

# DAA LAB MANUAL

## EXERCISE-1:

Implement Recursive Binary Search Algorithm

### PROGRAM:

```
#include <bits/stdc++.h>
using namespace std;
int binarySearch(vector<int>arr, int l, int r, int x)
{
    while (l <= r)
    {
        int m = l + (r - l) / 2;
        if (arr[m] == x)
            return m;
        else if (arr[m] < x)
            l = m + 1;
        else
            r = m - 1;
    }
    return -1;
}

int main()
{
    int n;
    cout<<"Enter size of Array: "; cin>>n;
    cout<<endl<<"Enter elements: "; vector<int>v;
    for(int i=0;i<n;i++)
    {
        int k; cin>>k;
        v.push_back(k);
    }
    cout<<endl<<"Enter element to be searched:";
    int x;
```

```
    cin>>x;
    int ans = binarySearch(v, 0, n - 1, x);
    if (ans == -1)
        cout << "Element is not present in array"<<endl;
    else
        cout << "Element is present at index " << ans<<endl;
    return 0;
}
```

## OUTPUT:

```
Enter size of Array: 3
Enter elements: 1 2 3
Enter element to be searched:2
Element is present at index 1
```

## EXERCISE-2:

Implement Recursive Quicksort Algorithm

### PROGRAM:

```
#include <iostream>

using namespace std;

int partition(int arr[], int start, int end)
{
    int pivot = arr[start];
    int count = 0;
    for (int i = start + 1; i <= end; i++)
    {
        if (arr[i] <= pivot)
            count++;
    }
    int pivotIndex = start + count;
    swap(arr[pivotIndex], arr[start]);
    int i = start, j = end;
    while (i < pivotIndex && j > pivotIndex)
    {
        while (arr[i] <= pivot)
        {
            i++;
        }
        while (arr[j] > pivot)
        {
            j--;
        }
        if (i < pivotIndex && j > pivotIndex)
        {
            swap(arr[i++], arr[j--]);
        }
    }
    return pivotIndex;
}
```

```

void quickSort(int arr[], int start, int end)
{
    if (start >= end)
        return;

    int p = partition(arr, start, end);
    quickSort(arr, start, p - 1);
    quickSort(arr, p + 1, end);
}

void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout<<arr[i]<<" ";
}

int main()
{
    int n;
    cout<<"Enter number of elements: "<<endl;
    cin>>n;
    int arr[n];
    cout<<"Enter elements: "<<endl;
    for(int i=0;i<n;i++)
        cin>>arr[i];
    cout<<"Given array: "<<endl;
    for(int i=0;i<n;i++)
        cout<<arr[i]<<" ";
    cout<<endl;
    quickSort(arr, 0, n - 1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}

```

## OUTPUT:

```
Enter number of elements:
```

```
6
```

```
Enter elements:
```

```
2 5 10 2 5 1
```

```
Given array:
```

```
2 5 10 2 5 1
```

```
Sorted array:
```

```
1 2 2 5 5 10
```

### EXERCISE-3:

Implement Recursive Mergesort Algorithm

#### PROGRAM:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
void merge(int arr[], int l, int m, int r)
```

```
{
```

```
    int i, j, k;
```

```
    int n1 = m - l + 1;
```

```
    int n2 = r - m;
```

```
    int L[n1], R[n2];
```

```
    for (i = 0; i < n1; i++)
```

```
        L[i] = arr[l + i];
```

```
    for (j = 0; j < n2; j++)
```

```
        R[j] = arr[m + 1 + j];
```

```
    i = 0;
```

```
    j = 0;
```

```
    k = l;
```

```
    while (i < n1 && j < n2)
```

```
    {
```

```
        if (L[i] <= R[j])
```

```
        {
```

```
            arr[k] = L[i];
```

```
            i++;
```

```
        }
```

```
    else
```

```
    {
```

```
        arr[k] = R[j];
```

```
        j++;
```

```
    }
```

```
    k++;
```

```
}
```

```
while (i < n1)
```

```

    {
        arr[k] = L[i];

        i++;

        k++;
    }
    while (j < n2)
    {
        arr[k] = R[j];

        j++;

        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);

        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

void printArray(int A[], int size)
{
    int i;

    for (i = 0; i < size; i++)
        printf("%d ", A[i]);

    printf("\n");
}

int main()
{
    int n,i;

    printf("enter n value\n");

    scanf("%d",&n);

```

```
int a[n];  
printf("enter array to be sorted\n");  
for(i=0;i<n;i++)  
{  
    scanf("%d",&a[i]);  
}  
printf("Given array is \n");  
printArray(a,n);  
mergeSort(a, 0,n-1);  
printf("\nSorted array is \n");  
printArray(a,n);  
return 0;  
}
```

**OUTPUT:**

```
enter n value  
10  
enter array to be sorted  
1 4 3 2 5 6 8 97 1 2  
Given array is  
1 4 3 2 5 6 8 97 1 2  
Sorted array is  
1 1 2 2 3 4 5 6 8 97
```



#### EXERCISE-4:

Implement Randomized Quicksort Algorithm

#### PROGRAM:

```
#include <iostream>

using namespace std;

int partition(int arr[], int low, int high)
{
    int pivot = arr[low];
    int i = low - 1, j = high + 1;
    while (true)
    {
        do
        {
            i++;
        } while (arr[i] < pivot);
        do
        {
            j--;
        } while (arr[j] > pivot);
        if (i >= j)
            return j;
        swap(arr[i], arr[j]);
    }
}

int partition_r(int arr[], int low, int high)
{
    srand(time(NULL));
    int random = low + rand() % (high - low);
    swap(arr[random], arr[low]);
    return partition(arr, low, high);
}
```

```

void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition_r(arr, low, high);
        quickSort(arr, low, pi);
        quickSort(arr, pi + 1, high);
    }
}

void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    int n;
    cout<<"Enter number of elements: "<<endl;
    cin>>n;
    int arr[n];
    cout<<"Enter elements: "<<endl;
    for(int i=0;i<n;i++)
        cin>>arr[i];
    cout<<"Given array: "<<endl;
    for(int i=0;i<n;i++) cout<<arr[i]<<" ";
    cout<<endl;
    quickSort(arr, 0, n - 1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}

```

## OUTPUT:

```
Enter number of elements:
```

```
7
```

```
Enter elements:
```

```
7 8 9 6 5 2 3
```

```
Given array:
```

```
7 8 9 6 5 2 3
```

```
Sorted array:
```

```
2 3 5 6 7 8 9
```

### EXERCISE-5:

Find an optimal solution for a Knapsack Problem

#### PROGRAM:

```
#include <bits/stdc++.h>

using namespace std;

int main()
{
    int n,mx;

    cout<<"Enter number of elements: "<<endl;

    cin>>n;

    cout<<endl<<"Enter the maximum capacity:"<<endl;

    cin>>mx;

    vector<int>w,p;
    vector<double>d;

    cout<<"Enter weights: "<<endl;
    for(int i=0;i<n;i++)
    {
        int k;

        cin>>k;

        w.push_back(k);
    }

    cout<<endl<<"Enter profits: "<<endl;
    for(int i=0;i<n;i++)
    {
        int k;

        cin>>k;

        p.push_back(k);
    }

    for(int i=0;i<n;i++)
    {
        d.push_back(p[i]/(w[i]*1.0));
    }
```

```

for(int i=0;i<n;i++)
{
    for(int j=i+1;j<n;j++)
    {
        if(d[i]<d[j])
        {
            double tm=d[i]; d[i]=d[j];
            d[j]=tm;
            int t=w[i];
            w[i]=w[j];
            w[j]=t;
            t=p[i];
            p[i]=p[j];
            p[j]=t;
        }
    }

}

double profit=0;
for(int i=0;i<n;i++)
{
    if(mx<w[i])
    {
        profit+=mx*d[i];
        break;
    }
    else
    {
        profit+=p[i];
        mx-=w[i];
    }
}

```

```
cout<<endl<<"Maximum Profit is "<<profit<<endl;  
return 0;  
}
```

### OUTPUT:

```
Enter number of elements:  
3  
Enter the maximum capacity:  
20  
Enter weights:  
18 15 10  
Enter profits:  
25 24 15  
Maximum Profit is 31.5  
|
```

### EXERCISE-6:

Find the shortest path using Single Source Shortest Path Algorithm

#### PROGRAM:

```
#include <bits/stdc++.h>

using namespace std;

int miniDistance(int dist[], bool visited[],int V)
{
    int min = INT_MAX;
    int min_index;
    for (int v = 0; v < V; v++)
        if (visited[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;
    return min_index;
}

void print(int dist[],int V)
{
    printf("Vertex Distance from Source\n");
    for (int i = 0; i < V; i++)
        cout<<i<<" "<<dist[i]<<endl;
}

void dijkstra(vector<vector<int>>graph, int src,int V)
{
    int dist[V];
    bool visited[V];
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, visited[i] = false;
    dist[src] = 0;
    for (int i = 0; i < V - 1; i++)
    {
```

```

    int u = miniDistance(dist, visited,V);
    visited[u] = true;
    for (int v = 0; v < V; v++)
        if (!visited[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
    }
    print(dist, V);
}

```

```

int main()
{
    int V;
    cout<<"Enter number of vertices:"<<endl;
    cin>>V;
    cout<<endl<<"Enter adjacency matrix elements:"<<endl;
    vector<vector<int>>>graph;
    for(int i=0;i<V;i++)
    {
        vector<int>>v;
        for(int j=0;j<V;j++)
        {
            int k;
            cin>>k;
            v.push_back(k);
        }
        graph.push_back(v);
    }
    dijkstra(graph, 0,V);
    return 0;
}

```

**OUTPUT:**



Enter number of vertices:

5

Enter adjacency matrix elements:

0 10 5 0 0

0 0 0 1 0

0 3 0 9 2

0 0 0 0 0

2 0 0 6 0

Vertex Distance from Source

0 0

1 8

2 5

3 9

4 7

### EXERCISE-7:

Implement Huffman Coding Technique

#### PROGRAM:

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

struct MinHeapNode
{
    char data;
    int freq;
    MinHeapNode *left, *right;
    MinHeapNode(char data, int freq)
    {
        left = right = NULL;
        this->data = data;
        this->freq = freq;
    }
};

struct compare
{
    bool operator()(MinHeapNode* l, MinHeapNode* r)
    {
        return (l->freq > r->freq);
    }
};

void printCodes(struct MinHeapNode* root, string str)
{
    if (!root) return;
    if (root->data != '$')
        cout << root->data << ": " << str << "\n";
    printCodes(root->left, str + "0");
    printCodes(root->right, str + "1");
}
```

```

}

void HuffmanCodes(char data[], int freq[], int size)
{
    struct MinHeapNode *left, *right, *top;
    priority_queue<MinHeapNode*, vector<MinHeapNode*>, compare> minHeap;
    for (int i = 0; i < size; ++i)
        minHeap.push(new MinHeapNode(data[i], freq[i]));
    while (minHeap.size() != 1)
    {
        left = minHeap.top();
        minHeap.pop();
        right = minHeap.top();
        minHeap.pop();
        top = new MinHeapNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        minHeap.push(top);
    }
    printCodes(minHeap.top(), "");
}

int main()
{
    int n;
    cout<<"Enter number of characters:"<<endl;
    cin>>n;
    char arr[n];
    int freq[n];
    cout<<endl<<"Enter Characters:"<<endl;
    for(int i=0;i<n;i++)
        cin>>arr[i];
    cout<<endl<<"Enter Frequencies"<<endl;

```

```
for(int i=0;i<n;i++)  
    cin>>freq[i];  
HuffmanCodes(arr, freq, n);  
return 0;  
}
```

## OUTPUT:

```
Enter number of characters:  
5  
Enter Characters:  
s t u v z  
Enter Frequencies  
33 12 9 7 22  
s: 0  
z: 10  
t: 110  
v: 1110  
u: 1111
```

### EXERCISE-8:

Implement 0/1 Knapsack Problem

#### PROGRAM:

```
#include <stdio.h>

int max(int a, int b)
{
    return (a > b) ? a : b;
}

int knapsack(int weights[], int values[], int n, int capacity)
{
    int i, w;
    int dp[n + 1][capacity + 1];
    for (i = 0; i <= n; i++)
    {
        for (w = 0; w <= capacity; w++)
        {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (weights[i - 1] <= w)
            {
                int p = dp[i - 1][w - weights[i - 1]] + values[i - 1];
                dp[i][w] = max(dp[i - 1][w], p);
            }
            else
                dp[i][w] = dp[i - 1][w];
        }
    }
    printf("Selected items: ");
    i = n;
    w = capacity;
    while (i > 0 && w > 0)
    {
        if (dp[i][w] != dp[i - 1][w])
```

```

{
    printf("%d ", i - 1);
    w -= weights[i - 1];
}
i--;
}
printf("\n");
return dp[n][capacity];
}

```

```

int main()
{
    int n,i;
    printf("Enter Number of items:");
    scanf("%d",&n);
    int weights[n],profits[n];
    printf("\nEnter Weights:");
    for(i=0;i<n;i++)
    {
        scanf("%d",&weights[i]);
    }
    printf("\nEnter profits:");
    for(i=0;i<n;i++)
    {
        scanf("%d",&profits[i]);
    }
    int capacity;
    printf("\nEnter max capacity:");
    scanf("%d",&capacity);
    int maxValue = knapsack(weights,profits, n, capacity);
    printf("Maximum value: %d\n", maxValue);
    return 0;
}

```

## OUTPUT:

```
Enter Number of items:7
Enter Weights:2 3 5 7 1 4 1
Enter profits:10 5 15 7 6 18 3
Enter max capacity:15
Selected items: 5 4 2 1 0
Maximum value: 54
|
```

### EXERCISE-9:

Find the shortest path using All Pairs Shortest Path Algorithm

#### PROGRAM:

```
#include <bits/stdc++.h>

using namespace std;

void printSolution(vector<vector<int>>>dist,int V)
{
    cout << "The shortest distances between every pair of vertices \n";
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            if (dist[i][j] == INT_MAX)
                cout << "∞" << " ";
            else
                cout << dist[i][j] << " ";
        }
        cout << endl;
    }
}

void floydWarshall(vector<vector<int>>>dist,int V)
{
    int i, j, k;
    for (k = 0; k < V; k++)
    {
        for (i = 0; i < V; i++)
        {
            for (j = 0; j < V; j++)
            {
                if (dist[i][j] > (dist[i][k] + dist[k][j]) && (dist[k][j] != INT_MAX && dist[i][k] != INT_MAX))
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
}
```



```

    printSolution(dist,V);
}
int main()
{
    int V;
    cout<<"Enter number of vertices:";
    cin>>V;
    vector<vector<int>>>graph;
    cout<<endl<<"Enter adjacency matrix -1 if no edge exists:";
    for(int i=0;i<V;i++)
    {
        vector<int>v;
        for(int j=0;j<V;j++)
        {
            int k; cin>>k;
            if(k==-1)
                v.push_back(INT_MAX);
            else
                v.push_back(k);
        }
        graph.push_back(v);
    }
    floydWarshall(graph,V);
    return 0;
}

```

## OUTPUT:

```

Enter number of vertices:4
Enter adjacency matrix -1 if no edge exists:
0 2 3 -1
-1 0 -1 -1
-1 6 0 4
5 -1 -1 0
The shortest distances between every pair of vertices
0 2 3 7
∞ 0 ∞ ∞
9 6 0 4
5 7 8 0

```

### EXERCISE-10:

Implement Travelling Salesman Problem

#### PROGRAM:

```
#include <bits/stdc++.h>

using namespace std;

int travllingSalesmanProblem(vector<vector<int>>>graph, int s,int V)
{
    vector<int> vertex;
    for (int i = 0; i < V; i++)
        if (i != s)
            vertex.push_back(i);
    int min_path = INT_MAX;
    do
    {
        int current_pathweight = 0;
        int k = s;
        for (int i = 0; i < vertex.size(); i++)
        {
            current_pathweight += graph[k][vertex[i]];
            k = vertex[i];
        }
        current_pathweight += graph[k][s];
        min_path = min(min_path, current_pathweight);
    } while (next_permutation(vertex.begin(), vertex.end()));
    return min_path;
}

int main()
{
    vector<vector<int>>>graph;
    int V,s;
    cout<<"Enter number of vertices:";
    cin>>V;
    cout<<"Enter adjacency matrix of graph:";
    for(int i=0;i<V;i++)
```

```

{
    vector<int>v;
    for(int j=0;j<V;j++)
    {
        int k; cin>>k;
        v.push_back(k);
    }
    graph.push_back(v);
}

cout<<"Enter Source:";

cin>>s;

cout << travllingSalesmanProblem(graph, s,V) << endl;

return 0;

}

```

## OUTPUT:

```

Enter number of vertices:4
Enter adjacency matrix of graph:
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0
Enter Source:0
80

```

### EXERCISE-11:

Implement Sum of Subsets Problem

#### PROGRAM:

```
#include <bits/stdc++.h>

using namespace std;

int found = 0;

void PrintSubsetSum(int i, int n, vector<int>set, int targetSum,vector<int>& subset)
{
    if (targetSum == 0)
    {
        found = 1;
        cout << "[ ";
        for (int i = 0; i < subset.size(); i++)
        {
            cout << subset[i] << " ";
        }
        cout << "]" ;
        return;
    }
    if (i == n)
        return;
    PrintSubsetSum(i + 1, n, set, targetSum, subset);
    if (set[i] <= targetSum)
    {
        subset.push_back(set[i]);
        PrintSubsetSum(i + 1, n, set, targetSum - set[i],subset);
        subset.pop_back();
    }
}

int main()
{
    int n;
    cout<<"Enter number of elements:";
```

```

cin>>n;
cout<<"Enter elements:";
vector<int> subset,set;
for(int i=0;i<n;i++)
{
    int k; cin>>k;
    subset.push_back(k);
}
cout<<"Enter target sum:";
int targetSum;
cin>>targetSum;
PrintSubsetSum(0,n,subset,targetSum,set);
if(found==0)
    cout<<"No Subset";
return 0;
}

```

#### OUTPUT:

```

Enter number of elements:3
Enter elements:1 2 1
Enter target sum:3
[ 2 1 ][ 1 2 ]

```

## EXERCISE-12:

Implement N-Queens Problem

### PROGRAM:

```
#include <bits/stdc++.h>

using namespace std;

void printSolution(vector<vector<int>>&board,int N)
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
            if(board[i][j])
                cout << "Q ";
            else
                cout<<". ";
            printf("\n");
        }
    }

bool isSafe(vector<vector<int>>&board, int row, int col,int N)
{
    int i, j;
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return false;
    return true;
}
```

```

bool solveNQUtil(vector<vector<int>>&board, int col,int N)
{
    if (col >= N)
        return true;
    for (int i = 0; i < N; i++)
    {
        if (isSafe(board, i, col,N))
        {
            board[i][col] = 1;
            if (solveNQUtil(board, col + 1,N))
                return true;
            board[i][col] = 0;
        }
    }
    return false;
}

bool solveNQ()
{
    int n;
    cout<<"Enter size of board(n):";
    cin>>n;
    vector<vector<int>>board(n,vector<int>(n,0));
    if (solveNQUtil(board, 0,n) == false)
    {
        cout << "Solution does not exist";
        return false;
    }
    printSolution(board,n);
    return true;
}

int main()
{
    solveNQ();
    return 0;
}

```

```
}
```

OUTPUT:

```
Enter size of board(n):4
```

```
. . Q .
```

```
Q . . .
```

```
. . . Q
```

```
. Q . .
```

```
|
```



