

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Analysis and Design of Algorithms

Submitted by

BHARATH M (1BM22CS405)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

June-2023 to September-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by **Bharath M (1BM22CS405)**, who is a bonafide student of **B.M.S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to September-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Prof. Prameetha Pai
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Lab Program No.	Program Details	Page No.
1	Write program to do the following: a. Print all the nodes reachable from a given starting node in a di-graph using BFS method. b. Check whether a given graph is connected or not using DFS method.	1
2	Write a program to obtain the Topological ordering of vertices in a given digraph.	5
3	Implement Johnson Trotter algorithm to generate permutations.	9
4	Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.	15
5	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	20
6	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	24
7	Implement 0/1 Knapsack problem using dynamic programming.	29
8	Implement All Pair Shortest paths problem using Floyd's algorithm.	32
9	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm.	36
10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	42
11	Implement "N-Queens Problem" using Backtracking.	45

Course Outcome

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete.
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

Laboratory Program 1

Write program to do the following:

- Print all the nodes reachable from a given starting node in a di-graph using BFS method.
- Check whether a given graph is connected or not using DFS method.

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

// Function to perform Breadth First Search (BFS)
void BFS(int** graph, int numNodes, int startNode) {
    bool* visited = (bool*)malloc(numNodes * sizeof(bool)); // Keep track of visited nodes
    for (int i = 0; i < numNodes; ++i) {
        visited[i] = false;
    }

    int* queue = (int*)malloc(numNodes * sizeof(int)); // Queue for BFS traversal
    int front = 0, rear = 0;

    visited[startNode] = true; // Mark the start node as visited
    queue[rear++] = startNode; // Enqueue the start node

    printf("Nodes reachable from %d using BFS: ", startNode);
    while (front < rear) {
        int currentNode = queue[front++];
        printf("%d ", currentNode);

        // Enqueue all the unvisited neighbors of the current node
        for (int neighbor = 0; neighbor < numNodes; ++neighbor) {
            if (graph[currentNode][neighbor] && !visited[neighbor]) {
                visited[neighbor] = true;
                queue[rear++] = neighbor;
            }
        }
    }
    printf("\n");
}
```

```
    free(visited);
    free(queue);
}
```

```
// Depth First Search (DFS) helper function
```

```
void DFSUtil(int** graph, int node, bool* visited, int numNodes) {
    visited[node] = true;

    // Recursive call for all unvisited neighbors
    for (int neighbor = 0; neighbor < numNodes; ++neighbor) {
        if (graph[node][neighbor] && !visited[neighbor]) {
            DFSUtil(graph, neighbor, visited, numNodes);
        }
    }
}
```

```
// Function to perform Depth First Search (DFS)
```

```
bool DFS(int** graph, int numNodes) {
    bool* visited = (bool*)malloc(numNodes * sizeof(bool)); // Keep track of visited nodes
    for (int i = 0; i < numNodes; ++i) {
        visited[i] = false;
    }
}
```

```
DFSUtil(graph, 0, visited, numNodes); // Start DFS traversal from node 0
```

```
// Check if all nodes were visited
```

```
for (int i = 0; i < numNodes; ++i) {
    if (!visited[i]) {
        free(visited);
        return false; // Graph is not connected
    }
}
```

```
free(visited);
return true; // Graph is connected
}
```

```
int main() {
```

```

int numNodes;
printf("Enter the number of nodes in the graph: ");
scanf("%d", &numNodes);

int** graph = (int**)malloc(numNodes * sizeof(int*));
for (int i = 0; i < numNodes; ++i) {
    graph[i] = (int*)malloc(numNodes * sizeof(int));
}

printf("Enter the adjacency matrix (0 for no edge, 1 for edge exists):\n");
for (int i = 0; i < numNodes; ++i) {
    for (int j = 0; j < numNodes; ++j) {
        scanf("%d", &graph[i][j]);
    }
}

int startNode;
printf("Enter the starting node for BFS: ");
scanf("%d", &startNode);

BFS(graph, numNodes, startNode);

bool isConnected = DFS(graph, numNodes);
if (isConnected) {
    printf("The graph is connected.\n");
} else {
    printf("The graph is not connected.\n");
}

for (int i = 0; i < numNodes; ++i) {
    free(graph[i]);
}
free(graph);

return 0;
}

```

Output

Case 1:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of nodes in the graph: 4
Enter the adjacency matrix (0 for no edge, 1 for edge exists):
0 1 0 0
1 0 1 1
0 0 0 1
0 1 0 0
Enter the starting node for BFS: 1
Nodes reachable from 1 using BFS: 1 0 2 3
The graph is connected.
```

Case 2:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of nodes in the graph: 4
Enter the adjacency matrix (0 for no edge, 1 for edge exists):
0 1 1 0
1 0 1 0
1 1 0 0
0 0 0 0
Enter the starting node for BFS: 0
Nodes reachable from 0 using BFS: 0 1 2
The graph is not connected.
```


Laboratory Program 2

Write a program to obtain the Topological ordering of vertices in a given digraph.

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

#define MAX_VERTICES 100

struct Node {
    int data;
    struct Node* next;
};

struct Graph {
    int numVertices;
    struct Node* adjacencyList[MAX_VERTICES];
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

struct Graph* createGraph(int numVertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = numVertices;

    for (int i = 0; i < numVertices; ++i) {
        graph->adjacencyList[i] = NULL;
    }

    return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
```

```

    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjacencyList[src];
    graph->adjacencyList[src] = newNode;
}

void topologicalSortDFS(struct Graph* graph, int vertex, int visited[], struct Node* stack) {
    visited[vertex] = 1;

    struct Node* adjNode = graph->adjacencyList[vertex];
    while (adjNode != NULL) {
        int adjVertex = adjNode->data;
        if (!visited[adjVertex]) {
            topologicalSortDFS(graph, adjVertex, visited, stack);
        }
        adjNode = adjNode->next;
    }

    struct Node* newNode = createNode(vertex);
    newNode->next = stack->next;
    stack->next = newNode;
}

void topologicalSort(struct Graph* graph) {
    int visited[MAX_VERTICES] = {0};
    struct Node* stack = createNode(-1);

    for (int i = 0; i < graph->numVertices; ++i) {
        if (!visited[i]) {
            topologicalSortDFS(graph, i, visited, stack);
        }
    }

    struct Node* temp = stack->next;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

```

```
    free(stack);
}

int main() {
    int numVertices, numEdges;
    printf("Enter the number of vertices and edges: ");
    scanf("%d %d", &numVertices, &numEdges);

    struct Graph* graph = createGraph(numVertices);

    printf("Enter edges (src dest):\n");
    for (int i = 0; i < numEdges; ++i) {
        int src, dest;
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }

    printf("Topological Sort: ");
    topologicalSort(graph);

    return 0;
}
```

Output

Case 1:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of vertices and edges: 11 14
Enter edges (src dest):
0 4
0 2
1 0
1 2
2 3
3 7
3 5
4 5
5 6
6 8
7 6
7 9
9 8
8 10
Topological Sort: 1 0 4 2 3 7 9 5 6 8 10
C:\Users\Jothi\Documents\ADALAB>
```

Case 2:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of vertices and edges: 5 5
Enter edges (src dest):
0 1
0 2
0 3
3 2
3 4
Topological Sort: 0 1 3 2 4
C:\Users\Jothi\Documents\ADALAB>
```

Laboratory Program 3

Implement Johnson Trotter algorithm to generate permutations.

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

```
#include <stdbool.h>
```

```
#define LEFT_TO_RIGHT 1
```

```
#define RIGHT_TO_LEFT 0
```

```
int searchArr(int a[], int n, int mobile) {
```

```
    for (int i = 0; i < n; i++)
```

```
        if (a[i] == mobile)
```

```
            return i + 1;
```

```
    return -1;
```

```
}
```

```
int getMobile(int a[], bool dir[], int n) {
```

```
    int mobile_prev = 0, mobile = 0;
```

```
    for (int i = 0; i < n; i++) {
```

```
        if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0) {
```

```
            if (a[i] > a[i - 1] && a[i] > mobile_prev) {
```

```
                mobile = a[i];
```

```
                mobile_prev = mobile;
```

```
            }
```

```
    }
```

```

    if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1) {
        if (a[i] > a[i + 1] && a[i] > mobile_prev) {
            mobile = a[i];
            mobile_prev = mobile;
        }
    }
}

if (mobile == 0 && mobile_prev == 0)
    return 0;
else
    return mobile;
}

```

```

void printOnePerm(int a[], bool dir[], int n) {
    int mobile = getMobile(a, dir, n);
    int pos = searchArr(a, n, mobile);

    if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT) {
        int temp = a[pos - 1];
        a[pos - 1] = a[pos - 2];
        a[pos - 2] = temp;
    } else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT) {

```

```
int temp = a[pos];  
a[pos] = a[pos - 1];  
a[pos - 1] = temp;  
}
```

```
for (int i = 0; i < n; i++) {  
    if (a[i] > mobile) {  
        if (dir[a[i] - 1] == LEFT_TO_RIGHT)  
            dir[a[i] - 1] = RIGHT_TO_LEFT;  
        else if (dir[a[i] - 1] == RIGHT_TO_LEFT)  
            dir[a[i] - 1] = LEFT_TO_RIGHT;  
    }  
}
```

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

```
int fact(int n) {  
    int res = 1;  
    for (int i = 1; i <= n; i++)  
        res = res * i;  
    return res;  
}
```

```
}
```

```
void printPermutation(int n) {
```

```
    int a[n];
```

```
    bool dir[n];
```

```
    for (int i = 0; i < n; i++) {
```

```
        a[i] = i + 1;
```

```
        printf("%d ", a[i]);
```

```
    }
```

```
    printf("\n");
```

```
    for (int i = 0; i < n; i++)
```

```
        dir[i] = RIGHT_TO_LEFT;
```

```
    for (int i = 1; i < fact(n); i++)
```

```
        printOnePerm(a, dir, n);
```

```
}
```

```
int main() {
```

```
    int n;
```

```
    printf("Enter the value of n: ");
```

```
    scanf("%d", &n);
```

```
    printf("Number of permutations:%d\n",fact(n));
```



```
printPermutation(n);  
  
return 0;  
}
```

Output

Case 1:

```
C:\Users\Jothi\Documents\ADALAB>a  
Enter the value of n: 4  
Number of permutations:24  
1 2 3 4  
1 2 4 3  
1 4 2 3  
4 1 2 3  
4 1 3 2  
1 4 3 2  
1 3 4 2  
1 3 2 4  
3 1 2 4  
3 1 4 2  
3 4 1 2  
4 3 1 2  
4 3 2 1  
3 4 2 1  
3 2 4 1  
3 2 1 4  
2 3 1 4  
2 3 4 1  
2 4 3 1  
4 2 3 1  
4 2 1 3  
2 4 1 3  
2 1 4 3  
2 1 3 4
```

Case 2:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the value of n: 3
Number of permutations:6
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
```

Laboratory Program 4

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

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

void merge(int array[], int left, int mid, int right) {
    int subArrayOne = mid - left + 1;
    int subArrayTwo = right - mid;

    int *leftArray = (int *)malloc(subArrayOne * sizeof(int));
    int *rightArray = (int *)malloc(subArrayTwo * sizeof(int));

    for (int i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
    for (int j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

    int indexOfSubArrayOne = 0, indexOfSubArrayTwo = 0;
    int indexOfMergedArray = left;

    while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        } else {
            array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }

    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
    }
```

```

        indexOfMergedArray++;
    }

    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }

    free(leftArray);
    free(rightArray);
}

void mergeSort(int array[], int begin, int end) {
    if (begin >= end)
        return;

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

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

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    clock_t start, end;

    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100;
    }
}

```

```

}

int arr_size = sizeof(arr) / sizeof(arr[0]);
//printf("Given array is:\n");
//printArray(arr, arr_size);

start = clock();
mergeSort(arr, 0, arr_size - 1);
end = clock();

//printf("\nSorted array is:\n");
//printArray(arr, arr_size);
    printf("Start : %ld\n",start);
    printf("End : %ld\n",end);
    printf("CLOCKS_PER_SEC : %ld\n",CLOCKS_PER_SEC);
printf("Time taken is: %lf seconds\n", ((end - start)*1.0/ CLOCKS_PER_SEC));

return 0;
}

```

Output

Case 1:

```

C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 1000
Start : 2104
End : 2105
CLOCKS_PER_SEC : 1000
Time taken is: 0.001000 seconds

```

Case 2:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 5000
Start : 3079
End : 3081
CLOCKS_PER_SEC : 1000
Time taken is: 0.002000 seconds
```

Case 3:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 10000
Start : 2296
End : 2300
CLOCKS_PER_SEC : 1000
Time taken is: 0.004000 seconds
```

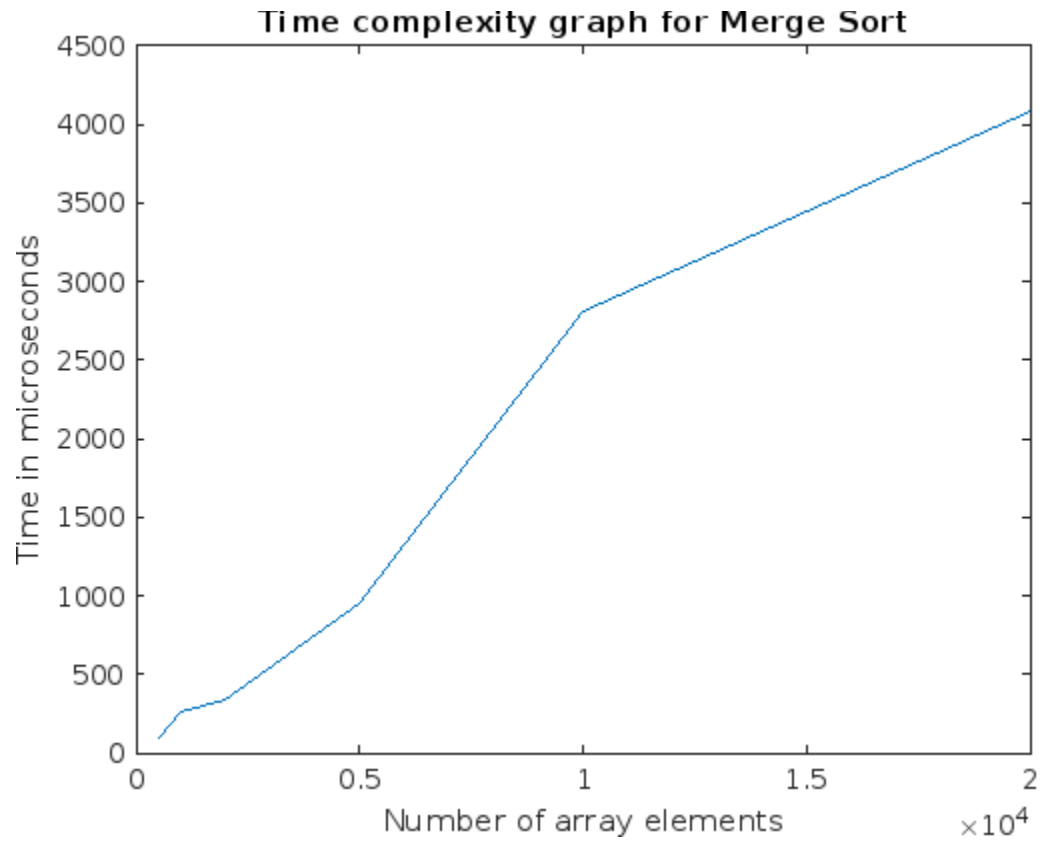
Case 4:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 30000
Start : 2997
End : 3007
CLOCKS_PER_SEC : 1000
Time taken is: 0.010000 seconds
```

Case 5:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 100000
Start : 4697
End : 4728
CLOCKS_PER_SEC : 1000
Time taken is: 0.031000 seconds
```

Graph



```
x=[500,1000,2000,5000,10000,20000]
y=[95,260,343,956,2811,4086]
plot(x,y)
xlabel('Number of array elements')
ylabel('Time in microseconds')
title('Time complexity graph for Merge Sort')
```

Laboratory Program 5

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

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

int partition(int arr[], int low, int high) {
    int p = arr[high];
    int i = low - 1;
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < p) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return i + 1;
}

void quicksort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quicksort(arr, low, pi - 1);
        quicksort(arr, pi + 1, high);
    }
}

int main() {
    clock_t start, end;
    int n;
    printf("Enter the number of elements: ");
```



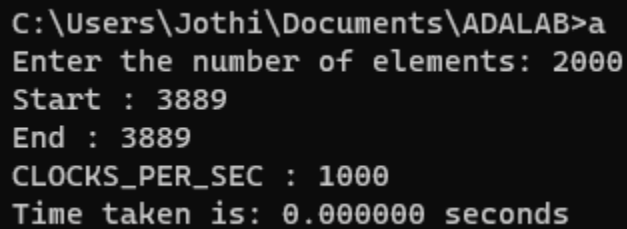
```
scanf("%d", &n);
int arr[n];
for (int i = 0; i < n; i++)
    arr[i] = rand()%100;

start = clock();
quicksort(arr, 0, n - 1);
end = clock();
printf("Start : %ld\n",start);
printf("End : %ld\n",end);
printf("CLOCKS_PER_SEC : %ld\n",CLOCKS_PER_SEC);
printf("Time taken is: %lf seconds\n", ((end - start)*1.0/ CLOCKS_PER_SEC));

return 0;
}
```

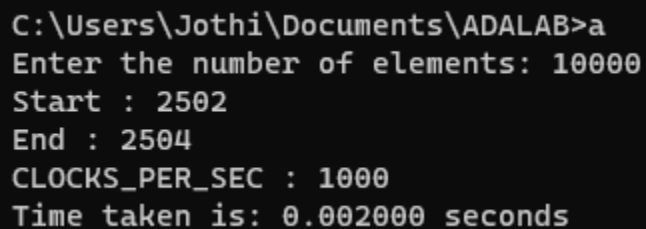
Output

Case 1:



```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 2000
Start : 3889
End : 3889
CLOCKS_PER_SEC : 1000
Time taken is: 0.000000 seconds
```

Case 2:



```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 10000
Start : 2502
End : 2504
CLOCKS_PER_SEC : 1000
Time taken is: 0.002000 seconds
```

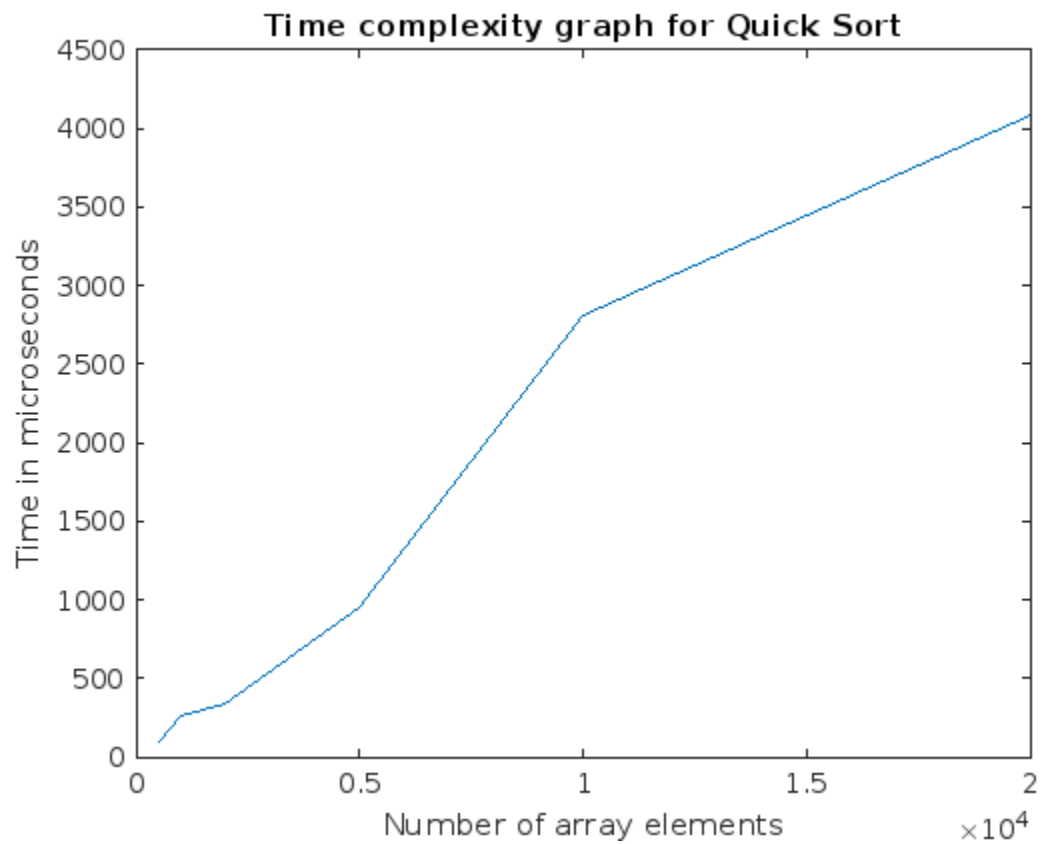
Case 3:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 100000
Start : 14825
End : 14929
CLOCKS_PER_SEC : 1000
Time taken is: 0.104000 seconds
```

Case 4:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 300000
Start : 5052
End : 5947
CLOCKS_PER_SEC : 1000
Time taken is: 0.895000 seconds
```

Graph



```
x=[500,1000,2000,5000,10000,20000]
y=[44,90,203,1164,1895,7443]
xlabel('Number of array elements')
ylabel('Time in microseconds')
title('Time complexity graph for Quick Sort')
```

Laboratory Program 6

Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

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

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

```
#include <time.h>
```

```
void heapify(int arr[], int n, int root) {
```

```
    int largest = root;
```

```
    int left = 2 * root + 1;
```

```
    int right = 2 * root + 2;
```

```
    if (left < n && arr[left] > arr[largest])
```

```
        largest = left;
```

```
    if (right < n && arr[right] > arr[largest])
```

```
        largest = right;
```

```
    if (largest != root) {
```

```
        int temp = arr[root];
```

```
        arr[root] = arr[largest];
```

```
        arr[largest] = temp;
```

```
        heapify(arr, n, largest);
```

```
    }
```

```
}
```

```
void heapSort(int arr[], int n) {
```

```
    for (int i = n / 2 - 1; i >= 0; i--)
```

```
        heapify(arr, n, i);
```

```
    for (int i = n - 1; i >= 0; i--) {
```

```
        int temp = arr[0];
```

```
        arr[0] = arr[i];
```

```
        arr[i] = temp;
```

```
        heapify(arr, i, 0);
```

```
    }
```

```
}
```

```
int main() {
```

```
    int n;
```

```
    printf("Enter the number of elements: ");
```

```
    scanf("%d", &n);
```

```
    int arr[n];
```

```
    for (int i = 0; i < n; i++)
```

```
        arr[i] = rand() % 100;
```

```
    clock_t start, end;
```

```
start = clock();

heapSort(arr, n);

end = clock();

printf("Start : %ld\n",start);

printf("End : %ld\n",end);

printf("CLOCKS_PER_SEC : %ld\n",CLOCKS_PER_SEC);

printf("Time taken is: %lf seconds\n", ((end - start)*1.0/ CLOCKS_PER_SEC));


return 0;

}
```

Output

Case 1:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 1000
Start : 2119
End : 2119
CLOCKS_PER_SEC : 1000
Time taken is: 0.000000 seconds
```

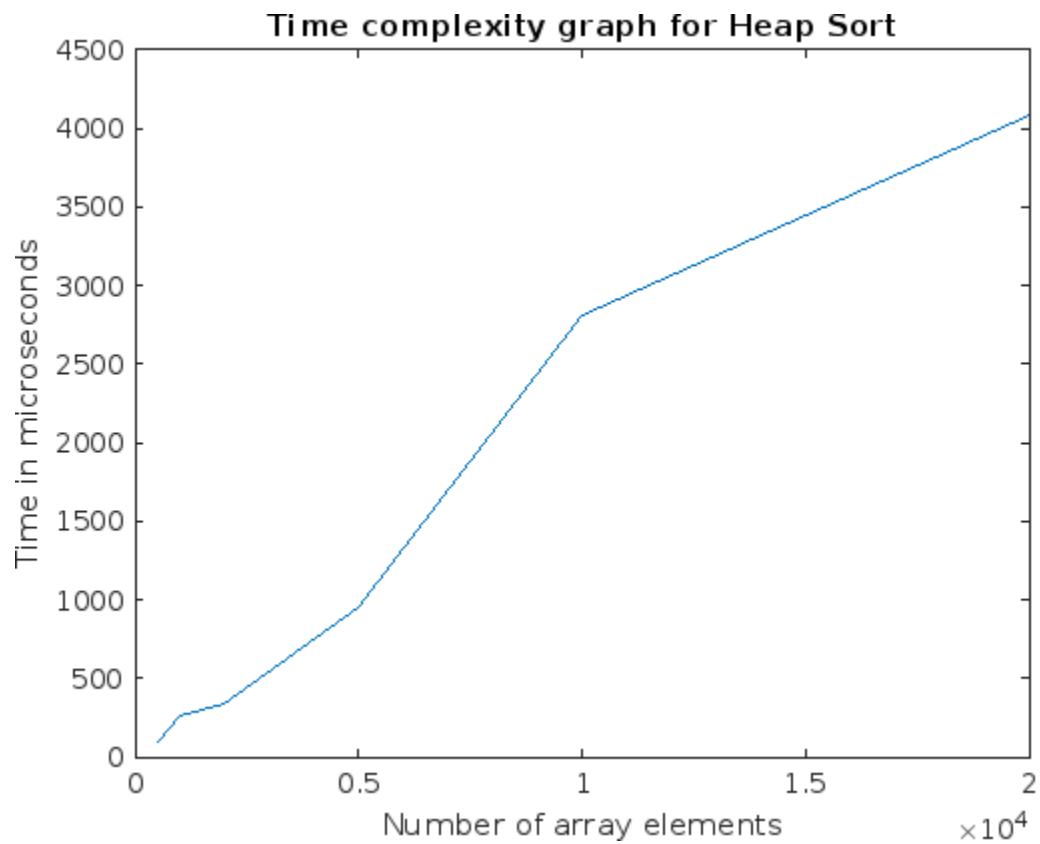
Case 2:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 10000
Start : 3495
End : 3497
CLOCKS_PER_SEC : 1000
Time taken is: 0.002000 seconds
```

Case 3:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of elements: 200000
Start : 2219
End : 2257
CLOCKS_PER_SEC : 1000
Time taken is: 0.038000 seconds
```

Graph



```
x=[500,1000,2000,5000,10000,20000]
y=[83,236,324,861,2416,5833]
xlabel('Number of array elements')
ylabel('Time in microseconds')
title('Time complexity graph for Heap Sort')
```


Laboratory Program 7

Implement 0/1 Knapsack problem using dynamic programming.

```
#include <stdio.h>

// Function to find maximum of two integers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Function to solve 0/1 Knapsack problem
int knapsack(int W, int wt[], int val[], int n) {
    int dp[n + 1][W + 1];

    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (wt[i - 1] <= w)
                dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]);
            else
                dp[i][w] = dp[i - 1][w];
        }
    }

    return dp[n][W];
}

int main() {
    int n; // Number of items
    printf("Enter the number of items: ");
    scanf("%d", &n);

    int val[n]; // Values of items
    int wt[n]; // Weights of items

    printf("Enter the values of items:\n");
```

```
for (int i = 0; i < n; i++)
    scanf("%d", &val[i]);

printf("Enter the weights of items:\n");
for (int i = 0; i < n; i++)
    scanf("%d", &wt[i]);

int W; // Maximum weight capacity of the knapsack
printf("Enter the maximum weight capacity of the knapsack: ");
scanf("%d", &W);

int result = knapsack(W, wt, val, n);
printf("Maximum value that can be obtained: %d\n", result);

return 0;
}
```

Output

Case 1:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of items: 4
Enter the values of items:
1 2 3 4
Enter the weights of items:
4 5 9 8
Enter the maximum weight capacity of the knapsack: 9
Maximum value that can be obtained: 4
```

Case 2:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of items: 6
Enter the values of items:
22 25 23 28 29 30
Enter the weights of items:
4 8 9 12 10 3
Enter the maximum weight capacity of the knapsack: 12
Maximum value that can be obtained: 55
```

Laboratory Program 8

Implement All Pair Shortest paths problem using Floyd's algorithm.

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

```
#include <limits.h>
```

```
#define INF INT_MAX
```

```
int min(int a, int b) {  
    return (a < b) ? a : b;  
}
```

```
int main() {  
    int v, i, j, k;  
    printf("Enter the number of vertices: ");  
    scanf("%d", &v);
```

```
    int arr[5][5], dist[5][5];
```

```
    printf("Enter the graph:\n");
```

```
    for (i = 0; i < v; i++) {  
        for (j = 0; j < v; j++) {  
            scanf("%d", &arr[i][j]);  
        }  
    }  
}
```

```
printf("\n");
```

```
for (i = 0; i < v; i++) {  
    for (j = 0; j < v; j++) {  
        dist[i][j] = arr[i][j];  
    }  
}
```

```
for (k = 0; k < v; k++) {  
    for (i = 0; i < v; i++) {  
        for (j = 0; j < v; j++) {  
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);  
        }  
    }  
}
```

```
printf("Distance matrix is:\n");
```

```
for (i = 0; i < v; i++) {  
    for (j = 0; j < v; j++) {  
        if (dist[i][j] == INF)  
            printf("INF ");  
        else  
            printf("%d ", dist[i][j]);  
    }  
}
```

```
    }  
    printf("\n");  
}  
  
return 0;  
}
```

Output

Case 1:

```
C:\Users\Jothi\Documents\ADALAB>a  
Enter the number of vertices: 5  
Enter the graph:  
0 99999 3 2 99999  
99999 0 99999 99999 99999  
99999 99999 0 99999 99999  
99999 99999 99999 0 99999  
1 99999 99999 5 0  
  
Distance matrix is:  
0 99999 3 2 99999  
99999 0 99999 99999 99999  
99999 99999 0 99999 99999  
99999 99999 99999 0 99999  
1 99999 4 3 0
```

Case 2:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of vertices: 4
Enter the graph:
0 99999 6 5
4 0 2 99999
99999 99999 0 99999
2 3 99999 0

Distance matrix is:
0 8 6 5
4 0 2 9
99999 99999 0 99999
2 3 5 0
```

Laboratory Program 9

Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm.

```
#include<stdio.h>
#include<conio.h>
void prims();
int cost[10][10],vt[10],et[10][10],vis[10],j,n;
int sum=0;
int x=1;
int e=0;
int find(int v,int parent[10])
{
    while(parent[v]!=v)
    {
        v=parent[v];
    }
    return v;
}

void union1(int i,int j,int parent[10])
{
    if(i<j)
        parent[j]=i;
    else
        parent[i]=j;
}

void kruskal(int n,int a[10][10])
{
    printf("Kruskal's Algorithm\n");
    int count,k,min,sum,i,j,t[10][10],u,v,parent[10];
    count=0;
    k=0;
    sum=0;
    for(i=0;i<n;i++)
        parent[i]=i;
```



```

while(count!=(n-1))
{
    min=999;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {

            if(a[i][j]<min && a[i][j]!=0)
            {
                min=a[i][j];
                u=i;
                v=j;
            }
        }
    }
    i=find(u,parent);
    j=find(v,parent);
    if(i!=j)
    {
        union1(i,j,parent);
        t[k][0]=u;
        t[k][1]=v;
        k++;
        count++;
        sum=sum+a[u][v];
    }
    a[u][v]=a[v][u]=999;
}
if(count==n-1)
{
    printf("Edges of the Minimal spanning tree are: ");
    for(i=0;i<n-1;i++)
    {
        printf("%d,%d\t",t[i][0]+1,t[i][1]+1);

    }
    printf("\nWeight of the minimal spanning tree is %d\n",sum);
}

```

```

    }
    else
        printf("Spanning tree does not exist.\n");
}

```

```

int main()
{
    int i,n1,ch,j;
    int cost1[10][10];
    printf("1.Prim's \n2.Kruskal's \nEnter your choice:");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:
            printf("Enter the number of vertices:");
            scanf("%d",&n);
            printf("Enter the cost adjacency matrix\n");
            for(i=1;i<=n;i++)
            {
                for(j=1;j<=n;j++)
                {
                    scanf("%d",&cost[i][j]);
                }
                vis[i]=0;
            }
            prims();
            printf("Prim's Algorithm\n");
            printf("Edges of the minimal spanning tree are: ");
            for(i=1;i<=e;i++)
            {
                printf("%d,%d\t",et[i][0],et[i][1]);
            }
            printf("\nWeight of the minimal spanning tree is %d\n",sum);
            break;
        case 2:

            printf("Enter the number of vertices:");
            scanf("%d",&n1);

```

```

printf("Enter the cost adjacency matrix\n");
for(i=0;i<n1;i++)
    for(j=0;j<n1;j++)
        scanf("%d",&cost1[i][j]);
kruskal(n1,cost1);

```

```

default:return 0;
}

```

```

}
/* Prim's algorithm */
void prims()
{
    int s,min,m,k,u,v;
    vt[x]=1;
    vis[x]=1;
    for(s=1;s<n;s++)
    {
        j=x;
        min=999;
        while(j>0)
        {
            k=vt[j];
            for(m=2;m<=n;m++)
            {
                if(vis[m]==0)
                {
                    if(cost[k][m]<min)
                    {
                        min=cost[k][m];
                        u=k;
                        v=m;
                    }
                }
            }
            j--;
        }
    }
}

```

```

}
vt[++x]=v;
et[s][0]=u;
et[s][1]=v;
e++;
vis[v]=1;
sum=sum+min;
}
}

```

Output

Case 1:

```

C:\Users\Jothi\Documents\ADALAB>a
1.Prim's
2.Kruskal's
Enter your choice:1
Enter the number of vertices:4
Enter the cost adjacency matrix
0 4 3 7
4 0 10 99999
3 10 0 11
7 99999 17 0
Prim's Algorithm
Edges of the minimal spanning tree are:1,3      1,2      1,4
Weight of the minimal spanning tree is 14

```

```

C:\Users\Jothi\Documents\ADALAB>a
1.Prim's
2.Kruskal's
Enter your choice:2
Enter the number of vertices:4
Enter the cost adjacency matrix
0 4 3 7
4 0 10 99999
3 10 0 11
7 99999 17 0
Kruskal's Algorithm
Edges of the Minimal spanning tree are: 1,3      1,2      1,4
Weight of the minimal spanning tree is 14

```

Case 2:

```
C:\Users\Jothi\Documents\ADALAB>a
1.Prim's
2.Kruskal's
Enter your choice:1
Enter the number of vertices:5
Enter the cost adjacency matrix
0 4 99999 8 99999
4 0 7 6 3
99999 7 0 99999 1
8 6 99999 0 2
99999 3 1 2 0
Prim's Algorithm
Edges of the minimal spanning tree are: 1,2    2,5    5,3    5,4
Weight of the minimal spanning tree is 10
```

```
C:\Users\Jothi\Documents\ADALAB>a
1.Prim's
2.Kruskal's
Enter your choice:2
Enter the number of vertices:5
Enter the cost adjacency matrix
0 4 99999 8 99999
4 0 7 6 3
99999 7 0 99999 1
8 6 99999 0 2
99999 3 1 2 0
Kruskal's Algorithm
Edges of the Minimal spanning tree are: 3,5    4,5    2,5    1,2
Weight of the minimal spanning tree is 10
```

Laboratory Program 10

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

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

int V;

int minDistance(int dist[], bool sptSet[])
{
    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;
}

void printSolution(int dist[])
{
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t\t %d\n", i+1, dist[i]);
}

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

```

        for (int count = 0; count < V - 1; count++)
        {
            int u = minDistance(dist, sptSet);
            sptSet[u] = true;
            for (int v = 0; v < V; 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];
        }
        printSolution(dist);
    }

int main()
{
    int i,j,src;
    printf("Enter the number of vertices:");
    scanf("%d",&V);
    int graph[V][V];
    printf("Enter the cost adjacency matrix:\n");
    for(i=0;i<V;i++)
    {
        for(j=0;j<V;j++)
            scanf("%d",&graph[i][j]);
    }
    printf("Enter the source vertex:");
    scanf("%d",&src);

    dijkstra(graph, src-1);

    return 0;
}

```

Output

Case 1:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of vertices:5
Enter the cost adjacency matrix:
0 4 99999 8 99999
4 0 7 6 3
99999 7 0 99999 1
8 6 99999 0 2
99999 3 1 2 0
Enter the source vertex:2
Vertex          Distance from Source
1                4
2                0
3                4
4                5
5                3
```

Case 2:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the number of vertices:4
Enter the cost adjacency matrix:
0 4 3 7
4 0 10 99999
3 10 0 11
7 99999 17 0
Enter the source vertex:1
Vertex          Distance from Source
1                0
2                4
3                3
4                7
```


Laboratory Program 11

Implement “N-Queens Problem” using Backtracking.

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

int N;

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

bool isSafe(int board[N][N], int row, int col)
{
    int i, j;

    // Check this row on left side
    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;

    // Check upper diagonal on left side
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return false;

    // Check lower diagonal on left side
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
```

```

        return false;

    return true;
}
bool solveNQUtil(int board[N][N], int col)
{
    // Base case: If all queens are placed
    // then return true
    if (col >= N)
        return true;

    // Consider this column and try placing
    // this queen in all rows one by one
    for (int i = 0; i < N; i++) {

        // Check if the queen can be placed on
        // board[i][col]
        if (isSafe(board, i, col)) {

            // Place this queen in board[i][col]
            board[i][col] = 1;

            // Recur to place rest of the queens
            if (solveNQUtil(board, col + 1))
                return true;

            // If placing queen in board[i][col]
            // doesn't lead to a solution, then
            // remove queen from board[i][col]
            board[i][col] = 0; // BACKTRACK
        }
    }

    // If the queen cannot be placed in any row in
    // this column col then return false
    return false;
}
bool solveNQ()

```

```

{
    int board[N][N];
    int i,j;
    for(i=0;i<N;i++)
    {
        for(j=0;j<N;j++)
            board[i][j]=0;
    }

    if (solveNQUtil(board, 0) == false)

    {
        printf("Solution does not exist");
        return false;
    }

    printSolution(board);
    return true;
}

// Driver program to test above function
int main()
{
    printf("Enter the value of N, if N X N is the size of the board:");
    scanf("%d",&N);
    solveNQ();
    return 0;
}

```

Output

Case 1:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the value of N, if N X N is the size of the board:2
Solution does not exist
```

Case 2:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the value of N, if N X N is the size of the board:1
Q
```

Case 3:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the value of N, if N X N is the size of the board:4
. . Q .
Q . . .
. . . Q
. Q . .
```

Case 4:

```
C:\Users\Jothi\Documents\ADALAB>a
Enter the value of N, if N X N is the size of the board:8
Q . . . . . . .
. . . . . Q .
. . . . Q . . .
. . . . . Q
. Q . . . . .
. . . Q . . . .
. . . . . Q .
. . Q . . . . .
```

