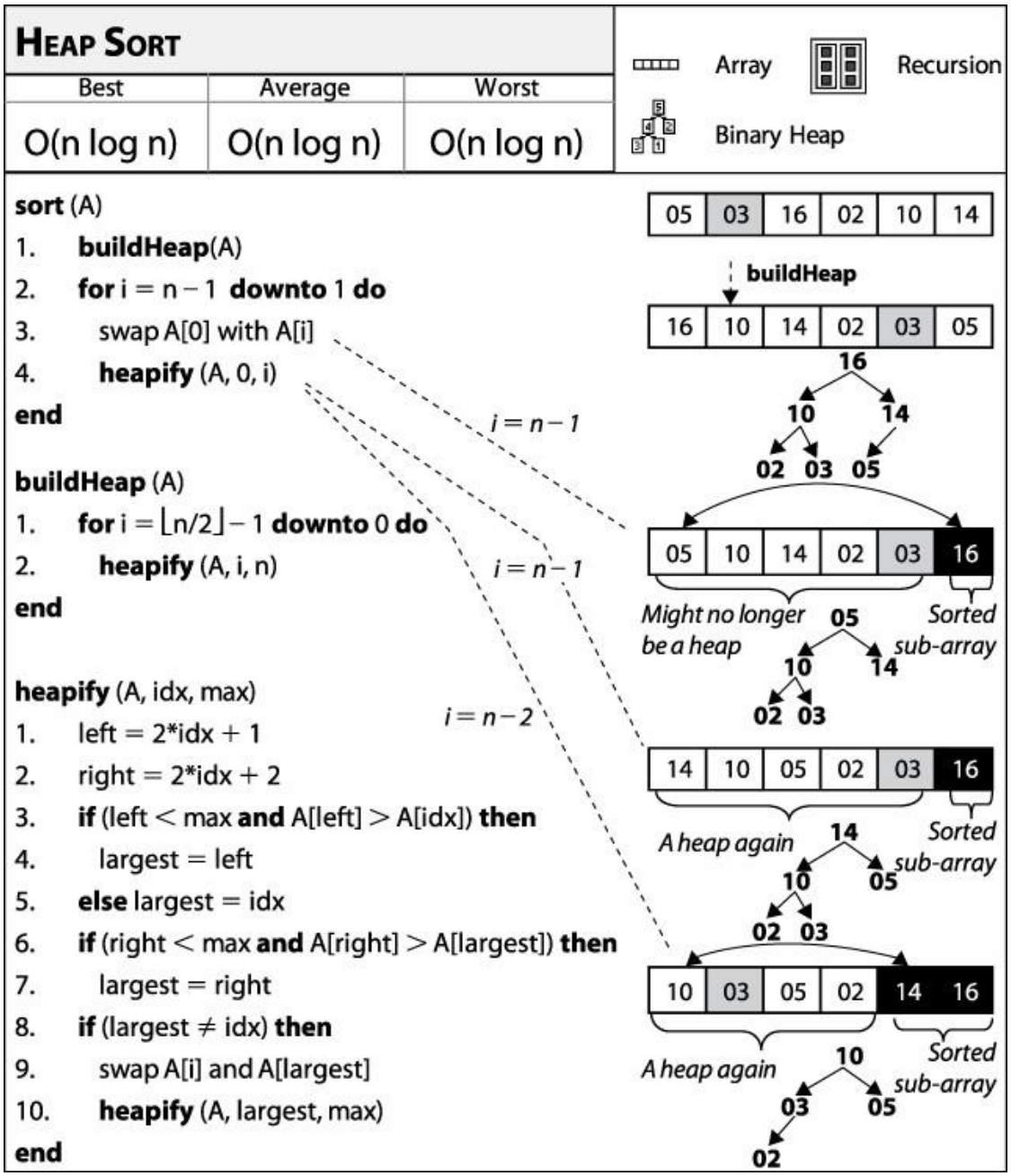
SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 02

DOP: **DOC:**
Roll No:

Title: Write a program for creating max./min. heap using ADJUST/HEAPIFY

1. **Objective :** To understand & implement heapify method.
2. **Algorithm :**



3. **Sample Code:**

```
// C++ program for implementation of Heap Sort
#include <iostream>
using namespace std;

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
	int largest = i; // Initialize largest as root
	int l = 2 * i + 1; // left = 2*i + 1
	int r = 2 * i + 2; // right = 2*i + 2

	// If left child is larger than root
	if (l < n && arr[l] > arr[largest])
		largest = l;

	// If right child is larger than largest so far
	if (r < n && arr[r] > arr[largest])
		largest = r;

	// If largest is not root
	if (largest != i) {
		swap(arr[i], arr[largest]);

		// Recursively heapify the affected sub-tree
		heapify(arr, n, largest);
	}
}

// main function to do heap sort
void heapSort(int arr[], int n)
{
	// Build heap (rearrange array)
	for (int i = n / 2 - 1; i >= 0; i--)
		heapify(arr, n, i);

	// One by one extract an element from heap
	for (int i = n - 1; i >= 0; i--) {
		// Move current root to end
		swap(arr[0], arr[i]);
```

```
            // call max heapify on the reduced heap
            heapify(arr, i, 0);
        }
    }

    /* A utility function to print array of size n */
    void printArray(int arr[], int n)
    {
        for (int i = 0; i < n; ++i)
            cout << arr[i] << " ";
        cout << "\n";
    }

    // Driver program
    int main()
    {
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        int n = sizeof(arr) / sizeof(arr[0]);

        heapSort(arr, n);

        cout << "Sorted array is \n";
        printArray(arr, n);
    }
```

Output:

```
Sorted array is
5 6 7 11 12 13
```

4. **Conclusion:**

We have perform in this practical to demonstrate To understand & implement heapify method.

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 03

DOP: **DOC:**
Roll No:

Title: Write a program to implement union and find operation

1. **Objective :** To understand and implement union and find operation.
2. **Algorithm :**

```
// A union-find algorithm to detect cycle in a graph
#include <bits/stdc++.h>
using namespace std;

// a structure to represent an edge in graph
class Edge {
public:
    int src, dest;
};

// a structure to represent a graph
class Graph {
public:
    // V-> Number of vertices, E-> Number of edges
    int V, E;

    // graph is represented as an array of edges
    Edge* edge;
};

// Creates a graph with V vertices and E edges
Graph* createGraph(int V, int E)
{
    Graph* graph = new Graph();
    graph->V = V;
    graph->E = E;

    graph->edge = new Edge[graph->E * sizeof(Edge)];

    return graph;
}

// A utility function to find the subset of an element i
int find(int parent[], int i)
{
```

```
    if (parent[i] == i)
        return i;
    return find(parent, parent[i]);
}

// A utility function to do union of two subsets
void Union(int parent[], int x, int y) { parent[x] = y; }

// The main function to check whether a given graph contains
// cycle or not
int isCycle(Graph* graph)
{
    // Allocate memory for creating V subsets
    int* parent = new int[graph->V * sizeof(int)];

    // Initialize all subsets as single element sets
    for(int i = 0; i < sizeof(int) * graph->V; i++) {
        parent[i] = i;
    }

    // Iterate through all edges of graph, find subset of
    // both vertices of every edge, if both subsets are
    // same, then there is cycle in graph.
    for (int i = 0; i < graph->E; ++i) {
        int x = find(parent, graph->edge[i].src);
        int y = find(parent, graph->edge[i].dest);

        if (x == y)
            return 1;

        Union(parent, x, y);
    }
    return 0;
};

// Driver code
int main()
{
    /* Let us create the following graph
        0
        | \
        |  \
        1---2 */
    int V = 3, E = 3;
    Graph* graph = createGraph(V, E);
```

```
    // add edge 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;

    // add edge 1-2
    graph->edge[1].src = 1;
    graph->edge[1].dest = 2;

    // add edge 0-2
    graph->edge[2].src = 0;
    graph->edge[2].dest = 2;

    if (isCycle(graph))
        cout << "Graph contains cycle";
    else
        cout << "Graph doesn't contain cycle";

    return 0;
}
```

Output:

Graph contains cycle

3. **Conclusion:**

We have perform in this practical to demonstrate To understand and implement union and find operation.

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 04

DOP: **DOC:**
Roll No:

Title: Write a program to find minimum and maximum form a given array.

1. **Objective :** To implement to find minimum and maximum form a given array
2. **Algorithm/ Logic :**
   1. The user is initially asked to enter the size of the array and it is stored in the variable 'n'.
   2. An array 'arr' of data type integer is declared with size 10.
   3. Elements of the array are asked to enter and stored in 'arr' using a for loop.
   4. The value at index 0 of arr is assigned to the variable 'max'.
   5. Using a for loop and initializing 'i' as 0, the largest element is found.
   6. If max is less than arr[i], then value of arr[i] is assigned to max. i is incremented in every iteration.
   7. The loop continues till 'i' is less than 'n'.
   8. Similarly, the smallest element is found.
   9. The value at index 0 of arr is assigned to the variable 'min'.
   10. Using a for loop the smallest element is assigned to min.
   11. The result is then printed.
3. **Sample Code:**

```
#include<iostream>
using namespace std;
int main ()
{
   int arr[10], n, i, max, min;
   cout << "Enter the size of the array : ";
   cin >> n;
   cout << "Enter the elements of the array : ";
   for (i = 0; i < n; i++)
      cin >> arr[i];
   max = arr[0];
   for (i = 0; i < n; i++)
   {
      if (max < arr[i])
         max = arr[i];
```

```
    }
    min = arr[0];
    for (i = 0; i < n; i++)
    {
        if (min > arr[i])
            min = arr[i];
    }
    cout << "Largest element : " << max;
    cout << "Smallest element : " << min;
    return 0;
}
```

Output :

```
Enter the size of the array : 5
Enter the elements of the array : 11 22 33 44 55
Largest element : 55
Smallest element : 11
```

4. **Conclusion:**

We have perform in this practical to demonstrate to implement to find minimum and maximum form a given array

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 05

DOP: **DOC:**
Roll No:

Title: Write a program for searching element form given array using binary search for n=1000,2000,3000 find exact time of execution.

1. **Objective :** To understand and implement binary search and analysis of it.
2. **Algorithm :**

Iteration Method :

```
binarySearch(arr, x, low, high)
    repeat till low = high
        mid = (low + high)/2
            if (x == arr[mid])
            return mid

            else if (x > arr[mid]) // x is on the right side
                low = mid + 1

            else                // x is on the left side
                high = mid – 1
```

Recursive Method :

```
binarySearch(arr, x, low, high)
      if low > high
        return False

      else
        mid = (low + high) / 2
            if x == arr[mid]
            return mid

        else if x > arr[mid]        // x is on the right side
            return binarySearch(arr, x, mid + 1, high)

        else                        // x is on the left side
            return binarySearch(arr, x, low, mid - 1)
```

3. **Sample Code:**

```
// C++ program to implement recursive Binary Search
#include <bits/stdc++.h>
```

```
using namespace std;

// A recursive binary search function. It returns
// location of x in given array arr[l..r] is present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
	if (r >= l) {
		int mid = l + (r - l) / 2;

		// If the element is present at the middle
		// itself
		if (arr[mid] == x)
			return mid;

		// If element is smaller than mid, then
		// it can only be present in left subarray
		if (arr[mid] > x)
			return binarySearch(arr, l, mid - 1, x);

		// Else the element can only be present
		// in right subarray
		return binarySearch(arr, mid + 1, r, x);
	}

	// We reach here when element is not
	// present in array
	return -1;
}

int main(void)
{
	int arr[] = { 2, 3, 4, 10, 40 };
	int x = 10;
	int n = sizeof(arr) / sizeof(arr[0]);
	int result = binarySearch(arr, 0, n - 1, x);
	(result == -1)
		? cout << "Element is not present in array"
		: cout << "Element is present at index " << result;
	return 0;
}
```

Output:

Element is present at index 3

```
// C++ program to implement iterative Binary Search#include <bits/stdc++.h>
    using namespace std;

    // A iterative binary search function. It returns
    // location of x in given array arr[l..r] if present,
    // otherwise -1
    int binarySearch(int arr[], int l, int r, int x)
    {
        while (l <= r) {
            int m = l + (r - l) / 2;

            // Check if x is present at mid
            if (arr[m] == x)
                return m;

            // If x greater, ignore left half
            if (arr[m] < x)
                l = m + 1;

            // If x is smaller, ignore right half
            else
                r = m - 1;
        }

        // if we reach here, then element was
        // not present
        return -1;
    }

    int main(void)
    {
        int arr[] = { 2, 3, 4, 10, 40 };
        int x = 10;
        int n = sizeof(arr) / sizeof(arr[0]);
        int result = binarySearch(arr, 0, n - 1, x);
        (result == -1)
            ? cout << "Element is not present in array"
            : cout << "Element is present at index " << result;
        return 0;
```

}

Analysis of Algorithms:

Linear search runs in ***O(n)*** time. Whereas binary search produces the result in ***O(log n)*** time

Let **T(n)** be the number of comparisons in worst-case in an array of **n** elements.

Hence,

$$T(n)= \{0 \quad T(n2)+1 \quad \text{if } n=1$$

$$\text{Otherwise } T(n)= \{0 \quad \text{if } n=1 \quad T(n2)+1 \quad \text{otherwise}$$

Using this recurrence relation T(n)= logn T (n)=logn.
Therefore, binary search uses O(logn) O(logn) time.

Output:

Element is present at index 3

4. Conclusion:

We have perform in this practical to demonstrate to understand and implement binary search and analysis of it.

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 06

DOP: **DOC:**
Roll No:

Title: Write a program for sorting given array in ascending/descending order with n=1000,2000,3000

1. **Objective :** To understand and implement sorting by using bubble sort.
2. **Algorithm :**

Algorithm 1: Bubble sort

```
Data: Input array A[]
Result: Sorted A[]
int i, j, k;
N = length(A);
for j = 1 to N do
    for i = 0 to N-1 do
        if A[i] > A[i+1] then
            temp = A[i];
            A[i] = A[i+1];
            A[i+1] = temp;
        end
    end
end
```

3. **Sample Code:**

```
// C++ program for implementation
// of Bubble sort
#include <bits/stdc++.h>
using namespace std;

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already
        // in place
```

```
            for (j = 0; j < n - i - 1; j++)
                if (arr[j] > arr[j + 1])
                    swap(arr[j], arr[j + 1]);
}

// Function to print an array
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int arr[] = { 5, 1, 4, 2, 8};
    int N = sizeof(arr) / sizeof(arr[0]);
    bubbleSort(arr, N);
    cout << "Sorted array: \n";
    printArray(arr, N);
    return 0;
}
// This code is contributed by rathbhupendra }
```

Output:

```
Sorted array:
1 2 4 5 8
```

4. **Conclusion:**

We have perform this practical to demonstrate To understand and implement sorting by using bubble sort.

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 07

DOP: **DOC:**
Roll No:

1. **Title:** find exact time of execution using
   - Heap sort
   - Merge sort
   - Quick sort

1. **Objective :** To analyze Heap Sort, Merge Sort and Quick Sort.
2. **Analysis of Heap Sort:**

C++ program for implementation of Heap Sort

```
#include <iostream>
using namespace std;

// To heapify a subtree rooted with node i
// which is an index in arr[].
// n is size of heap
void heapify(int arr[], int N, int i)
{

    // Initialize largest as root
    int largest = i;

    // left = 2*i + 1
    int l = 2 * i + 1;

    // right = 2*i + 2
    int r = 2 * i + 2;

    // If left child is larger than root
    if (l < N && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest
    // so far
    if (r < N && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i) {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected
        // sub-tree
        heapify(arr, N, largest);
```

```
    }
}

// Main function to do heap sort
void heapSort(int arr[], int N)
{

    // Build heap (rearrange array)
    for (int i = N / 2 - 1; i >= 0; i--)
        heapify(arr, N, i);

    // One by one extract an element
    // from heap
    for (int i = N - 1; i > 0; i--) {

        // Move current root to end
        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

// A utility function to print array of size n
void printArray(int arr[], int N)
{
    for (int i = 0; i < N; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

// Driver's code
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int N = sizeof(arr) / sizeof(arr[0]);

    // Function call
    heapSort(arr, N);

    cout << "Sorted array is \n";
    printArray(arr, N);
}
```

Output :

Output
Sorted array is
5 6 7 11 12 13

3. Analysis of Merge Sort:

```
C++ program for Merge Sort
#include <iostream>
using namespace std;

// Merges two subarrays of array[].
// First subarray is arr[begin..mid]
// Second subarray is arr[mid+1..end]
void merge(int array[], int const left, int const mid,
           int const right)
{
    auto const subArrayOne = mid - left + 1;
    auto const subArrayTwo = right - mid;

    // Create temp arrays
    auto *leftArray = new int[subArrayOne],
         *rightArray = new int[subArrayTwo];

    // Copy data to temp arrays leftArray[] and rightArray[]
    for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
    for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

    auto indexOfSubArrayOne
        = 0, // Initial index of first sub-array
        indexOfSubArrayTwo
        = 0; // Initial index of second sub-array
    int indexOfMergedArray
        = left; // Initial index of merged array

    // Merge the temp arrays back into array[left..right]
    while (indexOfSubArrayOne < subArrayOne
           && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne]
            <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray]
                = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        }
        else {
            array[indexOfMergedArray]
                = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }
    // Copy the remaining elements of
    // left[], if there are any
    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray]
```

```
            = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }
    // Copy the remaining elements of
    // right[], if there are any
    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray]
            = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
    delete[] leftArray;
    delete[] rightArray;
}

// begin is for left index and end is
// right index of the sub-array
// of arr to be sorted */
void mergeSort(int array[], int const begin, int const end)
{
    if (begin >= end)
        return; // Returns recursively

    auto mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);
}

// UTILITY FUNCTIONS
// Function to print an array
void printArray(int A[], int size)
{
    for (auto i = 0; i < size; i++)
        cout << A[i] << " ";
}

// Driver code
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    auto arr_size = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array is \n";
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    cout << "\nSorted array is \n";
    printArray(arr, arr_size);
    return 0;
```

```
}
```

Output :

```
Given array is
12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13
```

Analysis of Quick Sort:

```
C++ implementation of QuickSort */
#include <bits/stdc++.h>
using namespace std;

// A utility function to swap two elements
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition(int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i
        = (low
          - 1); // Index of smaller element and indicates
                // the right position of pivot found so far

    for (int j = low; j <= high - 1; j++) {
        // If current element is smaller than the pivot
        if (arr[j] < pivot) {
            i++; // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
```

```
{
   if (low < high) {
      /* pi is partitioning index, arr[p] is now
      at right place */
      int pi = partition(arr, low, high);

      // Separately sort elements before
      // partition and after partition
      quickSort(arr, low, pi - 1);
      quickSort(arr, pi + 1, high);
   }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
   int i;
   for (i = 0; i < size; i++)
      cout << arr[i] << " ";
   cout << endl;
}

// Driver Code
int main()
{
   int arr[] = { 10, 7, 8, 9, 1, 5 };
   int n = sizeof(arr) / sizeof(arr[0]);
   quickSort(arr, 0, n - 1);
   cout << "Sorted array: \n";
   printArray(arr, n);
   return 0;
}
```

Output :

```
Sorted array:
1 5 7 8 9 10
```

4. Conclusion:

We have perform in this practical to demonstrate To analyze Heap Sort, Merge Sort and Quick Sort.

Time complexity:

Algorithm	**Best Case**	**Worst Case**	**Average Case**
Heap Sort	O(n)	O(n*log(n))	O(n*log(n))
Merge Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))
Quick Sort	O(n*log(n))	$O(n^2)$	O(n*log(n))

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 08

DOP: **DOC:**
Roll No:

Title: Write a program for matrix multiplication using Strassen's matrix multiplication.

Objective: To understand and implement matrix multiplication using Strassen's matrix multiplication.

1.
2. **Algorithm :**

Algorithm 3 Strassen's Matrix Multiplication Algorithm

Input: $A=\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ and $B=\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \in \mathbb{R}^{n\times n}$

1: **if** $n=1$ **then**
2: $C=A\cdot B$
3: **else**
4: $M_1=(A_{11}+A_{22})\cdot(B_{11}+B_{22})$
5: $M_2=(A_{21}+A_{22})\cdot B_{11}$
6: $M_3=A_{11}\cdot(B_{12}-B_{22})$
7: $M_4=A_{22}\cdot(B_{21}-B_{11})$
8: $M_5=(A_{11}+A_{12})\cdot B_{22}$
9: $M_6=(A_{21}-A_{11})\cdot(B_{11}+B_{12})$
10: $M_7=(A_{12}-A_{22})\cdot(B_{21}+B_{22})$
11: $C_{11}=M_1+M_4-M_5+M_7$
12: $C_{12}=M_3+M_5$
13: $C_{21}=M_2+M_4$
14: $C_{22}=M_1-M_2+M_3+M_6$

Output: $A\cdot B=C=\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \in \mathbb{R}^{n\times n}$

3. **Sample Code:**

```
#include <bits/stdc++.h>
using namespace std;

// print the matrix
void print(vector<vector<int> > matrix) {
for(int i = 0; i < matrix.size(); i++){ for(int j = 0; j < matrix[i].size(); j++){
        cout << matrix[i][j] << ' ';
    }
```

```
        cout << endl;
    }
}

void multiply(vector<vector<int>> &A, vector<vector<int>> &B, vector<vector<int>> &C) {
      int N = 4;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            C[i][j] = 0;
            for (int k = 0; k < N; k++) {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
}

int main() {

    // Input Matrix A
      vector<vector<int>> A = {{2, 2, 3, 1},{1, 4, 1, 2},{2, 3, 1, 1}, {1, 3, 1, 2}};

      // Input Matrix B
    vector<vector<int>> B = {{2, 1, 2, 1},{3, 1, 2, 1},{3, 2, 1, 1}, {1, 4, 3, 2}};

      vector<vector<int>> C(4, vector<int>(4));

      multiply(A, B, C);

    // Printing the result
      print(C);
      return 0;
}
```

Output :

```
20 14 14 9
19 15 17 10
17 11 14 8
16 14 15 9
```

4. Conclusion:

We have perform in this practical to demonstrate to understand and implement matrix multiplication using Strassen's matrixmultiplication.

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 09

DOP: **DOC:**
Roll No:

Title: Write a program to find solution of Knapsack instant

1. **Objective :** To understand and implement of Knapsack instant.
2. **Algorithm :**

The algorithm takes the following inputs

- The maximum weight **W**
- The number of items **n**
- The two sequences **v = <v_1, v_2, …, v_n>** and **w = <w_1, w_2, …, w_n>**

```
Dynamic-0-1-knapsack (v, w, n, W)
for w = 0 to W do
c[0, w] = 0
for i = 1 to n do
c[i, 0] = 0
for w = 1 to W do
if w_i ≤ w then
if v_i + c[i-1, w-w_i] then
c[i, w] = v_i + c[i-1, w-w_i]
else c[i, w] = c[i-1, w]
else
c[i, w] = c[i-1, w]
```

3. **Sample Code:**

```
// C++ implementation to print all
// the possible solutions of the
// 0/1 Knapsack problem

#include <bits/stdc++.h>

using namespace std;

// Utility function to find the
// maximum of the two elements
int max(int a, int b) {
    return (a > b) ? a : b;
}
```

```
// Function to find the all the
// possible solutions of the
// 0/1 knapSack problem
int knapSack(int W, vector<int> wt,
                        vector<int> val, int n)
{
    // Mapping weights with Profits
    map<int, int> umap;

    set<vector<pair<int, int>>> set_sol;
    // Making Pairs and inserting
    // into the map
    for (int i = 0; i < n; i++) {
        umap.insert({ wt[i], val[i] });
    }

    int result = INT_MIN;
    int remaining_weight;
    int sum = 0;

    // Loop to iterate over all the
    // possible permutations of array
    do {
        sum = 0;

        // Initially bag will be empty
        remaining_weight = W;
        vector<pair<int, int>> possible;

        // Loop to fill up the bag
        // until there is no weight
        // such which is less than
        // remaining weight of the
        // 0-1 knapSack
        for (int i = 0; i < n; i++) {
            if (wt[i] <= remaining_weight) {

                remaining_weight -= wt[i];
                auto itr = umap.find(wt[i]);
                sum += (itr->second);
                possible.push_back({itr->first,
                    itr->second
```

```
                        });
                }
            }
            sort(possible.begin(), possible.end());
            if (sum > result) {
                result = sum;
            }
            if (set_sol.find(possible) ==
                                        set_sol.end()){
                for (auto sol: possible){
                    cout << sol.first << ": "
                         << sol.second << ", ";
                }
                cout << endl;
                set_sol.insert(possible);
            }

    } while (
            next_permutation(wt.begin(),
                                        wt.end()));
    return result;
}

// Driver Code
int main()
{
    vector<int> val{ 60, 100, 120 };
    vector<int> wt{ 10, 20, 30 };
    int W = 50;
    int n = val.size();
    int maximum = knapSack(W, wt, val, n);
    cout << "Maximum Profit = ";
    cout << maximum;
    return 0;
}
```

Output :

```
10: 60, 20: 100,
10: 60, 30: 120,
20: 100, 30: 120,
Maximum Profit = 220
```

4. Conclusion:

We have perform in this practical to demonstrate To understand and implement of Knapsack instant.

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 10

DOP: **DOC:**

Roll No:

Title: Write a program to find shortest path using single source shortest path

1. **Objective** : To understand and implement to find shortest path using single source shortest path.
2. **Algorithm :**

```
Algorithm DIJAKSTRA_SHORTEST_PATH(G, s, t)
// s is the source vertex
// t is the target vertex
// π[u] stores the parent / previous node of u
// V is the set of vertices in graph G

dist[s] ← 0
π[s] ← NIL

for each vertex v ∈ V do
   if v ≠ s then
      dist[v] ← ∞
      π[v] ← undefined
   end
   ENQUEUE(v, Q)                    // insert v to queue Q
end

while Q is not empty do
   u ← vertex in Q having minimum dist[u]
   if u == t then
      break
   end
   DEQUEUE(u, Q)            // Remove u from queue Q

   for each adjacent node v of u do
      val ← dist[u] + weight(u, v)
      if val<dist[v] then
         dist[v] ← val
         π[v] ← u
      end
   end
```

end

3. **Sample Code:**

```
// A C++ program for Dijkstra's single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

#include <limits.h>
#include <stdio.h>

// Number of vertices in the graph
#define V 9

// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{
	// Initialize min value
	int min = INT_MAX, min_index;

	for (int v = 0; v < V; v++)
		if (sptSet[v] == false && dist[v] <= min)
			min = dist[v], min_index = v;

	return min_index;
}

// A utility function to print the constructed distance array
int printSolution(int dist[], int n)
{
	printf("Vertex Distance from Source\n");
	for (int i = 0; i < V; i++)
		printf("%d \t\t %d\n", i, dist[i]);
}

// Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
	int dist[V]; // The output array. dist[i] will hold the shortest
	// distance from src to i

	bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
```

```
    // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist, V);
}

// driver program to test above function
int main()
{
    /* Let us create the example graph discussed above */
                int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                                    { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                                    { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                                    { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
```

```
                { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    dijkstra(graph, 0);

    return 0;
}
```

Output :

```
5       11
6       9
7       8
8       14
```

4. Conclusion:

We have perform in this practical to demonstrate To understand and implement to find shortest path using single source shortest path.

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 11

DOP: **DOC:**
Roll No:

Title: Write a program to find Minimum-Cost Spanning Trees.

1. **Objective :** Understand and implement to find Minimum-Cost Spanning Trees
2. **Algorithm :**

```
Procedure prims
        G – input graph
        U – random vertex
        V – vertices in graph G
begin
        T = Ø;
        U = { 1 };
        while (U ≠ V)
          let (u, v) be the least cost edge such that u ∈ U and v ∈ V - U;
          T = T U {(u, v)}
          U = U U {v}
end procedure
```

3. **Sample Code:**

```
// A C++ program for Prim's Minimum
// Spanning Tree (MST) algorithm. The program is
// for adjacency matrix representation of the graph
#include <bits/stdc++.h>
using namespace std;

// Number of vertices in the graph
#define V 5

// A utility function to find the vertex with
// minimum key value, from the set of vertices
// not yet included in MST
int minKey(int key[], bool mstSet[])
{
        // Initialize min value
```

```
	int min = INT_MAX, min_index;

	for (int v = 0; v < V; v++)
		if (mstSet[v] == false && key[v] < min)
			min = key[v], min_index = v;

	return min_index;
}

// A utility function to print the
// constructed MST stored in parent[]
void printMST(int parent[], int graph[V][V])
{
	cout << "Edge \tWeight\n";
	for (int i = 1; i < V; i++)
		cout << parent[i] << " - " << i << " \t"
			<< graph[i][parent[i]] << " \n";
}

// Function to construct and print MST for
// a graph represented using adjacency
// matrix representation
void primMST(int graph[V][V])
{
	// Array to store constructed MST
	int parent[V];

	// Key values used to pick minimum weight edge in cut
	int key[V];

	// To represent set of vertices included in MST
	bool mstSet[V];

	// Initialize all keys as INFINITE
	for (int i = 0; i < V; i++)
		key[i] = INT_MAX, mstSet[i] = false;

	// Always include first 1st vertex in MST.
	// Make key 0 so that this vertex is picked as first
	// vertex.
	key[0] = 0;
```

```
        parent[0] = -1; // First node is always root of MST

        // The MST will have V vertices
        for (int count = 0; count < V - 1; count++) {
                // Pick the minimum key vertex from the
                // set of vertices not yet included in MST
                int u = minKey(key, mstSet);

                // Add the picked vertex to the MST Set
                mstSet[u] = true;

                // Update key value and parent index of
                // the adjacent vertices of the picked vertex.
                // Consider only those vertices which are not
                // yet included in MST
                for (int v = 0; v < V; v++)

                        // graph[u][v] is non zero only for adjacent
                        // vertices of m mstSet[v] is false for vertices
                        // not yet included in MST Update the key only
                        // if graph[u][v] is smaller than key[v]
                        if (graph[u][v] && mstSet[v] == false
                                && graph[u][v]  < key[v])
                                parent[v] = u, key[v] = graph[u][v];
        }

        // print the constructed MST
        printMST(parent, graph);
}

// Driver's code
int main()
{
        /* Let us create the following graph
                2 3
        (0)--(1)--(2)
        | / \ |
        6| 8/ \5 |7
        | / \ |
        (3)-------(4)
                        9          */
```

```
        int graph[V][V] = { { 0, 2, 0, 6, 0 },
                                                { 2, 0, 3, 8, 5 },
                                                { 0, 3, 0, 0, 7 },
                                                { 6, 8, 0, 0, 9 },
                                                { 0, 5, 7, 9, 0 } };

        // Print the solution
        primMST(graph);

        return 0;
    }
```

Output :

```
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

4. Conclusion:

We have perform in this practical to demonstrate Understand and implement to find Minimum-Cost Spanning Trees

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 12

DOP: **DOC:**
Roll No:

Title: Write a program to find shortest path using all pair path

1. **Objective:** To understand and implement to find shortest path using all pair path.
2. **Algorithm :**

FloydWarshal(cost)
Input − The cost matrix of given Graph.

Output − Matrix to for shortest path between any vertex to any vertex.

```
Begin
  for k := 0 to n, do
    for i := 0 to n, do
      for j := 0 to n, do
        if cost[i,k] + cost[k,j] < cost[i,j], then
          cost[i,j] := cost[i,k] + cost[k,j]
        done
      done
    done
    display the current cost matrix
End
```

3. **Sample Code:**

```
#include<iostream>
#include<iomanip>
#define NODE 7
#define INF 999
using namespace std;
//Cost matrix of the graph
int costMat[NODE][NODE] = {
  {0, 3, 6, INF, INF, INF, INF},
  {3, 0, 2, 1, INF, INF, INF},
  {6, 2, 0, 1, 4, 2, INF},
  {INF, 1, 1, 0, 2, INF, 4},
  {INF, INF, 4, 2, 0, 2, 1},
  {INF, INF, 2, INF, 2, 0, 1},
```

```
  {INF, INF, INF, 4, 1, 1, 0}
};
void floydWarshal(){
  int cost[NODE][NODE]; //defind to store shortest distance from any node to any node
  for(int i = 0; i<NODE; i++)
    for(int j = 0; j<NODE; j++)
      cost[i][j] = costMat[i][j]; //copy costMatrix to new matrix
      for(int k = 0; k<NODE; k++){
        for(int i = 0; i<NODE; i++)
          for(int j = 0; j<NODE; j++)
            if(cost[i][k]+cost[k][j] < cost[i][j])
              cost[i][j] = cost[i][k]+cost[k][j];
  }
  cout << "The matrix:" << endl;
  for(int i = 0; i<NODE; i++){
    for(int j = 0; j<NODE; j++)
      cout << setw(3) << cost[i][j];
    cout << endl;
  }
}
int main(){
  floydWarshal();
}
```

Output :

```
The matrix:
  0  3  5  4  6  7  7
  3  0  2  1  3  4  4
  5  2  0  1  3  2  3
  4  1  1  0  2  3  3
  6  3  3  2  0  2  1
  7  4  2  3  2  0  1
  7  4  3  3  1  1  0
```

4. Conclusion:

We have perform in this practical to demonstrate To understand and implement to find shortest path using all pair path.

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 13

DOP: **DOC:**
Roll No:

Title: Write a program to find longest common subsequence

1. **Objective :** To understand and implement find longest common subsequence
2. **Algorithm :**

X and Y be two given sequences
Initialize a table LCS of dimension X.length * Y.length
X.label = X
Y.label = Y
LCS[0][] = 0
LCS[][0] = 0
Start from LCS[1][1]
Compare X[i] and Y[j]
If X[i] = Y[j]
LCS[i][j] = 1 + LCS[i-1, j-1]
Point an arrow to LCS[i][j]
Else
LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])
Point an arrow to max(LCS[i-1][j], LCS[i][j-1])

3. **Sample Code:**

```
// The longest common subsequence in C++
#include <iostream>
#include <cstring>
using namespace std;
void lcsAlgo(char *S1, char *S2, int m, int n) {
 int LCS_table[m + 1][n + 1];
 // Building the mtrix in bottom-up way
 for (int i = 0; i <= m; i++) {
 for (int j = 0; j <= n; j++) {
 if (i == 0 || j == 0)
 LCS_table[i][j] = 0;
 else if (S1[i - 1] == S2[j - 1])
 LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
 else
 LCS_table[i][j] = max(LCS_table[i - 1][j], LCS_table[i][j - 1]);
 }
 }
```

```
 int index = LCS_table[m][n];
 char lcsAlgo[index + 1];
 lcsAlgo[index] = '\0';
 int i = m, j = n;
 while (i > 0 && j > 0) {
 if (S1[i - 1] == S2[j - 1]) {
 lcsAlgo[index - 1] = S1[i - 1];
 i--;
 j--;
 index--;
 }
 else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
 i--;
 else
 j--;
 }

 // Printing the sub sequences
 cout << "S1 : " << S1 << "\nS2 : " << S2 << "\nLCS: " << lcsAlgo << "\n";
}
int main() {
 char S1[] = "ACADB";
 char S2[] = "CBDA";
 int m = strlen(S1);
 int n = strlen(S2);
 lcsAlgo(S1, S2, m, n);
}
```

Output :

S1 : ACADB
S2 : CBDA
LCS: CB

4. **Conclusion:**

We have perform in this practical to demonstrate To understand and implement find longest common subsequence

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

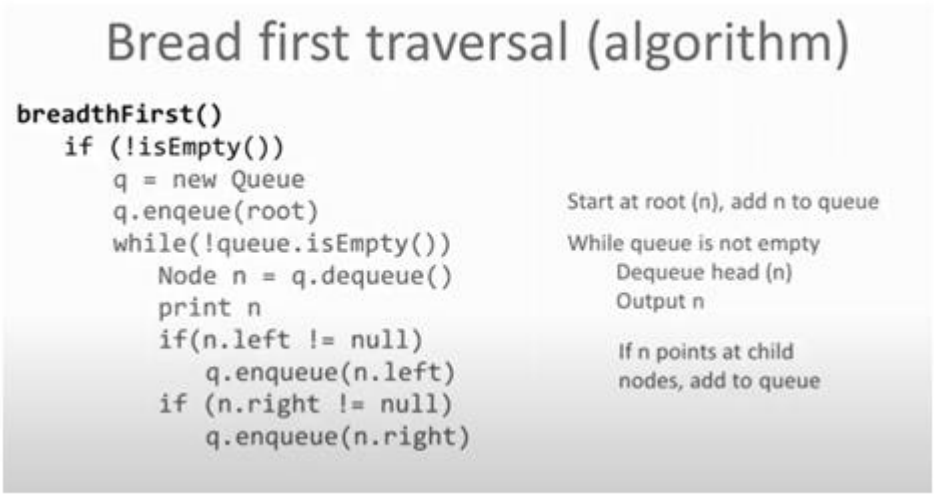
Practical: 14

DOP: **DOC:**

Roll No:

Title: Write a program to implement breadth first traversal

1. **Objective :** To implement breadth first traversal
2. **Algorithm :**



3. **Sample Code:**

```
// Program to print BFS traversal from a given
// source vertex. BFS(int s) traverses vertices
// reachable from s.
#include<bits/stdc++.h>
using namespace std;

// This class represents a directed graph using
// adjacency list representation
class Graph
{
    int V;    // No. of vertices

    // Pointer to an array containing adjacency
    // lists
    vector<list<int>> adj;
public:
    Graph(int V);  // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w);

    // prints BFS traversal from a given source s
    void BFS(int s);
```

```
};

Graph::Graph(int V)
{
   this->V = V;
   adj.resize(V);
}

void Graph::addEdge(int v, int w)
{
   adj[v].push_back(w); // Add w to v's list.
}

void Graph::BFS(int s)
{
   // Mark all the vertices as not visited
   vector<bool> visited;
   visited.resize(V,false);

   // Create a queue for BFS
   list<int> queue;

   // Mark the current node as visited and enqueue it
   visited[s] = true;
   queue.push_back(s);

   while(!queue.empty())
   {
      // Dequeue a vertex from queue and print it
      s = queue.front();
      cout << s << " ";
      queue.pop_front();

      // Get all adjacent vertices of the dequeued
      // vertex s. If a adjacent has not been visited,
      // then mark it visited and enqueue it
      for (auto adjecent: adj[s])
      {
         if (!visited[adjecent])
         {
            visited[adjecent] = true;
            queue.push_back(adjecent);
         }
      }
   }
}

// Driver program to test methods of graph class
int main()
{
   // Create a graph given in the above diagram
   Graph g(4);
   g.addEdge(0, 1);
   g.addEdge(0, 2);
```

```
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "
         << "(starting from vertex 2) \n";
    g.BFS(2);

    return 0;
}
```

Output:

```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
```

4. Conclusion:

We have perform in this practical to demonstrate To implement breadth first traversal

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 15

DOP: **DOC:**
Roll No:

Title: Write a program to implement depth first traversal.

1. **Objective :** To implement depth first traversal
2. **Algorithm :**

Depth-First Traversal Algorithm

- In this method, After visiting a vertex v, which is adjacent to w1, w2, w3, ...; Next we visit one of v's adjacent vertices, w1 say. Next, we visit all vertices adjacent to w1 before coming back to w2, etc.
- Must keep track of vertices already visited to avoid cycles.
- The method can be implemented using recursion or iteration.
- The iterative preorder depth-first algorithm is:

```
1 push the starting vertex onto the stack
2 while(stack is not empty){
3     pop a vertex off the stack, call it v
4     if v is not already visited, visit it
5     push vertices adjacent to v, not visited, onto the stack
6 }
```

- Note: Adjacent vertices can be pushed in any order; but to obtain a unique traversal, we will push them in reverse alphabetical order.

3. **Sample Code:**

implementation of **DFS tree traversal algorithm**,

```
#include <iostream>
using namespace std;
struct Node {
  int data;
  struct Node* left, *right;
  Node(int data) {
    this->data = data;
    left = right = NULL;
  }
};
```

```
void printPostorder(struct Node* node) {
  if (node == NULL)
    return;
  printPostorder(node->left);
  printPostorder(node->right);
  cout << node->data << " ";
}
void printInorder(struct Node* node) {
  if (node == NULL)
    return;
  printInorder(node->left);
  cout << node->data << " ";
  printInorder(node->right);
}
void printPreorder(struct Node* node) {
  if (node == NULL)
    return;
  cout << node->data << " ";
  printPreorder(node->left);
  printPreorder(node->right);
}
int main() {
  struct Node *root = new Node(1);
  root->left = new Node(2);
  root->right = new Node(3);
  root->left->left = new Node(4);
  root->left->right = new Node(5);
  cout << "
Preorder traversal of binary tree is
";
  printPreorder(root);
  cout << "
Inorder traversal of binary tree is
";
  printInorder(root);
  cout << "
Postorder traversal of binary tree is
";
  printPostorder(root);
  return 0;
}
```

Output:

```
Preorder traversal of binary tree is 1 2 4 5 3
Inorder traversal of binary tree is 4 2 5 1 3
Postorder traversal of binary tree is 4 5 2 3 1
```

4. Conclusion:

We have perform in this practical to demonstrate To implement breadth first traversal

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Computer Applications

Practical: 16

DOP: **DOC:**
Roll No:

Title: Write a program to find all solutions for 8-queen problem using backtracking.

1. **Objective :** To Implement find all solutions for 8-queen problem using backtracking
2. **Algorithm :**

```
Place (k, i)
1 {
2    For j ← 1 to k - 1
3.       do if (x [j] = i)
4.        or (Abs x [j]) - i) = (Abs (j - k))
5.      then return false;
6.       return true;
7. }

1. N - Queens (k, n)
2. {
3.     For i ← 1 to n
4.     do if Place (k, i) then5.
       {
6.       x [k] ← i;
7.       if (k ==n) then
8.         write (x [1...n));
9.       else
10.      N - Queens (k + 1, n);
11.    }
12. }
```

8. **Sample Code:**

```
include<iostream>
using namespace std;
#define N 4
void printBoard(int board[N][N]) {
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++)
      cout << board[i][j] << " ";
      cout << endl;
  }
```

```
}
bool isValid(int board[N][N], int row, int col) {
   for (int i = 0; i < col; i++) //check whether there is queen in the left or not
      if (board[row][i])
         return false;
   for (int i=row, j=col; i>=0 && j>=0; i--, j--)
      if (board[i][j]) //check whether there is queen in the left upper diagonal or not
         return false;
   for (int i=row, j=col; j>=0 && i<N; i++, j--)
      if (board[i][j]) //check whether there is queen in the left lower diagonal or not
         return false;
   return true;
}
bool solveNQueen(int board[N][N], int col) {
   if (col >= N) //when N queens are placed successfully
      return true;
   for (int i = 0; i < N; i++) { //for each row, check placing of queen is possible or not
      if (isValid(board, i, col) ) {
         board[i][col] = 1; //if validate, place the queen at place (i, col)
         if ( solveNQueen(board, col + 1)) //Go for the other columns recursively
            return true;
         board[i][col] = 0; //When no place is vacant remove that queen
      }
   }
   return false; //when no possible order is found
}
bool checkSolution() {
   int board[N][N];
   for(int i = 0; i<N; i++)
   for(int j = 0; j<N; j++)
   board[i][j] = 0; //set all elements to 0
   if ( solveNQueen(board, 0) == false ) { //starting from 0th column
      cout << "Solution does not exist";
      return false;
   }
   printBoard(board);
   return true;
}
int main() {
   checkSolution();
}
```

Output :

```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

9. Conclusion:

We have perform in this practical to demonstrate To Implement find all solutions for 8-queen problem using backtracking