VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

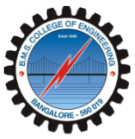
LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

RUSHI HUNDIWALA (1BM22CS224)

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 **RUSHI HUNDIWALA (1BM22CS224),** 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 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.

Name of the Lab-In charge: MADHAVI R.P	Dr. Jyothi S Nayak
Designation	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Lab Program No.	Program Details	Page No.
1	Leetcode exercises on Stacks, Queues, Circular Queues, Priority Queues.	
2	Leetcode exercises on DFS, BFS.(ZigZag Binary Tree)	
3	Leetcode exercises on Trees and Graphs.(Increasing Order Search Tree)	
4	Write a program to obtain the Topological ordering of vertices in a given digraph.	
5	Selection and 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.	
6	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	
7	Johnson Trotter	
8	● Sort a given set of N integer elements using Heap Sort technique and compute its time taken AND ● Implement All Pair Shortest paths problem using Floyd's algorithm.	
9	● Implement 0/1 Knapsack problem using dynamic programming. ● Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	
10	● From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. ● Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.	

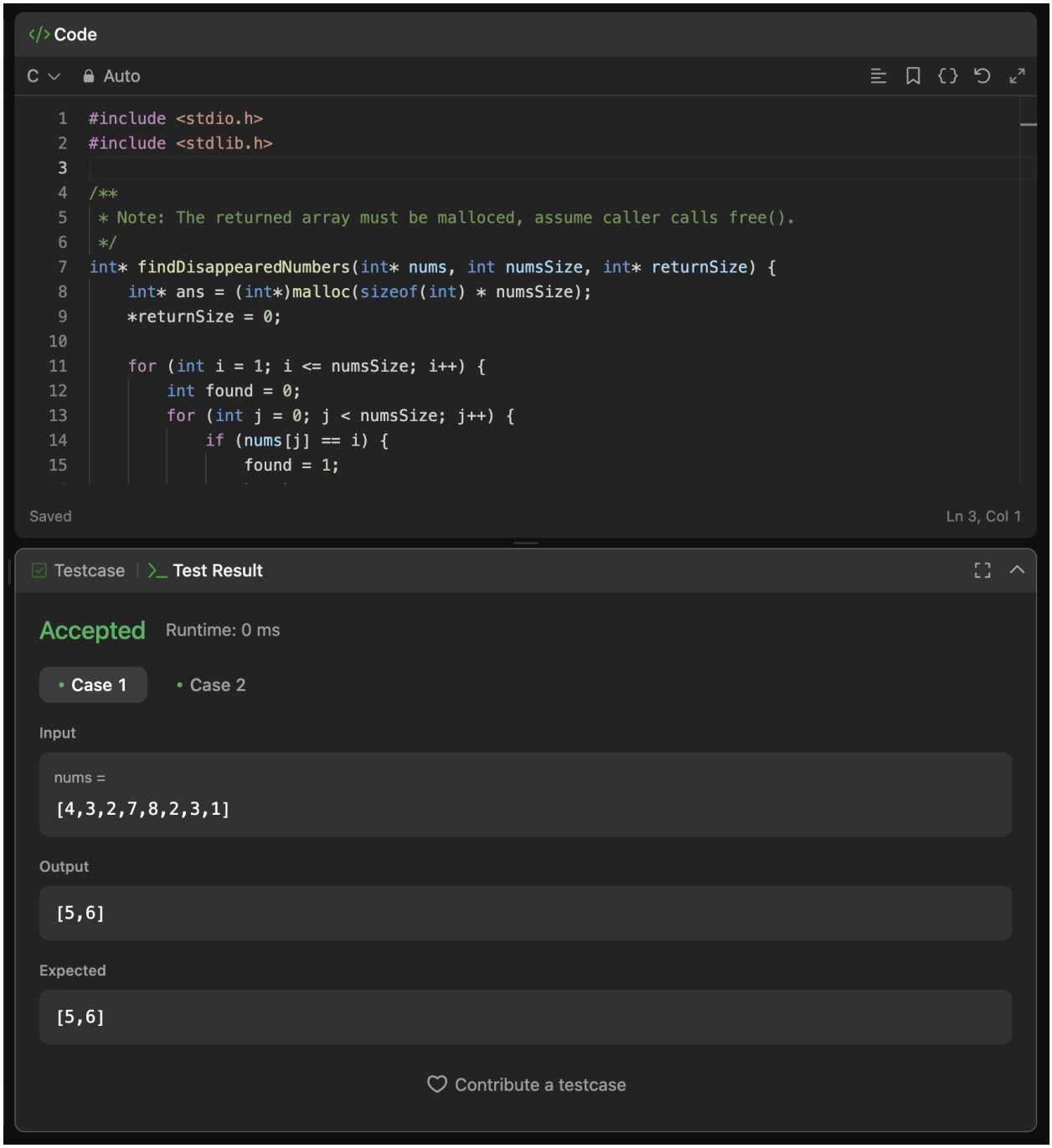
Course Outcome

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.

QUESTION: Leetcode exercises on Stacks, Queues, Circular Queues, Priority Queues.(Find Disappeared Array)

```
int * findDisappearedArray(int nums[], int numsize) {
	int result = (int*) malloc (sizeof (int) * numsize);
	for (int i=0; i&lt;numsize; i++) {
		int idx = abs (nums[i])-1;
		if (nums [idx] &gt; 0) {
			nums[idx] = - nums[idx];
		}
	}
int count = 0;
	for (int i = 0; i &lt; numssize; i++) {
		if (nums[i] &gt; 0) {
			result[count++] = i + 1;
		}
	}
	return result;
}
```

Result Screen Shot:



2/05/24 Lab. 1

```
int * findDisappearedArray(int *nums, int
                           numsSize) {
    int* result = (int *) malloc
                  (sizeof(int) * numsSize);
    for (int i=0 ; i < numsSize ; i++) {
        int idx = abs(nums[i]) - 1;
        if (nums[idx] > 0) {
            nums[idx] = -nums[idx];
        }
    }
    int count = 0;
    for (int i = 0 ; i< numsSize ;i++) {
        if (nums[i] > 0) {
            result[count++] = i+1;
        }
    }

    *returnSize = count;
    return result;
}
```

Outputs

input = 4, 3, 2, 7, 8, 2, 3, 1

output = [5, 6]

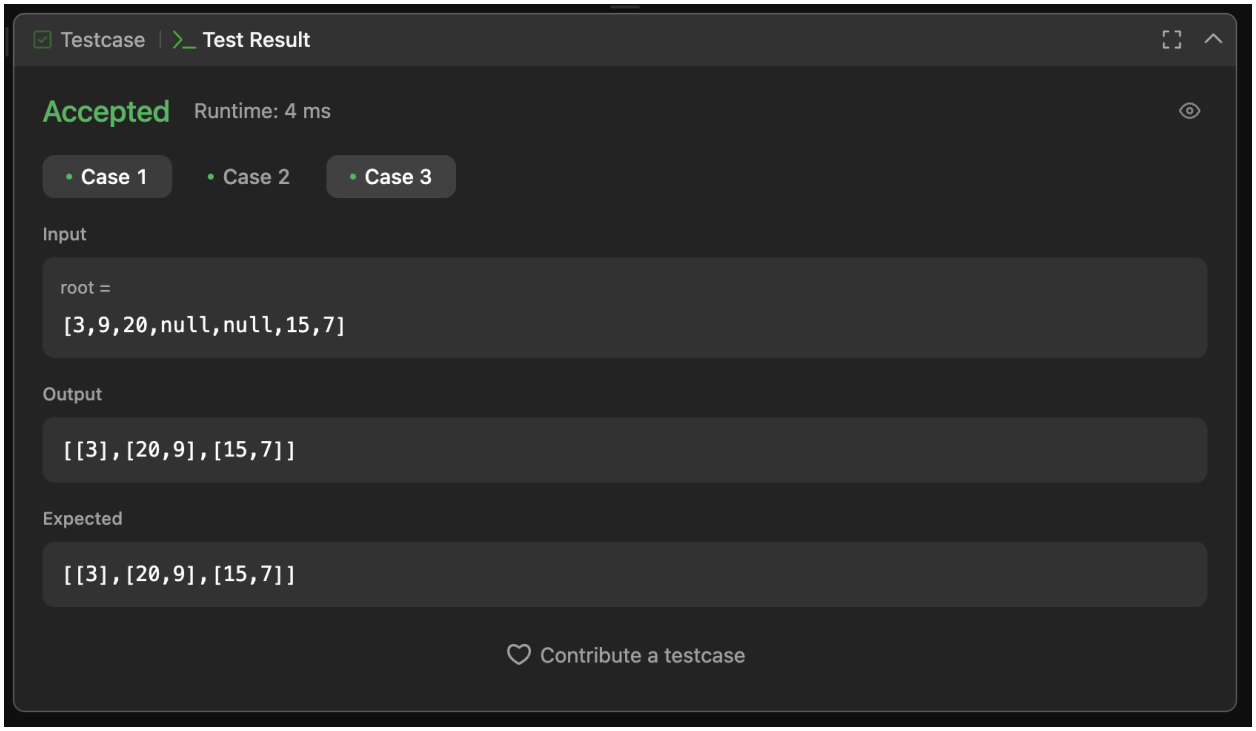
P. a15/24

QUESTION: Leetcode exercise (ZigZag Binary Tree)

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

#include <stdlib.h>

int** zigzagLevelOrder(struct TreeNode* root, int* returnSize, int**
returnColumnSizes) {

   if (root == NULL) {

       *returnSize = 0;

       *returnColumnSizes = NULL;

       return NULL;

   }


   int** result = (int**)malloc(sizeof(int*) * 2048);

   *returnColumnSizes = (int*)malloc(sizeof(int) * 2048);

   *returnSize = 0;


   struct TreeNode* queue[2048];

   int front = 0, rear = 0;

   queue[rear++] = root;


   int level = 0;

   while (front != rear) {

       int size = rear - front;

       result[*returnSize] = (int*)malloc(sizeof(int) * size);

       (*returnColumnSizes)[*returnSize] = size;
```

```
        for (int i = 0; i < size; i++) {
            struct TreeNode* node = queue[front++];
            if (level % 2 == 0) {
                result[*returnSize][i] = node->val;
            } else {
                result[*returnSize][size - i - 1] = node->val;
            }

            if (node->left) {
                queue[rear++] = node->left;
            }
            if (node->right) {
                queue[rear++] = node->right;
            }
        }

        (*returnSize)++;
        level++;
    }

    return result;
}
```

OUTPUT:


Testcase
Test Result
Accepted
Runtime: 4 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]]
Contribute a testcase

9/5/24

Lab 2

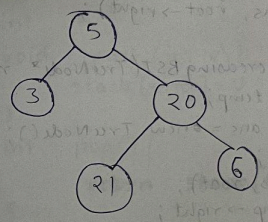
Binary Tree ZigZag

```
struct pair {
    struct TreeNode* node;
    int level;
};
struct Queue {
    struct Pair data[3000];
    int size;
    int i, j;
};
struct result {
    int **r;
    int* cols;
    int size;
    int cap;
}
void flush(struct Result* r){
    if(r->cur == NULL){
        return;
    }
    if(r->size >= r->cap){
        r->cap = (r->cap+1)*2;
        r->r = realloc(r->r, r->cap*sizeof(int*));
    r->r[r->size] = r->cur;
    r->cols[r->size] = r->cur_size;
    r->size++;
    r->cur = NULL;
    r->cur_size = r->cur_cap = 0;
}
```

```
void add(struct ...
    if (r->cur_size >= r->cur_cap){
        r->cur_cap = (r->cur_cap+1)*2;
        r->cur = realloc(r->cur, r->cur_cap*sizeof(int));
    }
    r->cur[r->cur_size++] = n;
}
void reverse(int* data, int size){
    int i;
    for(i=0; i<size/2; i++){
        int t = data[i];
        data[i] = data[size-i-1];
        data[size-i-1] = t;
    }
}
int** zigzagLevelOrder(struct TreeNode* root,
        int* returnSize
    , int** returnColumnSizes){
    struct Queue q;
    struct Result r;
    int level = 0, i;
    struct pair c = {root, level};
    memset(&q, 0, sizeof(q));
    memset(&r, 0, sizeof(r));
    if(root){
        qpush(&q, c);
    }
    while(!qempty(&q)){
        struct pair cur = qpop(&q);
        if(cur.level != level)
        add(&r, cur.node->val);
        if(cur.node->left){
            struct pair n = {cur.node->left,
```

```
    level = cur.level;
    flush(&r);
    return r.r;
}
```

Output:

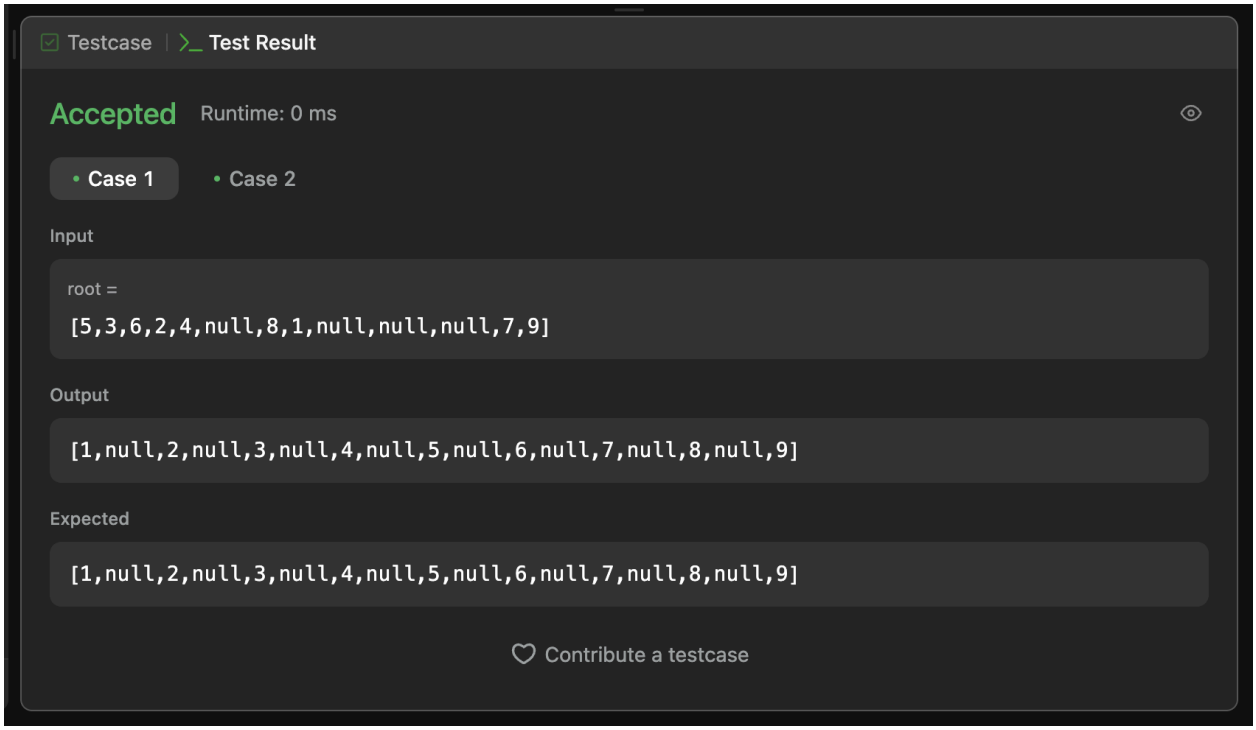


QUESTION: Leetcode exercises on Trees and Graphs.(Increasing Order Search Tree)

```
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};

struct TreeNode* head = (struct TreeNode*)malloc(sizeof(struct TreeNode));
struct TreeNode* ptr = head;

void in_order(struct TreeNode* root){
    if(root == NULL) return;

    in_order(root->left);

    head->right = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    head->right->val = root->val;
    head = head->right;

    in_order(root->right);
```

```
    return;
}

struct TreeNode* increasingBST(struct TreeNode* root) {
    in_order(root);
    return ptr->right;
}
```

SCREENSHOT OUTPUT:



16/5/24

Lab - 3

23/5/24

increasing order search tree.

```
void inorder (TreeNode*& ans, TreeNode* root){
    if(!root) return;
    inorder(ans, root->left);
    ans->right = new TreeNode(root->val);
    ans = ans->right;
    inorder(ans, root->right);
}
TreeNode* increasingBST(TreeNode* root) {
    TreeNode* temp;
    TreeNode* ans = new TreeNode();
    temp = ans;
    inorder(ans, root);
    return temp->right;
}
```

Output

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

23/5/24

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

```
//C program to implement topological sort using DFS
#include <stdio.h>
int n, a[10][10], res[10], s[10], top = 0;
void dfs(int, int, int[][10]);
void dfs_top(int, int[][10]);
int main()
{
printf("Enter the no. of nodes");
scanf("%d", &n);
int i, j;
for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
scanf("%d", &a[i][j]);
}
}
dfs_top(n, a);
printf("Solution: ");
for (i = n - 1; i >= 0; i--) {
printf("%d ", res[i]);
}
return 0;
}
void dfs_top(int n, int a[][10]) {
int i;
for (i = 0; i < n; i++) {
s[i] = 0;
}
```

```
for (i = 0; i < n; i++) {
if (s[i] == 0) {
dfs(i, n, a);
}
}
}
void dfs(int j, int n, int a[][10]) {
s[j] = 1;2
int i;
for (i = 0; i < n; i++) {
if (a[j][i] == 1 && s[i] == 0) {
dfs(i, n, a);
}
}
res[top++] = j;
}
```

OUTPUT:

```
Enter the no. of nodes6
0 0 1 1 0 0
0 0 0 1 1 0
0 0 0 1 0 1
0 0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 0 0
Solution: 1 4 0 2 3 5
```

23/5/24

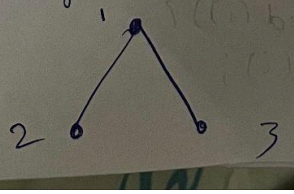
Topological sort.

```
# include <stdio.h>
int adj[MAX][MAX];
int visited[MAX];
int stack[MAX];
int top = -1
int n;
void push(int v){
    stack[++top] = v;
}
int pop(){
    return stack[top--]
}
void dfs(int v){
    visited[v] = 1;
    for(int i=0; i<n; i++){
        if(adj[v][i] && !visited[i]){
            dfs[i];
        }
        push(v);
void topologicalsort(){
    for(int i=0; i<n; i++){
        visited[i] = 0;
    }
    for(int i=0; i<n; i++){
        if(!visited[i]){
            dfs(i);
        }
}
```

```
    while (top != -1) {
        printf("%d", pop());
    }
    printf("\n");
}

int main() {
    int edges, start, end;
    printf("Enter the no. of vertices");
    scanf("%d", &n);
    printf("Enter the no. of edges:");
    scanf("%d", &edges);
    for (int i=0; i<edges; i++) {
        pf("Enter start & end vertices
            of edge %d", i+1);
        scanf("%d %d", &start, &end);
        adj[start][end] = 1;
    }
    printf("Topological order");
    topologicalSort();
    return 0;
}
```

Output

```
Enter the no. of vertices : 3
Enter the no. of edges : 2
Enter start & end vertex of edge 1: 1 2
Enter start & end vertex of edge 2: 1 3
Topological Sort: 1 2 0
```



QUESTION: Selection and 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.

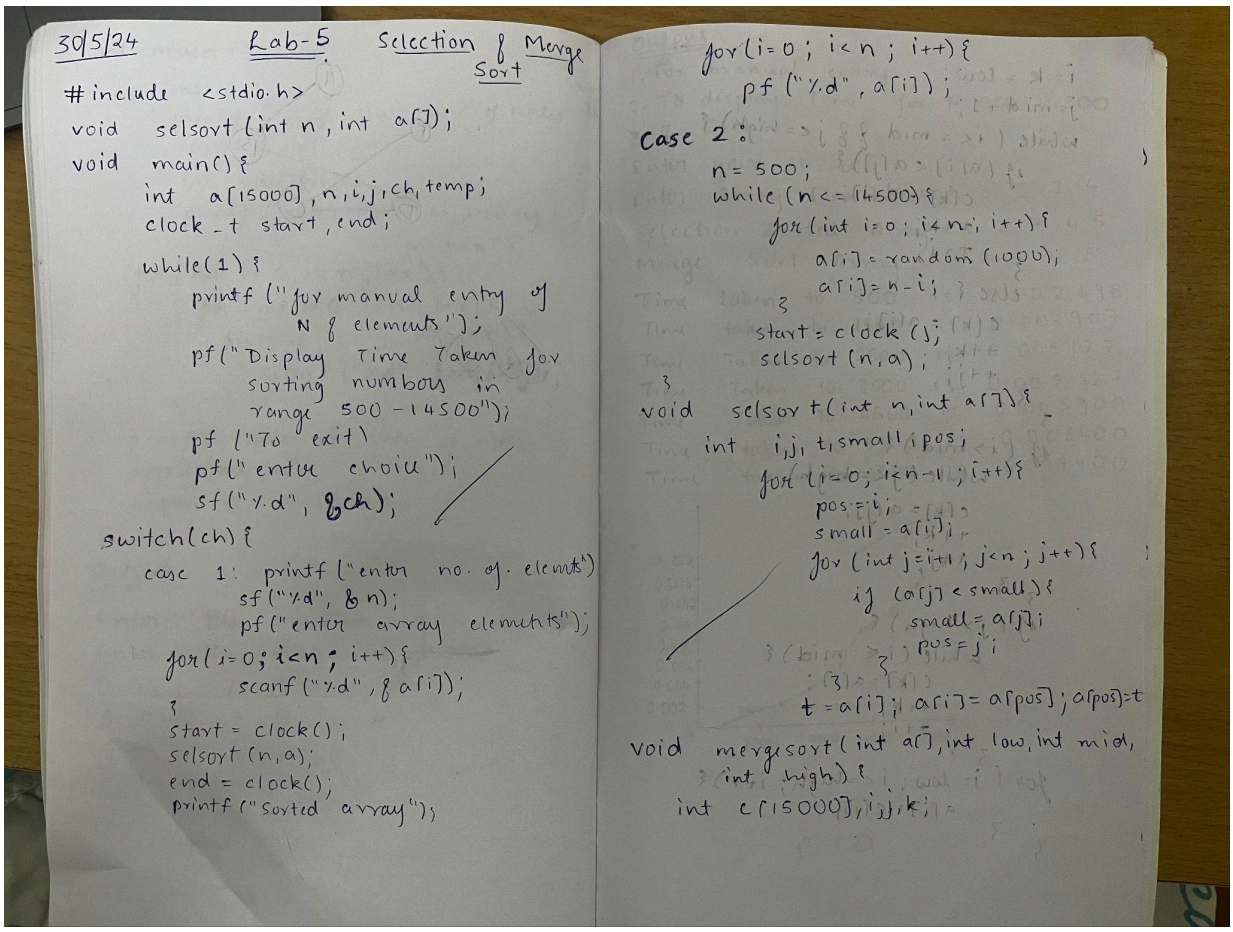
```
//C program to implement merge sort
#include <stdio.h>
#include<time.h>
int a[20],n;
void simple_sort(int [],int,int,int);
void merge_sort(int[],int,int);
int main()
{
int i;
clock_t start, end;
double time_taken;
printf("Enter the no. of elements:");
scanf("%d", &n);
printf("Enter the array elements:");
for (i = 0; i < n; i++) {
scanf("%d", &a[i]);
}
start = clock();
merge_sort(a, 0, n - 1);
end = clock();
time_taken = (double)(end - start) / CLOCKS_PER_SEC;
printf("Sorted array:");
for (i = 0; i < n; i++) {
printf("%d ", a[i]);
}
printf("\n");
```

```
printf("Time taken to sort: %f seconds\n", time_taken);
return 0;
}
void merge_sort(int a[],int low, int high){
if(low<high){
int mid=(low+high)/2;7
merge_sort(a,low,mid);
merge_sort(a,mid+1,high);
simple_sort(a,low,mid,high);
}
}
void simple_sort(int a[],int low, int mid, int high){
int i=low,j=mid+1,k=low;
int c[n];
while(i<=mid && j<=high){
if(a[i]<a[j]){
c[k++]=a[i];
i++;
}else{
c[k++]=a[j];
j++;
}
}
while(i<=mid){
c[k++]=a[i];
i++;
}
while(j<=high){
c[k++]=a[j];
j++;
```

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

OUTPUT:

Enter the no. of elements:10
Enter the array elements:8 96 32 75 62 78 63 48 56 100
Sorted array:8 32 48 56 62 63 75 78 96 100
Time taken to sort: 0.000002 seconds



30/5/24 Lab-5 Selection & Merge Sort

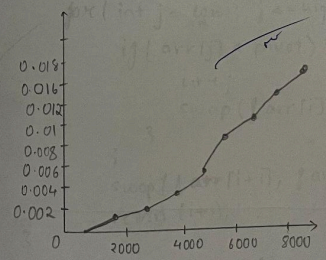
```
#include <stdio.h>
void selsort(int n, int a[]);
void main(){
    int a[15000], n, i, j, ch, temp;
    clock_t start, end;
    while(1){
        printf("for manual entry of N & elements");
        pf("Display Time Taken for sorting numbers in range 500 - 14500");
        pf("To exit");
        pf("enter choice");
        sf("%d", &ch);
switch(ch){
    case 1: printf("enter no. of elements")
            sf("%d", &n);
            pf("enter array elements");
        for(i=0; i<n; i++){
            scanf("%d", &a[i]);
        }
        start = clock();
        selsort(n,a);
        end = clock();
        printf("Sorted array");
        for(i=0; i<n; i++){
            pf("%d", a[i]);
Case 2:
    n = 500;
    while(n<=14500){
        for(int i=0; i<n; i++){
            a[i] = random(1000);
            a[i] = n-i;
        }
        start = clock();
        selsort(n,a);
    }
void selsort(int n, int a[]){
    int i,j, t, small, pos;
    for(i=0; i<n-1; i++){
        pos = i;
        small = a[i];
        for(int j=i+1; j<n; j++){
            if(a[j] < small){
                small = a[j];
                pos = j;
            }
        t = a[i]; a[i] = a[pos]; a[pos] = t
void mergesort(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;
    }
}
if (i > mid) {
    while (j <= high)
        c[k] = a[j];
        ++k;
        ++j;
}
if (j > high) {
    while (i <= mid) {
        c[k] = a[i];
        ++k;
        ++i;
    }
for ( i = low ; i <= high; i++) {
    a[i] = c[i];
}
}
```

Output.

1. For manual entry
2. To display Time for N=500-14500
3. To Exit.

Enter your choice = 1
Enter Array Elements = 1 3 5 2 4
selection Sorted Array = 1 2 3 4 5
Merge Sort Array = 1 2 3 4 5
Time taken to 500 is 0.002698
Time taken to 1500 is 0.002907
Time Taken to 2500 is 0.005127
Time Taken to 3500 is 0.005527
Time Taken to 4500 is 0.005900
Time taken to 5500 is 0.006400
Time taken to 6500 is 0.007400



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

```
//C program to implement quick sort
#include <stdio.h>
#include<time.h>
int a[20],n;
int partition(int [],int, int);
void quick_sort(int [],int,int);
void swap(int*,int*);
int main()
{
int i;
clock_t start, end;
double time_taken;
printf("Enter the no. of elements:");
scanf("%d", &n);
printf("Enter the array elements:");
for (i = 0; i < n; i++) {
scanf("%d", &a[i]);
}
start = clock();
quick_sort(a, 0, n - 1);
end = clock();
time_taken = (double)(end - start) / CLOCKS_PER_SEC;
printf("Sorted array:");
for (i = 0; i < n; i++) {
printf("%d ", a[i]);
}
printf("\n");
```

```
printf("Time taken to sort: %f seconds\n", time_taken);
return 0;
}
void swap(int *a,int *b){
int temp=*a;
*a=*b;9
*b=temp;
}
void quick_sort(int a[],int low,int high){
if(low<high){
int mid=partition(a,low,high);
quick_sort(a,low,mid-1);
quick_sort(a,mid+1,high);
}
}
int partition(int a[],int low,int high){
int pivot=a[low];
int i=low;
int j=high+1;
while(i<=j){
do{
i=i+1;
}while(a[i]<pivot && i<=high);
do{
j=j-1;
}while(a[j]>pivot && j>=low);
if(i<j){
swap(&a[i],&a[j]);
}
}
```

```
swap(&a[j],&a[low]);
return j;
}
```

OUTPUT:

```
Enter the no. of elements:10
Enter the array elements:96 53 26 78 12 63 85 12 06 95
Sorted array:6 12 12 26 53 63 78 85 95 96
Time taken to sort: 0.000002 seconds
```

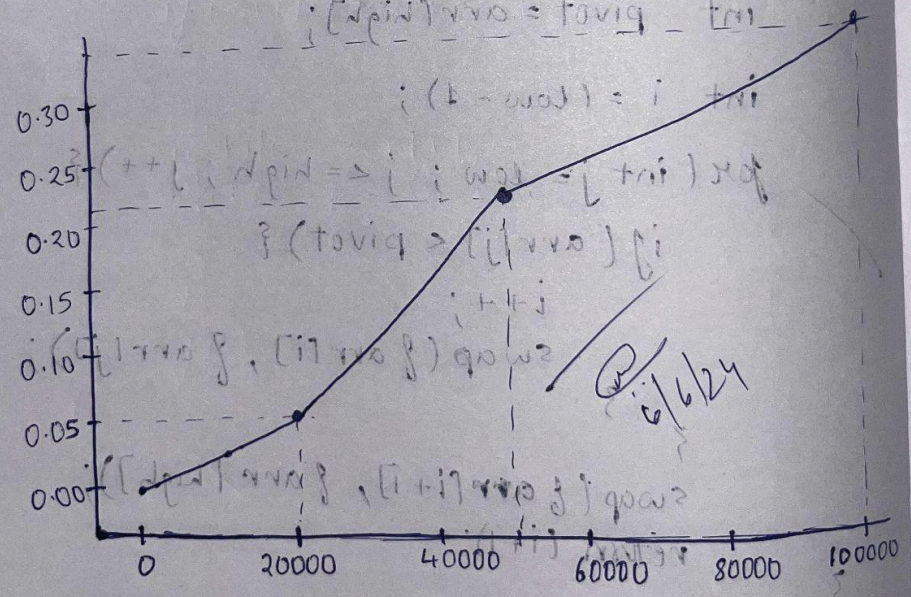
6/6/24

LAB - 6

Quick Sort

```
void swap(int* p1, int* p2) {
    int temp;
    temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i+1], &arr[high]);
    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() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    quicksort(arr, 0, n-1);
    printf("sorted Array \n");
    for (int i=0; i<n; i++) {
        printf("%d", arr[i]);
    }
    return 0;
```



6/6/24

Output:

Sorted Array
1 5 7 8 9 10

13/6/24.

QUESTION: Johnson Trotter

```
#include <stdio.h>
#include <stdlib.h>
void swap(int* a, int* b) {
int temp = *a;
*a = *b;
*b = temp;
}
void generatePermutations(int arr[], int start, int end) {
if (start == end) {
for (int i = 0; i <= end; i++) {
printf("%d ", arr[i]);
}
printf("\n");
} else {
for (int i = start; i <= end; i++) {
swap(&arr[start], &arr[i]);
generatePermutations(arr, start + 1, end);
swap(&arr[start], &arr[i]); // backtrack
}
}
}
int main() {
int n;
printf("Enter the number of elements: ");
scanf("%d", &n);
int* arr = (int*)malloc(n * sizeof(int));
printf("Enter the elements: ");
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
```

```
}
generatePermutations(arr, 0, n - 1);
free(arr);
return 0;
}
```

OUTPUT:

```
Enter the number of elements: 4
Enter the elements: 1 2 3 4
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 3 2
1 4 2 3
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 3 1
2 4 1 3
3 2 1 4
3 2 4 1
3 1 2 4
3 1 4 2
3 4 1 2
3 4 2 1
4 2 3 1
4 2 1 3
4 3 2 1
```

13/6/24

Lab-7

Johnson Trotter Algorithm

```
# include <stdio.h>
# include <stdio.h>
int flag = 0;
int 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){
            int return g+1;
        }
    else {
        flag++;
    }
    return -1
}
int find_Mobile(int arr[], int d[], int num)
{
    int imobile = 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
            && d[arr[i]-1] == 1 && i != 0)
        {
            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
                && d[arr[i]-1] == 1)
            {
                mobile = arr[i];
                mobile_p = mobile;
            }
            else {
                flag++
            }
        }
        else {
            flag++;
        }
        if ((mobile_p == 0) && (mobile != p))
            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 (int 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;
}
```

Output

```
Enter the number : 3 1 5
Total permutation = 6
All position permutations are :
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
```

QUESTION:

- Sort a given set of N integer elements using Heap Sort technique and compute its time taken AND
- Implement All Pair Shortest paths problem using Floyd's algorithm.

1)

```
//C program to implement merge sort
#include <stdio.h>
#include<time.h>
int a[20],n;
void simple_sort(int [],int,int,int);
void merge_sort(int[],int,int);
int main()
{
int i;
clock_t start, end;
double time_taken;
printf("Enter the no. of elements:");
scanf("%d", &n);
printf("Enter the array elements:");
for (i = 0; i < n; i++) {
scanf("%d", &a[i]);
}
start = clock();
merge_sort(a, 0, n - 1);
end = clock();
time_taken = (double)(end - start) / CLOCKS_PER_SEC;
printf("Sorted array:");
for (i = 0; i < n; i++) {
```

```
printf("%d ", a[i]);
}
printf("\n");
printf("Time taken to sort: %f seconds\n", time_taken);
return 0;
}
void merge_sort(int a[],int low, int high){
if(low<high){
int mid=(low+high)/2;7
merge_sort(a,low,mid);
merge_sort(a,mid+1,high);
simple_sort(a,low,mid,high);
}
}
void simple_sort(int a[],int low, int mid, int high){
int i=low,j=mid+1,k=low;
int c[n];
while(i<=mid && j<=high){
if(a[i]<a[j]){
c[k++]=a[i];
i++;
}else{
c[k++]=a[j];
j++;
}
}
while(i<=mid){
c[k++]=a[i];
i++;
}
```

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

OUTPUT:

```
Enter the no. of elements:10
Enter the array elements:8 96 32 75 62 78 63 48 56 100
Sorted array:8 32 48 56 62 63 75 78 96 100
Time taken to sort: 0.000002 seconds
```

20/6/24

Lab - 8

Heap Sort

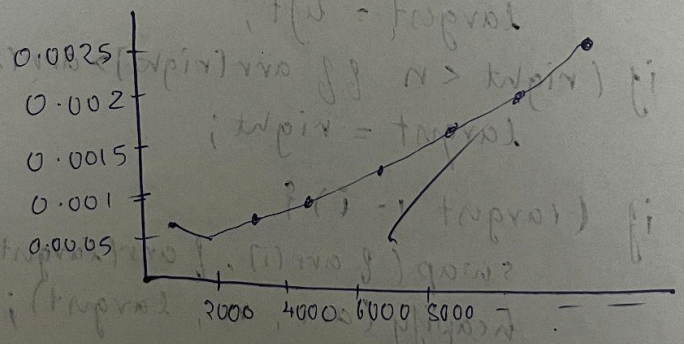
```
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;
    if (left < n && arr[left] > arr[largest])
        largest = left;
    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest != i) {
        swap(&arr[i], &arr[largest]);
        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--) {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
```

```
int main(){
    int arr[] = {1,12,9,5,6,10};
    int n = sizeof(arr)/sizeof(arr[0]);
    heapsort(arr,n);
    printf("sorted array is \n");
    printf(arr,n);
}
```

20/6/24

Output

sorted array is
{1, 5, 6, 9, 10, 12}



2)

```
//C program to implement floyd's algorithm
#include <stdio.h>
int a[10][10],D[10][10],n;
void floyd(int [][10],int);
int min(int,int);
int main()
{
printf("Enter the no. of vertices:");
scanf("%d",&n);
printf("Enter the cost adjacency matrix:\n");
int i,j;
for(i=0;i<n;i++){
for(j=0;j<n;j++){
scanf("%d",&a[i][j]);
}
}
floyd(a,n);
printf("Distance Matrix:\n");
for(i=0;i<n;i++){
for(j=0;j<n;j++){
printf("%d ",D[i][j]);
}
printf("\n");
}
return 0;
}
void floyd(int a[][10],int n){
int i,j,k;
for(i=0;i<n;i++){
```

```
for(j=0;j<n;j++){
D[i][j]=a[i][j];
}
}
for(k=0;k<n;k++){
for(i=0;i<n;i++){
for(j=0;j<n;j++){
D[i][j]=min(D[i][j],(D[i][k]+D[k][j]));
}
}
}14
}
int min(int a,int b){
if(a<b){
return a;
}else{
return b;
}
```

OUTPUT:

```
Enter the no. of vertices:4
Enter the cost adjacency matrix:
0 99 3 99
2 0 99 99
99 6 0 1
7 99 99 0
Distance Matrix:
0 9 3 4
2 0 5 6
```

20/6/24

Lab 8

All pair shortest Path

```
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 pSoll(int dist[][V]){
    pf("The foll. mat. shows - ");
}
int main(){
    int graph[V][V] = {{0, 5, INF, 10},
                       {INF, 0, 3, INF},
                       {INF, INF, 0, 1},
                       {INF, INF, INF, 0}
                      };
    floydWarshall(graph);
    return 0;
}
```

Output -

The following matrix shows the shortest distance

0	5	8	9
INF	0	3	4
INF	INF	0	1
INF	INF	INF	0

20/6/24

QUESTION:

- Implement 0/1 Knapsack problem using dynamic programming.
- Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

1)

```
//C program to implement knapsack problem in dynamic programming
#include <stdio.h>
int n,m,w[10],p[10],v[10][10];
void knapsack(int,int,int[],int[]);
int max(int,int);
int main()
{
int i,j;
printf("Enter the no. of items:");
scanf("%d",&n);
printf("Enter the capacity of knapsack:");
scanf("%d",&m);
printf("Enter weights:");
for(i=0;i<n;i++){
scanf("%d",&w[i]);
}
printf("Enter profits:");
for(i=0;i<n;i++){
scanf("%d",&p[i]);
}
knapsack(n,m,w,p);
printf("Optimal Solution:\n");
for(i=0;i<n;i++){
```

```
for(j=0;j<n;j++){
printf("%d ",v[i][j]);
}
printf("\n");
}
return 0;
}
void knapsack(int n, int m, int w[],int p[]){
int i,j;
for(i=0;i<n;i++){
for(j=0;j<m;j++){
if(i==0 || j==0){
v[i][j]=0;
}else if(w[i]>j){
v[i][j]=v[i-1][j];
}else{
v[i][j]=max(v[i-1][j],((v[i-1][j-w[i]])+p[i]));
}12
}
}
}
int max(int a,int b){
if(a>b){
return a;
}else{
return b;
}
}
```

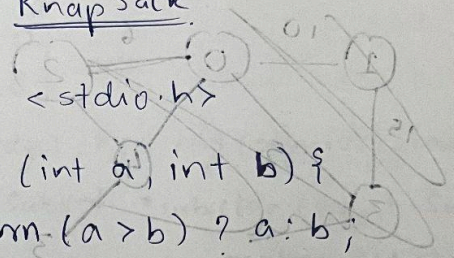
OUTPUT:

```
Enter the no. of items:4
```

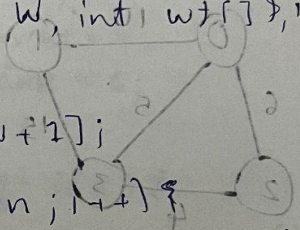
Enter the capacity of knapsack:5
Enter weights:2 1 3 2
Enter profits:12 10 20 15
Optimal Solution:
0 0 0 0
0 10 10 10
0 10 10 20
0 10 15 25

Lab - 9

Knapsack



```
# include <stdio.h>
int max (int a, int b) {
    return (a > b) ? a : b;
}
void knapsack (int W, int wt[], int val[]
                , int n){
    int i, w;
    int K[n+1][W+1];
    for (i=0 ; i<=n ; i++) {
        for (w=0; w<=W ; w++) {
            if (i==0 || w==0) {
                K[i][w] = 0;
    else if (wt[i-1] <= w)
                K[i][w] = max (val[i-1] +
                                K[i-1][w
    - wt[i-1],
    K[i-1][w]);
    else {
        K[i][w] = K[i-1][w]; } }
    pf ("DP table");
    for (i=0 ; i<=n ; i++) {
        for (w=0; w<=W ; w++) {
            pf ("%d", K[i][w]);
```



```
int res = K[n][W];
pf("Maximum profit : %d \n", res);
w = W;
pf("Selected items :");
for (i = n; i > 0 && res > 0; i--) {
    if (res != K[i-1][w]) {
        pf("%d", i);
        res = val[i-1];
        w -= wt[i-1];
    }
}
pf("\n");
int main(){
```

Output

DP 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	6

Max Profit : 7

Selected Items : 2 1

2)

```
/C program to implement prim's algorithm
#include <stdio.h>
int cost[10][10], n, t[10][2], sum;
void prims(int cost[10][10], int n);
int main() {
int i, j;
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]);
}
}
prims(cost, n);
printf("Edges of the minimal spanning tree:\n");
for (i = 0; i < n - 1; i++) {
printf("(%d, %d) ", t[i][0], t[i][1]);
}
printf("\nSum of minimal spanning tree: %d\n", sum);
return 0;
}
void prims(int cost[10][10], int n) {
int i, j, u, v;
int min, source;
int p[10], d[10], s[10];
min = 999;
source = 0;
// Initialize arrays
```

```
for (i = 0; i < n; i++) {16
d[i] = cost[source][i];
s[i] = 0;
p[i] = source;
}
s[source] = 1;
sum = 0;
int k = 0;
// Find MST
for (i = 0; i < n - 1; i++) {
min = 999;
u = -1;
// Find the vertex with minimum distance to the MST
for (j = 0; j < n; j++) {
if (s[j] == 0 && d[j] < min) {
min = d[j];
u = j;
}
}
if (u != -1) {
// Add edge to MST
t[k][0] = u;
t[k][1] = p[u];
k++;
sum += cost[u][p[u]];
s[u] = 1;
// Update distances
for (v = 0; v < n; v++) {
if (s[v] == 0 && cost[u][v] < d[v]) {
d[v] = cost[u][v];
```

```
p[v] = u;
}
}
}
}
}
```

OUTPUT:

```
Enter the number of vertices: 4
Enter the cost adjacency matrix:17
0 1 5 2
1 0 99 99
5 99 0 3
2 99 3 0
Edges of the minimal spanning tree:
(1, 0) (3, 0) (2, 3)
Sum of minimal spanning tree: 6
```

PRIMS

```
#include <stdio.h>

int minKey(int key[], bool mstSet[]);
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++){
        if (mstSet[v] == false && key[v] <
            min = key[v], v
            min_index = v;
    return min_index;
}
void printMST(int parent[], int n
                ,int graph[V][V]
{
    pf("Edge \t Weight\n");
}
for (int i = 1; i < V; i++) {
        printf("%d - %d \t %d \n",
            parent[i], i, graph[i][parent[i]]);
void primMST(int graph[V][V]) {
        int parent[V];
        int Key[V];
        int mstSet[V];
        for (int i=0; i< V; i++){
            Key[i] = INT_MAX;
            mstSet[i] = false;
```

```
        key[0] = 0;
        parent[0] = -1;
    for (int count = 0; count < V-1, count++)
    {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
        for (int v = 0; v < V; v++){
            if (graph[u][v] && mstSet[v]== false
                && graph[u][v] < key[v])
    parent[v] = u, key[v] = graph[u][v]; }
```

~~int~~

=> for

0	2	0	6	0
2	0	3	8	5
0	3	0	0	7
6	8	0	0	9
0	5	7	9	0

expected output

Edge	weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

QUESTION:

- From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.
- Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

1)

```
// C program to implement Dijkstra's algorithm
#include <stdio.h>
int cost[10][10], n, result[10][2], weight[10];
void dijkstras(int [][10], int );
int main()
{
int i, j, s;
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]);
}
}
printf("Enter the source vertex: ");
scanf("%d", &s);
dijkstras(cost, s);
printf("Path:\n");
for (i = 1; i < n; i++) {
printf("(%d, %d) with weight %d ", result[i][0], result[i][1],
weight[result[i][1]]);
}
```

```
return 0;
}
void dijkstras(int cost[][10], int s){
int d[10], p[10], visited[10];
int i, j, min, u, v, k;
for(i = 0; i < 10; i++){
d[i] = 999;
visited[i] = 0;
p[i] = s;
}
d[s] = 0;
visited[s] = 1;
for(i = 0; i < n; i++){23
min = 999;
u = 0;
for(j = 0; j < n; j++){
if(visited[j] == 0){
if(d[j] < min){
min = d[j];
u = j;
}
}
}
visited[u] = 1;
for(v = 0; v < n; v++){
if(visited[v] == 0 && (d[u] + cost[u][v] < d[v])){
d[v] = d[u] + cost[u][v];
p[v] = u;
}
}
```

```
}
for(i = 0; i < n; i++){
result[i][0] = p[i];
result[i][1] = i;
weight[i] = d[i];
}
}
```

OUTPUT:

```
Enter the number of vertices: 4
Enter the cost adjacency matrix:
0 1 5 2
1 0 99 99
5 99 0 3
2 99 3 0
Enter the source vertex: 0
Path:
(0, 1) with weight 1 (0, 2) with weight 5 (0, 3) with weight 2
```

Lab - 10

Dijkstra's Algorithm

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

    int j, count, u, min;

    for (j = 0; j <= n; j++) {
        dist[j] = c[src][j];
        vis[j] = 0;
    }

    dist[src] = 0;
    vis[src] = 1;
    count = 1;

    while (count != n) {
        min = INF;

        for (j = 0; j <= n; j++) {
            if (dist[j] < min && vis[j] != 1) {
                min = dist[j];
                u = j;
            }
        }

        vis[u] = 1;
        count++;

        for (j = 0; j < n; j++) {
            if (min + c[u][j] < dist[j] && vis[j]
                != 1)
            {
                dist[j] = min + c[u][j];
            }
        }
```

```
    printf("Shortest dist" from source %d:\n", src);
    for(j=0 ; j<n ; j++) {
        printf("Source %d to %d", src, j, dist[j]);
    }
}
int main(){
    int n = 5;
    int src = 0;
    int c[5][5] = {
        {0, 10, 5, INF, INF},
        {10, 0, 3, 1, INF},
        {5, 3, 0, 2, 7},
        {INF, 1, 2, 0, 6},
        {INF, INF, 7, 6, 0}
    };
    int dist[n], vis[n];
    dijkstra(n, src, c, dist, vis);
    return 0;
}
```

Output

Shortest distance from source 0:

Source 0 to 0 : 0
0 to 1 : 8
0 to 2 : 5
0 to 3 : 7
0 to 4 : 12

2)

```
//C program to implement Kruskal’s algorithm
#include <stdio.h>
int cost[10][10], n, t[10][2], sum;
void kruskal(int cost[10][10], int n);
int find(int parent[10], int i);
int main() {
int i, j;
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]);
}
}
kruskal(cost, n);
printf("Edges of the minimal spanning tree:\n");
for (i = 0; i < n - 1; i++) {
printf("(%d, %d) ", t[i][0], t[i][1]);
}
printf("\nSum of minimal spanning tree: %d\n", sum);
return 0;
}
void kruskal(int cost[10][10], int n) {18
int min, u, v, count, k;
int parent[10];
k = 0;
sum = 0;
// Initialize parent array for Union-Find
```

```
for (int i = 0; i < n; i++) {
parent[i] = i;
}
count = 0;
while (count < n - 1) {
min = 999;
u = -1;
v = -1;
// Find the minimum edge
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
if (find(parent, i) != find(parent, j) && cost[i][j] < min) {
min = cost[i][j];
u = i;
v = j;
}
}
}
// Perform Union operation
int root_u = find(parent, u);
int root_v = find(parent, v);
if (root_u != root_v) {
parent[root_u] = root_v;
t[k][0] = u;
t[k][1] = v;
sum += min;
k++;
count++;
}
}
```

```
}
int find(int parent[10], int i) {19
while (parent[i] != i) {
i = parent[i];
}
return i;
}
```

OUTPUT:

```
Enter the number of vertices: 4
Enter the cost adjacency matrix:
0 1 5 2
1 0 99 99
5 99 0 3
2 99 3 0
Edges of the minimal spanning tree:
(1, 0) (3, 0) (2, 3)
Sum of minimal spanning tree: 6
```

Kruskals Algorithm

```
struct Edge {
    int src, dest, weight;
};
struct subset {
    int parent;
    int rank;
};
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 unionsets(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;
}
```

```
void kruskalMST(int V, int E, struct Edge edgus[])
    struct Edge result[V];
    int e = 0;
    int i = 0;
    qsort(edgus, E, sizeof(edgus[0]), compare);
    struct subset *subsets = (struct subset*)
                malloc(V * sizeof(struct subset));
    for (int v = 0; v < V; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    while(e < V-1 && i < E) {
        struct Edge next_edge = edgus[i++];
        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);
        if(x != y) {
            result[e++] = next_edge;
            unionSets(subsets, x, y);
    }
```

Output

Edgs in MST:

2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10