

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

Analysis and Design of Algorithms

Submitted by

RUSHILA V (1BM22CS226)

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

April-2024 to August-2024

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 **RUSHILA V (1BM22CS226)**, who is 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 April-2024 to August-2024. The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms (23CS4PCADA)** work prescribed for the said degree.

Madhavi R.P
Associate 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	Leetcode -1 :Find all numbers disappeared in an array	1
2	Leetcode-2 :Binary tree zigzag level order traversal	2
3	Leetcode-3: Increasing order search tree	3
4	Write program to obtain the Topological ordering of vertices in a given digraph.	4-10
5	Implement Johnson Trotter algorithm to generate permutations. Leetcode	11-16
6	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.	16-19
7	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	20-23
8	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	24-28
9	Implement 0/1 Knapsack problem using dynamic programming.	29-31
10	Implement All Pair Shortest paths problem using Floyd's algorithm.	31-33
11	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	33-36

	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.	36-40
12	Implement Fractional Knapsack using Greedy technique.	40-42
13	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	42-44
14	Implement "N-Queens Problem" using Backtracking.	45-47

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.

Leetcode -1 :Find all numbers disappeared in an array

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* findDisappearedNumbers(int* nums, int numsSize, int* returnSize){
    int temp=0;

    for (int index = 0; index < numsSize; ++index){
        temp = abs(nums[index])-1;
        nums[temp] = abs(nums[temp]) * -1;
    }

    int insert_index = 0;

    *returnSize = 0;
    for (int index = 0; index < numsSize; ++index)
    {
        if (nums[index] > 0){
            ++*returnSize;
            nums[insert_index++] = index + 1;
        }
    }

    return nums;
}
```

Output:

☒ Testcase | [Test Result](#)

Accepted Runtime: 1 ms

• Case 1

• Case 2

Input

nums =
[4,3,2,7,8,2,3,1]

Output

[5,6]

Expected

[5,6]

Leetcode-2 :Binary tree zigzag level order traversal

```
int** zigzagLevelOrder(struct TreeNode* root, int* returnSize, int** returnColumnSizes) {
    int **ans = malloc(2000*sizeof(int*));
    *returnColumnSizes = malloc(2000*sizeof(int));
    *returnSize = 0;
    struct TreeNode *tmp[2000] = {0};
    int top = -1, start = 0;
    tmp[++top] = root;
    while(tmp[start])
    {
        int tmp_top = top;
        ans[(*returnSize)] = malloc((top-start+1)*sizeof(int));
        (*returnColumnSizes)[(*returnSize)] = (top-start+1);
        int idx = (*returnSize)%2 ? (top-start+1)-1:0;
        int step = (*returnSize)%2 ? -1:1;
        while(start <= tmp_top)
        {
            ans[(*returnSize)][idx] = tmp[start]->val;
            if(tmp[start]->left)
                tmp[++top] = tmp[start]->left;
            if(tmp[start]->right)
                tmp[++top] = tmp[start]->right;
            start++;
            idx += step;
        }
        (*returnSize)++;
    }
    return ans;
}
```

Output :

☒ Testcase | [Test Result](#)

Accepted Runtime: 3 ms

• Case 1

• Case 2

• Case 3

Input

root =
[3,9,20,null,null,15,7]

Output

[[3],[20,9],[15,7]]

Expected

[[3],[20,9],[15,7]]

Leetcode-3: Increasing order search tree

```
void inorder(struct TreeNode* root, struct TreeNode** head, struct TreeNode** tail){
    if(root==NULL) return;
    inorder(root->left, head, tail);
    if(*head==NULL){
        *head=root;
    }
    else
    {
        (*tail)->right=root;
    }
    *tail=root;
    root->left=NULL;
    inorder(root->right, head, tail);
}

struct TreeNode* increasingBST(struct TreeNode* root) {
    struct TreeNode* head=NULL, *tail=NULL;
    inorder(root, &head, &tail);
    return head;
}
```

Output :

Accepted Runtime: 1 ms

• Case 1 • Case 2

Input

root =
[5,3,6,2,4,null,8,1,null,null,null,7,9]

Output

[1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]

Expected

[1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]

Program 4

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

a)Topological sort using source removal

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int graph[MAX][MAX];

int in_degree[MAX];

int queue[MAX];

int front = -1, rear = -1;

void enqueue(int vertex) {
    if (rear == MAX - 1) {
        printf("Queue overflow\n");
        return;
    }
    if (front == -1) {
        front = 0;
    }
    queue[++rear] = vertex;
}

int dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue underflow\n");
        return -1;
    }
}
```



```

        return queue[front++];
    }

    int isEmpty() {
        return front == -1 || front > rear;
    }

    void topologicalSort(int n) {
        int topo_order[MAX];
        int topo_index = 0;
        for (int i = 0; i < n; i++) {
            if (in_degree[i] == 0) {
                enqueue(i);
            }
        }

        while (!isEmpty()) {
            int vertex = dequeue();
            topo_order[topo_index++] = vertex;
            for (int i = 0; i < n; i++) {
                if (graph[vertex][i] == 1) {
                    in_degree[i]--;
                    if (in_degree[i] == 0) {
                        enqueue(i);
                    }
                }
            }
        }
    }

```

```

    }

    if (topo_index != n) {
        printf("Graph has a cycle\n");
        return;
    }

    printf("Topological Order: ");
    for (int i = 0; i < topo_index; i++) {
        printf("%d ", topo_order[i]);
    }
    printf("\n");
}

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

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            graph[i][j] = 0;
        }
        in_degree[i] = 0;
    }

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {

```

```
scanf("%d", &graph[i][j]);  
if (graph[i][j] == 1) {  
    in_degree[j]++;  
}  
}  
}  
  
topologicalSort(n);  
return 0;  
}
```

OUTPUT:

```
Enter the number of vertices: 5  
Enter the adjacency matrix:  
0 0 1 0 0  
0 0 1 0 0  
0 0 0 1 1  
0 0 0 0 1  
0 0 0 0 0  
Topological Order: 0 1 2 3 4
```

b)Topological Sort using DFS

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int graph[MAX][MAX];

int visited[MAX];

int stack[MAX];

int top = -1;

void push(int vertex) {
    if (top == MAX - 1) {
        printf("Stack overflow\n");
        return;
    }
    stack[++top] = vertex;
}

int pop() {
    if (top == -1) {
        printf("Stack underflow\n");
        return -1;
    }
    return stack[top--];
}

void dfs(int vertex, int n) {
    visited[vertex] = 1;
    for (int i = 0; i < n; i++) {
```

```

        if (graph[vertex][i] == 1 && !visited[i]) {
            dfs(i, n);
        }
    }
    push(vertex);
}

void topologicalSort(int n) {
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i, n);
        }
    }
    printf("Topological Order: ");
    while (top != -1) {
        printf("%d ", pop());
    }
    printf("\n");
}

int main() {
    int n;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            graph[i][j] = 0;
        }
        visited[i] = 0;
    }
}

```

```
}  
printf("Enter the adjacency matrix:\n");  
for (int i = 0; i < n; i++) {  
    for (int j = 0; j < n; j++) {  
        scanf("%d", &graph[i][j]);  
    }  
}  
topologicalSort(n);  
return 0;  
}
```

OUTPUT :

```
Enter the number of vertices: 7  
Enter the adjacency matrix:  
0 1 1 0 0 0 0  
0 0 1 0 1 0 1  
0 0 0 0 0 1 0  
1 1 1 0 0 1 1  
0 0 0 0 0 0 0  
0 0 0 0 0 0 0  
0 0 0 0 1 1 0  
Topological Order: 3 0 1 6 4 2 5
```

Program 5

Implement Johnson Trotter algorithm to generate permutations.

```
#include <stdio.h>

#include <stdlib.h>

int flag = 0;

void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int search(int arr[], int num, int mobile) {
    int g;
    for (g = 0; g < num; g++) {
        if (arr[g] == mobile) {
            return g + 1;
        } else {
            flag++;
        }
    }
    return -1;
}

int find_Mobile(int arr[], int d[], int num) {
    int mobile = 0;
    int mobile_p = 0;
    int i;
    for (i = 0; i < num; i++) {
        if ((d[arr[i] - 1] == 0) && i != 0) {
```

```

        if (arr[i] > arr[i - 1] && arr[i] > mobile_p) {
            mobile = arr[i];
            mobile_p = mobile;
        } else {
            flag++;
        }
    } else if ((d[arr[i] - 1] == 1) && i != num - 1) {
        if (arr[i] > arr[i + 1] && arr[i] > mobile_p) {
            mobile = arr[i];
            mobile_p = mobile;
        } else {
            flag++;
        }
    } else {
        flag++;
    }
}

if ((mobile_p == 0) && (mobile == 0)) {
    return 0;
} else {
    return mobile;
}
}

void permutations(int arr[], int d[], int num) {
    int i;

    int mobile = find_Mobile(arr, d, num);

    int pos = search(arr, num, mobile);

```



```

    if (d[arr[pos - 1] - 1] == 0) {
        swap(&arr[pos - 1], &arr[pos - 2]);
    } else {
        swap(&arr[pos - 1], &arr[pos]);
    }
    for (i = 0; i < num; i++) {
        if (arr[i] > mobile) {
            if (d[arr[i] - 1] == 0) {
                d[arr[i] - 1] = 1;
            } else {
                d[arr[i] - 1] = 0;
            }
        }
    }
    for (i = 0; i < num; i++) {
        printf(" %d ", arr[i]);
    }
}

int factorial(int k) {
    int f = 1;
    int i = 0;
    for (i = 1; i < k + 1; i++) {
        f = f * i;
    }
    return f;
}

int main() {

```

```

int num = 0;
int i;
int j;
int z = 0;
printf("Johnson Trotter algorithm to find all permutations of given numbers\n");
printf("Enter the number\n");
scanf("%d", &num);
int arr[num], d[num];
z = factorial(num);
printf("Total permutations = %d\n", z);
printf("All possible permutations are:\n");
for (i = 0; i < num; i++) {
    d[i] = 0;
    arr[i] = i + 1;
    printf(" %d ", arr[i]);
}
printf("\n");
for (j = 1; j < z; j++) {
    permutations(arr, d, num);
    printf("\n");
}
return 0;
}

```

OUTPUT :

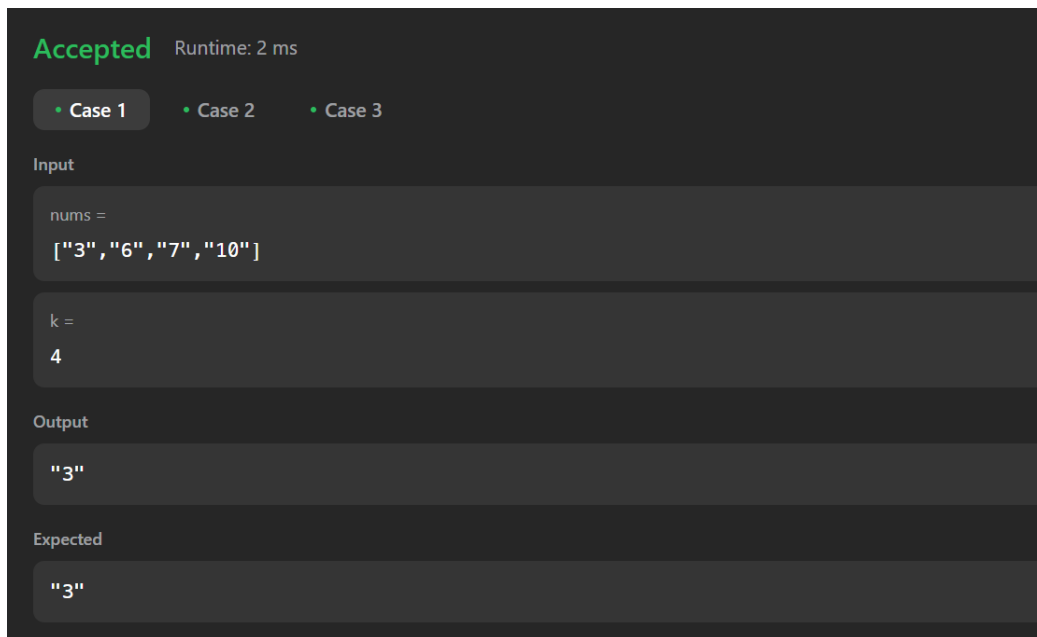
```
Johnson Trotter algorithm to find all permutations of given numbers
Enter the number
3
Total permutations = 6
All possible permutations are:
1  2  3
1  3  2
3  1  2
3  2  1
2  3  1
2  1  3
```

Leetcode : find kth largest element

```
int cmp(const void*a,const void*b) {
    const char* str1 = *(const char**)a;
    const char* str2 = *(const char**)b;
    if (strlen(str1) == strlen(str2)) {
        return strcmp(str1, str2);
    }
    return strlen(str1) - strlen(str2);
}

char * kthLargestNumber(char ** nums, int numsSize, int k){
    qsort(nums,numsSize,sizeof(char*),cmp);
    return nums[numsSize-k];
}
```

OUTPUT:



The screenshot shows a dark-themed interface for a code execution environment. At the top, it says 'Accepted' in green and 'Runtime: 2 ms' in white. Below this, there are three tabs: 'Case 1' (selected), 'Case 2', and 'Case 3'. Under the 'Case 1' tab, there is an 'Input' section with two text boxes: the first contains 'nums = ["3","6","7","10"]' and the second contains 'k = 4'. Below the input is an 'Output' section with a text box containing '"3"'. At the bottom is an 'Expected' section with a text box containing '"3"'. The entire interface is set against a dark background with light gray text.

Program 6

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

```
void split(int[], int, int);
void combine(int[], int, int, int);
```

```
void main() {
    int a[15000], n, i, j, ch, temp;
    clock_t start, end;
```

```
    while (1) {
```

```

printf("\n1: For manual entry of N value and array elements");
printf("\n2: To display time taken for sorting number of elements N in the range 500 to
14500");
printf("\n3: To exit");
printf("\nEnter your choice: ");
scanf("%d", &ch);

switch (ch) {
    case 1:
        printf("\nEnter the number of elements: ");
        scanf("%d", &n);
        printf("Enter array elements: ");
        for (i = 0; i < n; i++) {
            scanf("%d", &a[i]);
        }
        start = clock();
        split(a, 0, n - 1);
        end = clock();
        printf("\nSorted array is: ");
        for (i = 0; i < n; i++) {
            printf("%d\t", a[i]);
        }
        printf("\nTime taken to sort %d numbers is %f Secs\n", n, (((double)(end - start)) /
CLOCKS_PER_SEC));
        break;

    case 2:
        n = 500;
        while (n <= 14500) {
            for (i = 0; i < n; i++) {
                a[i] = n - i;
            }
            start = clock();
            split(a, 0, n - 1);
            for (j = 0; j < 90000000; j++) {
                temp = 38 / 600;
            }
            end = clock();
            printf("\nTime taken to sort %d numbers is %f Secs\n", n, (((double)(end - start)) /
CLOCKS_PER_SEC));
            n = n + 1000;
        }
    }
}

```

```

        }
        break;

        case 3:
            exit(0);
    }
    getchar();
}
}

void split(int a[], int low, int high) {
    int mid;
    if (low < high) {
        mid = (low + high) / 2;
        split(a, low, mid);
        split(a, mid + 1, high);
        combine(a, low, mid, high);
    }
}

void combine(int a[], int low, int mid, int high) {
    int c[15000], i, j, k;
    i = k = low;
    j = mid + 1;
    while (i <= mid && j <= high) {
        if (a[i] < a[j]) {
            c[k] = a[i];
            ++k;
            ++i;
        } else {
            c[k] = a[j];
            ++k;
            ++j;
        }
    }
    while (i <= mid) {
        c[k] = a[i];
        ++k;
        ++i;
    }
    while (j <= high) {
        c[k] = a[j];
        ++k;
    }
}

```

```

        ++j;
    }
    for (i = low; i <= high; i++) {
        a[i] = c[i];
    }
}

```

OUTPUT:

```

1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:1

Enter the number of elements: 4

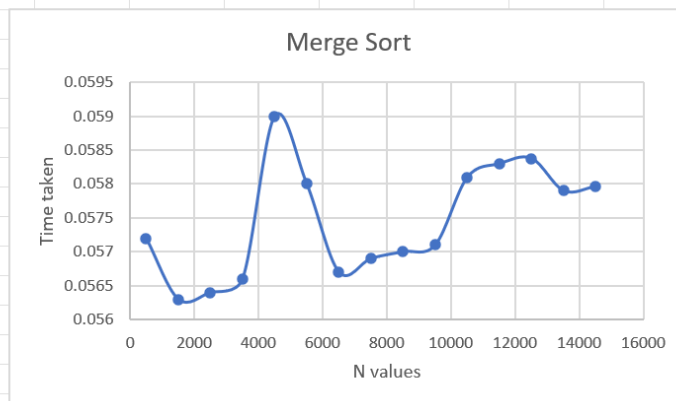
Enter array elements: 44 33 22 11

Sorted array is: 11      22      33      44
Time taken to sort 4 numbers is 0.000045 Secs
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:2

Time taken to sort 500 numbers is 0.057267 Secs
Time taken to sort 1500 numbers is 0.056371 Secs
Time taken to sort 2500 numbers is 0.056497 Secs
Time taken to sort 3500 numbers is 0.056614 Secs
Time taken to sort 4500 numbers is 0.059083 Secs
Time taken to sort 5500 numbers is 0.058599 Secs
Time taken to sort 6500 numbers is 0.056768 Secs
Time taken to sort 7500 numbers is 0.056926 Secs
Time taken to sort 8500 numbers is 0.057563 Secs
Time taken to sort 9500 numbers is 0.057136 Secs
Time taken to sort 10500 numbers is 0.058152 Secs
Time taken to sort 11500 numbers is 0.058349 Secs
Time taken to sort 12500 numbers is 0.058377 Secs
Time taken to sort 13500 numbers is 0.057963 Secs
Time taken to sort 14500 numbers is 0.057964 Secs

```

N values	Time taken
500	0.0572
1500	0.0563
2500	0.0564
3500	0.0566
4500	0.059
5500	0.058
6500	0.0567
7500	0.0569
8500	0.057
9500	0.0571
10500	0.0581
11500	0.0583
12500	0.058377
13500	0.0579
14500	0.057964



Program 7

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

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
void quickSort(int[], int, int);
int partition(int[], int, int);
void main() {
    int a[15000], n, i, j, ch, temp;
    clock_t start, end;

    while(1) {
        printf("\n1: For manual entry of N value and array elements");
        printf("\n2: To display time taken for sorting number of elements N in the range 500 to 14500");
        printf("\n3: To exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch(ch) {
            case 1:
                printf("\nEnter the number of elements: ");
                scanf("%d", &n);
                printf("\nEnter array elements: ");
                for(i = 0; i < n; i++) {
                    scanf("%d", &a[i]);
                }
            case 2:
            case 3:
            default:
                break;
        }
    }
}
```



```

    start = clock();
    quickSort(a, 0, n - 1);
    end = clock();
    printf("\nSorted array is: ");
    for(i = 0; i < n; i++) {
        printf("%d\t", a[i]);
    }

    printf("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end - start)) /
CLOCKS_PER_SEC));
    break;
case 2:
    n = 500;
    while(n <= 14500) {
        for(i = 0; i < n; i++) {
            a[i] = n - i;
        }
        start = clock();
        quickSort(a, 0, n - 1);
        for(j = 0; j < 900000000; j++) { temp = 38 / 600; }
        end = clock();

        printf("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end - start)) /
CLOCKS_PER_SEC));
        n = n + 1000;
    }
    break;
case 3:
    exit(0);
}

```

```

        getchar();
    }
}

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

```

```

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

```

```

    return (i + 1);
}

```

OUTPUT:

```

1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 1

Enter the number of elements: 5

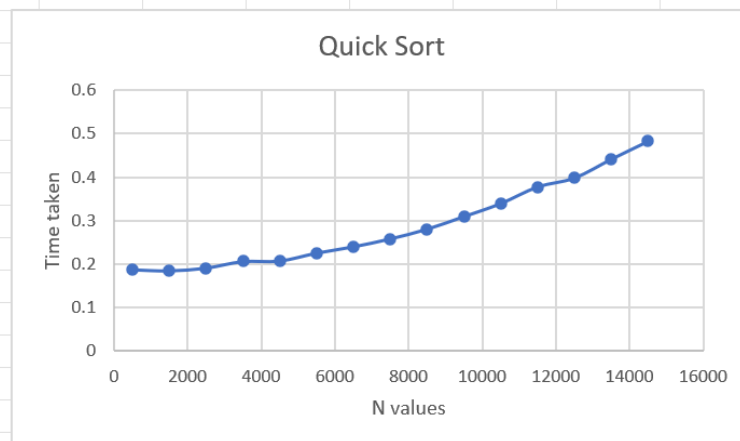
Enter array elements: 14 17 10 13 12

Sorted array is: 10    12    13    14    17
Time taken to sort 5 numbers is 0.000002 Secs
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 2

Time taken to sort 500 numbers is 0.187486 Secs
Time taken to sort 1500 numbers is 0.184879 Secs
Time taken to sort 2500 numbers is 0.190949 Secs
Time taken to sort 3500 numbers is 0.206965 Secs
Time taken to sort 4500 numbers is 0.207368 Secs
Time taken to sort 5500 numbers is 0.225479 Secs
Time taken to sort 6500 numbers is 0.240213 Secs
Time taken to sort 7500 numbers is 0.258185 Secs
Time taken to sort 8500 numbers is 0.280483 Secs
Time taken to sort 9500 numbers is 0.309480 Secs
Time taken to sort 10500 numbers is 0.339426 Secs
Time taken to sort 11500 numbers is 0.376617 Secs
Time taken to sort 12500 numbers is 0.398150 Secs
Time taken to sort 13500 numbers is 0.441232 Secs
Time taken to sort 14500 numbers is 0.482432 Secs

```

N values	Time taken
500	0.1874
1500	0.1848
2500	0.1909
3500	0.206
4500	0.2073
5500	0.2254
6500	0.24
7500	0.2581
8500	0.2804
9500	0.3094
10500	0.3394
11500	0.3776
12500	0.3981
13500	0.4412
14500	0.4824



Program 8

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 bottom_up_heapify(int n, int a[]);

void swap(int *a, int *b);

void heap_sort(int n, int a[]);

void bottom_up_heapify(int n, int a[]) {
    int p, item, c;
    for (p = (n-1)/2; p >= 0; p--) {
        item = a[p];
        c = 2 * p + 1;

        while (c <= n - 1) {
            if (c + 1 <= n - 1 && a[c] < a[c + 1]) {
                c++;
            }
            if (item >= a[c]) {
                break;
            }
            a[p] = a[c];
            p = c;
            c = 2 * p + 1;
        }
    }
}
```

```

        a[p] = item;
    }
}

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void heap_sort(int n, int a[]) {
    int i;
    bottom_up_heapify(n, a);

    for (i = n - 1; i >= 0; i--) {
        swap(&a[0], &a[i]);
        bottom_up_heapify(i, a);
    }
}

int main() {
    int a[15000], n, i, j, ch, temp;
    clock_t start, end;
    while (1) {
        printf("\n1: For manual entry of N value and array elements");

        printf("\n2: To display time taken for sorting number of elements N in the range 500 to 14500");

        printf("\n3: To exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
    }
}

```

```

switch (ch) {
    case 1:
        printf("\nEnter the number of elements: ");
        scanf("%d", &n);
        printf("\nEnter array elements: ");
        for (i = 0; i < n; i++) {
            scanf("%d", &a[i]);
        }
        start = clock();
        heap_sort(n, a);
        end = clock();
        printf("\nSorted array is: ");
        for (i = 0; i < n; i++)
            printf("%d\t", a[i]);

        printf("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end - start)) /
CLOCKS_PER_SEC));
        break;

    case 2:
        n = 500;
        while (n <= 14500) {
            for (i = 0; i < n; i++) {
                //a[i] = rand() % 1000;
                a[i] = n - i;
            }
            start = clock();
            heap_sort(n, a);
            for (j = 0; j < 500000; j++) {

```

```
        temp = 38 / 600;
    }
    end = clock();
    printf("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end - start)) /
CLOCKS_PER_SEC));
    n = n + 1000;
}
break;
case 3:
    exit(0);
}
getchar();
}
return 0;
}
```

OUTPUT:

```
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 1

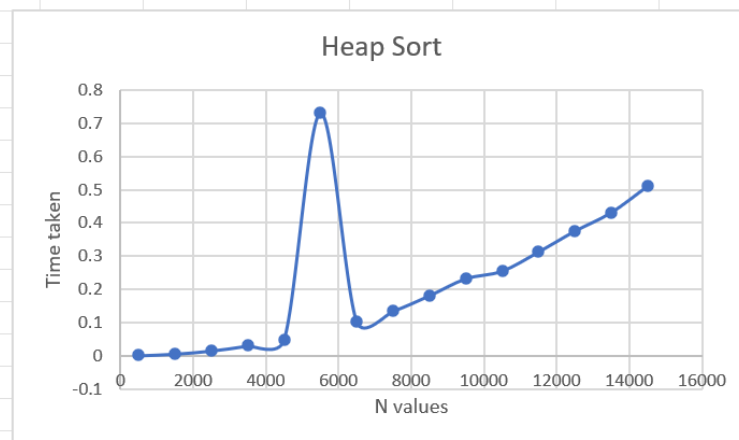
Enter the number of elements: 8

Enter array elements: 10 15 20 20 25 35 50 60

Sorted array is: 10    15    20    20    25    35    50    60
Time taken to sort 8 numbers is 0.000002 Secs
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500 to 14500
3: To exit
Enter your choice: 2

Time taken to sort 500 numbers is 0.001183 Secs
Time taken to sort 1500 numbers is 0.006062 Secs
Time taken to sort 2500 numbers is 0.015425 Secs
Time taken to sort 3500 numbers is 0.030444 Secs
Time taken to sort 4500 numbers is 0.049141 Secs
Time taken to sort 5500 numbers is 0.073357 Secs
Time taken to sort 6500 numbers is 0.103593 Secs
Time taken to sort 7500 numbers is 0.135304 Secs
Time taken to sort 8500 numbers is 0.181818 Secs
Time taken to sort 9500 numbers is 0.233097 Secs
Time taken to sort 10500 numbers is 0.256711 Secs
Time taken to sort 11500 numbers is 0.313318 Secs
Time taken to sort 12500 numbers is 0.376217 Secs
Time taken to sort 13500 numbers is 0.432017 Secs
Time taken to sort 14500 numbers is 0.511188 Secs
```

N values	Time taken
500	0.0011
1500	0.006
2500	0.0154
3500	0.0304
4500	0.0491
5500	0.733
6500	0.1035
7500	0.1353
8500	0.1818
9500	0.233
10500	0.256
11500	0.3133
12500	0.3762
13500	0.432
14500	0.511188



Program 9

Implement 0/1 Knapsack problem using dynamic programming.

```
#include <stdio.h>

#define MAX_ITEMS 100
#define MAX_CAPACITY 100

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

void knapsack(int n, int weights[], int profits[], int capacity) {
    int i, w;
    int K[MAX_ITEMS + 1][MAX_CAPACITY + 1];

    for (i = 0; i <= n; i++) {
        for (w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0) {
                K[i][w] = 0;
            } else if (weights[i - 1] <= w) {
                K[i][w] = max(profits[i - 1] + K[i - 1][w - weights[i - 1]], K[i - 1][w]);
            } else {
                K[i][w] = K[i - 1][w];
            }
        }
    }

    printf("Knapsack table:\n");
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= capacity; w++) {
            printf("%d\t", K[i][w]);
        }
    }
}
```

```

        printf("\n");
    }
    int totalProfit = K[n][capacity];
    printf("Maximum profit: %d\n", totalProfit);
    printf("Items included in the knapsack:\n");
    w = capacity;
    for (i = n; i > 0 && totalProfit > 0; i--) {
        if (totalProfit == K[i - 1][w]) {
            continue;
        } else {
            printf("Item %d (weight: %d, profit: %d)\n", i, weights[i - 1], profits[i - 1]);
            totalProfit -= profits[i - 1];
            w -= weights[i - 1];
        }
    }
}

int main() {
    int n = 4;
    int weights[] = {2, 3, 4, 5};
    int profits[] = {3, 4, 5, 6};
    int capacity = 5;
    knapsack(n, weights, profits, capacity);
    return 0;
}

```

OUTPUT:

```

Knapsack table:
0      0      0      0      0      0
0      0      3      3      3      3
0      0      3      4      4      7
0      0      3      4      5      7
0      0      3      4      5      7
Maximum profit: 7
Items included in the knapsack:
Item 2 (weight: 3, profit: 4)
Item 1 (weight: 2, profit: 3)

```

Program 10

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

```

#include <stdio.h>

#define V 5

#define INF 99999

void printSolution(int dist[][V]);
void floydWarshall(int dist[][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][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    printSolution(dist);
}

```

```

}

void printSolution(int dist[][V])
{
    printf(
        "The following matrix shows 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] == INF)
                printf("%7s", "INF");
            else
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    int graph[V][V] = { { 0, 4, INF, 5, INF },
                        { INF, 0, 1, INF, 6 },
                        { 2, INF, 0, 3, INF },
                        { INF, INF, 1, 0, 2 },
                        { 1, INF, INF, 4, 0 } };

    floydWarshall(graph);

    return 0;
}

```

OUTPUT:

The following matrix shows the shortest distances between every pair of vertices

0	4	5	5	7
3	0	1	4	6
2	6	0	3	5
3	7	1	0	2
1	5	5	4	0

Program 11

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

```
#include <stdio.h>

#include <limits.h>

#define INF 9999

#define MAX 100

void prims(int n, int cost[MAX][MAX]);

int main() {
    int n, i, j;

    int cost[MAX][MAX];

    printf("Enter the number of vertices: ");

    scanf("%d", &n);

    printf("Enter the cost adjacency matrix:\n");

    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);

            if (cost[i][j] == 0) {
                cost[i][j] = INF; // If there is no edge, set it to INF
            }
        }
    }
}
```

```

    }
    prims(n, cost);
    return 0;
}

void prims(int n, int cost[MAX][MAX]) {
    int i, j, k = 0, min, sum = 0;
    int source, u, v, flag = 0;
    int d[MAX], p[MAX], s[MAX], T[MAX][2];
    min = INF;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (cost[i][j] != 0 && cost[i][j] < min) {
                min = cost[i][j];
                source = i;
            }
        }
    }
    for (i = 0; i < n; i++) {
        s[i] = 0;
        d[i] = cost[source][i];
        p[i] = source;
    }
    s[source] = 1;
    for (i = 1; i < n; i++) {
        min = INF;
        u = -1;

```

```

    for (j = 0; j < n; j++) {
        if (s[j] == 0 && d[j] < min) {
            min = d[j];
            u = j;
        }
    }
    if (u == -1) {
        flag = 1;
        break;
    }
    T[k][0] = u;
    T[k][1] = p[u];
    k++;
    sum += cost[u][p[u]];
    s[u] = 1;
    for (v = 0; v < n; v++) {
        if (s[v] == 0 && cost[u][v] < d[v]) {
            d[v] = cost[u][v];
            p[v] = u;
        }
    }
}
if (flag || sum >= INF) {
    printf("Spanning tree does not exist\n");
} else {
    printf("Spanning tree exists and MST is:\n");
    for (i = 0; i < k; i++) {

```

```

        printf("%d - %d\n", T[i][0], T[i][1]);
    }
    printf("The cost of the spanning tree is %d\n", sum);
}
}

```

OUTPUT:

```

Enter the number of vertices: 5
Enter the cost adjacency matrix:
0 15 5 20 9999
5 0 25 9999 9999
15 25 0 30 37
20 9999 30 0 35
9999 9999 37 35 0
Spanning tree exists and MST is:
2 - 0
1 - 0
3 - 0
4 - 3
The cost of the spanning tree is 75

```

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

```

#include <stdio.h>

#include <stdlib.h>

#define MAX_VERTICES 100

struct Edge {
    int src, dest, weight;
};

```



```

struct Subset {
    int parent;
    int rank;
};

int find(struct Subset subsets[], int i);
void Union(struct Subset subsets[], int x, int y);
void KruskalMST(struct Edge edges[], int n, int e);
int compare(const void* a, const void* b) {
    struct Edge* edge1 = (struct Edge*)a;
    struct Edge* edge2 = (struct Edge*)b;
    return edge1->weight - edge2->weight;
}

int find(struct Subset subsets[], int i) {
    if (subsets[i].parent != i) {
        subsets[i].parent = find(subsets, subsets[i].parent);
    }
    return subsets[i].parent;
}

void Union(struct Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);
    if (subsets[xroot].rank < subsets[yroot].rank) {
        subsets[xroot].parent = yroot;
    } else if (subsets[xroot].rank > subsets[yroot].rank) {
        subsets[yroot].parent = xroot;
    } else {
        subsets[yroot].parent = xroot;
    }
}

```

```

        subsets[xroot].rank++;
    }
}

void KruskalMST(struct Edge edges[], int n, int e) {
    struct Edge result[n];
    int i = 0;
    int mincost = 0;
    struct Subset subsets[MAX_VERTICES];
    for (int v = 0; v < n; v++) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }
    qsort(edges, e, sizeof(struct Edge), compare);
    int edge_count = 0;
    while (edge_count < n - 1 && i < e) {
        struct Edge next_edge = edges[i++];
        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);
        if (x != y) {
            result[edge_count++] = next_edge;
            Union(subsets, x, y);
            mincost += next_edge.weight;
        }
    }
    printf("Edges in the Minimum Spanning Tree:\n");
    for (i = 0; i < edge_count; i++) {
        printf("%d - %d : %d\n", result[i].src, result[i].dest, result[i].weight);
    }
}

```

```

    }

    printf("Minimum Cost of MST: %d\n", mincost);
}

int main() {
    int n, i, j;
    struct Edge graph[MAX_VERTICES * MAX_VERTICES];
    int e = 0;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the cost adjacency matrix (enter 9999 for infinity):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            int weight;
            scanf("%d", &weight);
            if (weight != 9999 && i != j) {
                graph[e].src = i;
                graph[e].dest = j;
                graph[e].weight = weight;
                e++;
            }
        }
    }

    KruskalMST(graph, n, e);
    return 0;
}

```

OUTPUT:

```

Enter the number of vertices: 7
Enter the cost adjacency matrix (enter 9999 for infinity):
0 28 9999 9999 9999 10 9999
28 0 16 9999 9999 9999 14
9999 16 0 12 9999 9999 9999
9999 9999 12 0 22 9999 18
9999 9999 9999 22 0 25 24
10 9999 9999 9999 25 9999 9999
9999 14 9999 18 24 9999 9999
Edges in the Minimum Spanning Tree:
0 - 5 : 10
3 - 2 : 12
6 - 1 : 14
1 - 2 : 16
3 - 4 : 22
4 - 5 : 25
Minimum Cost of MST: 99

```

Program 12

Implement Fractional Knapsack using Greedy technique.

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

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

```

struct Item {
    int weight;
    int value;
    float value_per_unit_weight;
};

```

```

int compare_items(const void* a, const void* b) {
    struct Item *item1 = (struct Item *)a;
    struct Item *item2 = (struct Item *)b;

    if (item2->value_per_unit_weight > item1->value_per_unit_weight)
        return 1;
}

```

```

    else if (item2->value_per_unit_weight < item1->value_per_unit_weight)
        return -1;
    else
        return 0;
}

float fractional_knapsack(int capacity, struct Item items[], int n) {
    for (int i = 0; i < n; i++) {
        items[i].value_per_unit_weight = (float) items[i].value / items[i].weight;
    }
    qsort(items, n, sizeof(items[0]), compare_items);
    float total_value = 0.0;
    int current_weight = 0;
    for (int i = 0; i < n; i++) {
        if (current_weight + items[i].weight <= capacity) {
            total_value += items[i].value;
            current_weight += items[i].weight;
        } else {
            int remaining_capacity = capacity - current_weight;
            total_value += items[i].value_per_unit_weight * remaining_capacity;
            break;
        }
    }
    return total_value;
}

int main() {
    struct Item items[] = {

```

```

    {10, 60},
    {20, 100},
    {30, 120}
};

int capacity = 50;

int n = sizeof(items) / sizeof(items[0]);

float max_value = fractional_knapsack(capacity, items, n);

printf("Maximum value in Knapsack = %.2f\n", max_value);

return 0;
}

```

OUTPUT:

```
Maximum value in Knapsack = 240.00
```

Program 13

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

```

#include <stdio.h>

#include <limits.h>

#define MAX_VERTICES 100

void dijkstras(int c[MAX_VERTICES][MAX_VERTICES], int n, int src) {
    int dist[MAX_VERTICES];
    int vis[MAX_VERTICES];

    int count, u, i, j;

    for (i = 0; i < n; i++) {
        dist[i] = INT_MAX;
        vis[i] = 0;
    }
}

```

```

}
dist[src] = 0;
count = 0;
while (count < n-1) {
    int min_dist = INT_MAX;
    for (i = 0; i < n; i++) {
        if (!vis[i] && dist[i] < min_dist) {
            min_dist = dist[i];
            u = i;
        }
    }
    vis[u] = 1;
    for (j = 0; j < n; j++) {
        if (!vis[j] && c[u][j] && dist[u] != INT_MAX && dist[u] + c[u][j] < dist[j]) {
            dist[j] = dist[u] + c[u][j];
        }
    }
    count++;
}
printf("Shortest distances from source %d:\n", src);
for (i = 0; i < n; i++) {
    if (dist[i] == INT_MAX) {
        printf("%d -> %d : Unreachable\n", src, i);
    } else {
        printf("%d -> %d : %d\n", src, i, dist[i]);
    }
}
}

```

```

}

int main() {
    int n, src, i, j;
    int c[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the cost adjacency matrix (enter 9999 for infinity):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &c[i][j]);
        }
    }
    printf("Enter the source vertex: ");
    scanf("%d", &src);
    dijkstras(c, n, src);
    return 0;
}

```

OUTPUT:

```

Enter the number of vertices: 6
Enter the cost adjacency matrix (enter 9999 for infinity):
0 15 10 9999 45 9999
9999 0 15 9999 20 9999
20 9999 0 20 9999 9999
9999 10 9999 0 35 9999
9999 9999 9999 30 0 9999
9999 9999 9999 4 9999 0
Enter the source vertex: 5
Shortest distances from source 5:
5 -> 0 : 49
5 -> 1 : 14
5 -> 2 : 29
5 -> 3 : 4
5 -> 4 : 34
5 -> 5 : 0

```

Program 14

Implement “N-Queens Problem” using Backtracking.

```
#include <stdio.h>

#include <stdbool.h>

#define N 8

void print_solution(int board[N][N]);

bool is_safe(int board[N][N], int row, int col);

bool solve_n_queens(int board[N][N], int row);

void print_solution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%2d ", board[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}

bool is_safe(int board[N][N], int row, int col) {
    for (int i = 0; i < row; i++) {
        if (board[i][col])
            return false;
    }

    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j])
            return false;
    }
}
```

```

    }

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

    return true;
}

bool solve_n_queens(int board[N][N], int row) {
    if (row == N) {
        print_solution(board);
        return true;
    }
    bool result = false;
    for (int col = 0; col < N; col++) {
        if (is_safe(board, row, col)) {
            board[row][col] = 1;
            result = solve_n_queens(board, row + 1) || result;
            board[row][col] = 0;
        }
    }

    return result;
}

```

```

int main() {
    int board[N][N] = { {0} };

    if (!solve_n_queens(board, 0)) {
        printf("Solution does not exist for N = %d\n", N);
    }

    return 0;
}

```

OUTPUT:

```

0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0

0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0

0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0

```