DATE: / /
PAGE:

ADA - observation book

i) a. Linear search.

```
#include <stdio.h>
int LinearSearch(int arr[], int n, int x)
{
    for (int i=0; i<n; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}

int main() {
    int arr[] = {2,4,6,8,10,12,14};
    int n = sizeof(arr)/sizeof(arr[0]);
    int x = 10;
    int result = linearsearch(arr, n, x);

    if (result == -1) {
        printf("Element not found\n");
    }
    else {
        printf("Elements found at index %d\n", result);
    }
    return 0;
}
```

time taken for execution : 0.000025

ii) a Binary search

```
#include <stdio.h>
int binarySearch(int arr[], int low, int high, int x)
```

```
while (low <= high) {
    int mid = low + (high - low) / 2;
    if (arr[mid] == x)
        return mid;
    if (arr[mid] < x)
        low = mid + 1;
    else
        high = mid - 1;
}
return -1;

int main() {
    int arr[] = {2, 4, 6, 8, 10, 12, 14};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 10;
    int result = binarySearch(arr, 0, n-1, x);

    if (result == -1) {
        printf("Element not found \n");
    }
    else {
        printf("Element found at index %d \n", result);
    }
    return 0;
}
```

64 34 25 12 22 11 90

34 64 25 12 22 11 90

~~34 64 25~~

34 25 64 12 22 11 90

time taken for execution : 0.000002 s.

Q) iii) bubble sort.

PG.0

```
#include <stdio.h>

void bubbleSort(int arr[], int n) {
    for (int i=0; i<n-1; i++) {
        for (int j=0; j<n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Array befor

    bubbleSort(arr, n);
    printf("Array after sorting:");
    for (int i=0; i<n; i++) {
        printf("%d", arr[i]);
    }
    printf("\n")
    return 0;
}
```

time taken for execution: 0.000315

iv) Selection Sort

```
#include <stdio.h>

void selectionSort(int arr[], int n) {
    int i, j, min_idx;

    for (i=0; i<n-1; i++) {
        min_idx = i;

        for (j=i+1; j<n; j++)
        {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
    }
    int temp = arr[min_idx];
    arr[min_idx] = arr[i];
    arr[i] = temp;
  }
}
int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = 5;
    selectionSort(arr, n);
  printf("Array after sorting:");
  for (int i=0; i<n; i++)
    {
        printf("%d", arr[i]);
    }
  return 0;
```

time of execution: 0.000001s

Output

i) Element

ii) Elemen

iii) Array

iv) Array

DATE: / /
PAGE:

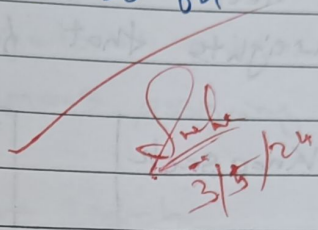
Outputs

i) Element found at index 4

ii) Element found at index 4

iii) Array after sorting : 11 12 22 25 34 64 90

iv) Array after sorting: 11 12 22 25 64



Lab Program

a) topological sort ~~BFS~~ DFS

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

int res[20];
int s[20];
int j = 0;

void dfs(int u, int n, int cost[20][20])
{
    int v;
    s[u] = 1
    for (v=0; v<n; v++)
    {
        if (cost[u][v] == 1 && s[v] == 0)
        {
            dfs(v, n, cost);
        }
    }
    res[j++] = u;
}

void dfs_TP(int n, int a[20][20])
{
    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 main()
{
    int i,j,k,m,n,cost[20][20];
    printf("Enter the number of nodes");
    scanf("%d",&n);

    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            cost[i][j]=0;
        }
    }
    printf("Enter the number of edges \n");
    scanf("%d",&m);
    for(k=0; k<m; k++)
    {
        printf("Enter the edge i,j \n");
        scanf("%d %d",&i,&j);
        cost[i][j]=1;
    }
    dfs_TP(n,cost);
    printf("The topological sequence");
    for(i=n-1; i>=0; i--)
        printf("%d",res[i]);
    printf("\n");
}
```

Output :- Enter the number of nodes = 4
Enter the number of edges = 4
Enter the edge i, j
0
1
Enter the edge i, j
0
3
Enter the edge i, j
1
2
Enter the edge i, j
3
2

The topological sequence is 0 3 1 2

ii) ~~Topological sort using source Removal method~~.

GCD of two numbers using iteration

Enter the two numbers 4 10
GCD of two number is 2

iii) GCD of two numbers using recurssion

Enter the two numbers 4 10
GCD of two numbers is 2

iv) Selection sort

Enter the number of elements
4
Enter the elements

DATE: / /
PAGE:

2
10
7

The sorted array,
1 2 7 10.

v) topological sort using Source Removal method.

Enter the number of elements
4
Enter the elements
1
2
10
7

The sorted array,
1 2 7 10.

$\Rightarrow \frac{n(n+1)}{2} \Rightarrow O(n^2)$

Lab programs

Quick Sort

```
#include <stdio.h>
        n=6;
        mid = n/2;
void swap (int* a, int* b) {

    int temp = *a;
    *a = *b;
    *b = temp;
}
int partition (int arr[], int low, int high) {
        int pivot = arr[high];
        int i = (low - 1);

for (int j = low; j <= high-1; 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);
    }
}

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

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

Output

Given array:
10 7 8 9 1 5

Sorted array:
1 5 7 8 9 10.

Merge Sort

```
#include <stdio.h>

void merge (int arr[] , int l , int m , int r){
    int i,j,k;
    int n1 = m-l+1;
    int n2 = r- m

    for (i=0 ; i<n1 ; i++)
        L[i] = arr [l+i];
    for (i=0 ; j<n2 ; j++)
        R[j] = arr [m+1+j];

    i=0;
    j=0;
    k=l;
while (i<n1 && j<n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i]
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
while (i<n1) {
    arr[k] = L[i];
    i++;
    k++;
}
while (j<n2) {
    arr[k] = R[j];
```

```
}
}
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r-l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
void printArray(int A[], int size) {
    int i;
    for (i=0; i<size; i++)
        printf("%d", A[i]);
    printf("\n");
}
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = 6;
    mergeSort(arr, 0, arr_size-1);
    printf("Sorted array is\n");
    printArray(arr, arr_size);
    return 0;
}
```

Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13.

9/6/24

DATE: PAGE:

Lab- Manual

8 Warshall's Algorithm.

```
#include <stdio.h>

void warshall(int a[][10], int n) {
        int p[10][10];

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

for(int k=0; k<n; k++) {
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            if (p[i][k] == 1 && p[k][j]==1){
                p[i][j] = 1;
            }
        }
    }
}

printf("Transitive closure \n");
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) {
        printf("%d", p[i][j]);
    }
    printf("\n");
}
}
```

DATE: PAGE:

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

    int a[10][10];
    printf ("Enter the adjacency matrix: \n");
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            scanf ("%d", &a[i][j]);
        }
    }
    warshall (a, n);
    return 0;
}
```

Output

```
Enter the number of vertices: 3
Enter the adjacency matrix: 0 0 0 1 1 0
1 1 1
Transitive closure:
0 0 0
1 1 0
1 1 1
```

2 Floyd's Algorithm

```
#include <stdio.h>
#define V 4
#define INF 99999

void printSolution (int dist[][V]);
void floydWarshall (int dist[][V])
{
```

DATE: PAGE:

```
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(
      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, INF, 3, INF},
                        {2, 0, INF, INF},
                        {INF, 7, 0, 1},
                        {7, INF, INF, 0}
```

```
floydWarshall(graph);
    return 0;
}
```

8 Johnson trotter Algorithm

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_N 10

int p[MAX_N];
int d[MAX_N];
int n;

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

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

int mobile(int i) {
        if (d[i] == -1 && i != 0)
            return (p[i] > p[i-1]) ? i : -1;
        if (d[i] == 1 && i != n-1)
            return (p[i] > p[i+1]) ? i : -1;
        return -1;
}
```

DATE: PAGE:

```
void johnson_trotter() {

    for (int i = 0; i < n; i++) {
        d[i] = -1;
        p[i] = i + 1;
    }
    print_perm();
    bool m = true;
    while (m) {
        m = false
        int mi = -1;
        int mn = -1;
        for (int i = 0; i < n; i++) {
            int t = mobile(i);
            if (t != -1 && p[i] > mn) {
                mn = p[i];
                mi = t;
                m = true;
            }
        }
    }
    if (m) {
        int mni;
        if (d[mi] == -1)
            mni = mi - 1;
        else
            mni = mi + 1;
        swap(&p[mni], &p[mi]);
        swap(&d[mni], &d[mi]);

        for (int i = 0; i < n; i++) {
            if (p[i] > mn)
                d[i] = -d[i];
        }
```

DATE: PAGE:

```
    print_perm();
    }
}
}

int main() {
    printf("Enter the value of n (max 10):");
    scanf("%d", &n);

    if (n > MAX_N || n <= 0) {
        printf("Invalid input.\n");
        return 1;
    }
    johnson_trotter();
    return 0;
}
```

Enter the value of n : 2

1 2

2 1

13/6/24

iv)

```
#include <stdio.h>

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

int 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];
        }
    }
}
    return K[n][w];
}
int main() {
    int val[] = {60, 100, 120};
    int wt[] = {10,20,30};
    int W = 50;
    int n= 3
printf("Maximum value in Knapsack = %d \n",
            Knapsack(W, wt, val, n));
    return 0;
}
```

~~Outside~~

Output

Maximum value in Knapsack = 220

8 i) Horspools Algorithm.

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

DATE: PAGE:

```
#define MAX_CHAR 256

void buildShiftTable(char *pattern, int patternLen,
                     int shiftTable[]) {

    for(int i=0; i<MAX_CHAR; i++)
    {
        ShiftTable[i] = patternLen;
    }
    for (int j = 0; j<= patternLen-1; j++) {
        ShiftTable[(unsigned char)pattern[i]]
            = patternLen-1-j;
    }
}
void horspoolSearch(char *text, char *pattern)
{
    int textLen = strlen(text)
    int patternLen= strlen(pattern);

    if (patternLen > textLen) {
        printf("Pattern length is greater than
                text length.\n");
        return;
    }
int ShiftTable[MAX_CHAR];
buildShiftTable(pattern, patternLen, ShiftTable)

int i= patterLen-1;
while(i<textLen) {
    int k=0;
    while(k<patternLen && pattern[patternLen
                                  -1-k]
          == text[i-k]) {
```

DATE: PAGE:

```
for (int i =0; i<N; i++)
    scanf ("%d", &arr[i]);
    heapsort (arr, N);
    printf ("sorted array")
    printf array (arr, N);
    free (arr);
    return 0;
```

Output:

Enter the no. of elements

5

Enter 5 elements = 5

26

48

32

Sorted array is 5 26 32 48

DATE: PAGE:

Lab-6

Q i) Prims Algorithm

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

#define V 5

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) {
            min = key[v];
            min_index = v;
        }
    }
    return min_index;
}

void printMST(int parent[], int graph[V][V])
{
    printf("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])
{
```

DATE: PAGE:

```
int parent[V];
int key[V];
bool 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];
    }
}
}

printMST(parent, graph);
}
int main() {

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

DATE: PAGE:

```
{0, 3, 0, 0, 7},
{6, 8, 0, 0, 9},
{0, 5, 7, 9, 0}
};
printMST(graph);
    return(0);
}
```

Output

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

ii) Dijkstra's Algorithm

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

#define V 10
int minDistance(int dist[], int sptSet[], int n)
{
    int min = INT_MAX, min_index;

    for (int v=0; v<n; v++)
  {
    if (sptSet[v]==0 && dist[v]<=min)
        {
            min = dist[v];
            min_index = v;
        }
    }
return min_index;
}
```

DATE: PAGE:

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

void dijkstra(int graph[v][v], int src, int n)
{
    int dist[v];
    int sptSet[v];
    for(int i=0; i<n; i++){
        dist[i] = INT_MAX;
        sptSet[i] = 0;
    }
    dist[src]=0;
    for(int count=0; count<n-1; count++)
    {
        int u= minDistance(dist, sptSet, n);
        sptSet[u]=1;

        for(int v=0; v<n; v++){

            if(!sptSet[v] && graph[u][v] && dist[u]!=
                INT_MAX && dist[u] + graph[u][v] <
                    dist[v])
        }
    }
    printSolution(dist, n);
}

int main()
{
```

DATE: PAGE:

```
int graph[v][v];
  int n;
  int src;

  Printf(" Enter the number of vertices in the
       graph %od:", v);
   Scanf("%od ", &n);

Printf(" Enter the adjacency matrix representation
      of the graph: \n");
   for (int i=0; i<n; i++)
       {
          for (int j=0; j<n; i++)
            {
              Scanf(" %od", &graph[i][i]);
            }
       }
Printf(" Enter the source vertex %od", n-1);
  Scanf ("%od", &src);

  dijkstra (graph, src, n);
  return 0;
}
```

Output

Enter the number of vertices in the graph : 3
Enter the adjacency matrix representation of the graph

1
2
3
4
5

DATE: PAGE:

6
7
8
9

Enter the source vertex (between 0 and 2):

Vertex	distance from source
0	7
1	8
2	6

5/11/24

iii) 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("\n Sum of minimal spanning tree %d\n", sum);
return 0;
}
void kruskal(int cost[10][10], int n) {

int min, u, v, count, k;
int parent[10];
k=0;
sum=0;

for(int i=0; i<n; i++) {
    parent[i] = i;
}
count=0;
while(count<n-1) {
    min=999;
    u=-1;
    v=-1;
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;
  }
}
}

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++;
 }
}
}
```

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.
```

DATE: PAGE:

iv) Fractional Knapsack

```
#include <stdio.h>
void knapsack(int n, int p[], int w[],
              int W)
{
    int used[n];
    for(int i=0; i<n; ++i)
        used[i] = 0;
    int cur_w = W;
    float tot_v = 0.0;
    int i, maxi;
    while (cur_w > 0) {
        maxi = -1;
        for(i=0; i<n; ++i)
            if ((used[i] == 0) &&
((maxi == -1) || ((float)w[i]/p[i] > (float)w[maxi]
/p[maxi])))
        maxi = i;
        used[maxi] = 1;
        if (w[maxi] <= cur_w) {
            cur_w -= w[maxi];
            tot_v += p[maxi];
printf("Added %od %0% (%od, %0d) of object
    %od in the bag\n", (int)(float)taken
/w[maxi]*100), w[maxi], p[maxi], maxi+1);
        }
    }
printf("Filled the bag with objects
    worth %2f.\n", tot_v);
}
int main() {
```

```
int n, w;
printf("Enter the number of objects:");
  scanf("%od", &n);
  int p[n], w[n];
printf("Enter the profits of the objects:");
  for (int i=0; i<n; i++) {
      scanf("%od", &p[i]);
  }
  printf("Enter the weights of the objects");
  for (int i=0; i<n; i++) {
      scanf("%od", &w[i]);
  }
  printf("Enter the maximum weight of
      the bag:");
  scanf("%od", &w);
  knapsack(n, p, w, W);
    return 0;
}
```

Output

```
Enter the number of objects: 7
Enter the profits of the object:
        5 10 15 7 8 9 4
Enter the weights of the object: 4
        1 3 5 4 1 3 2
Enter the maximum weight of the bag: 15
Added object 4 (4,7) completely in the
 bag. Space left 11.
Added object 7 (2,4) completely in the bag
   Space left 9.
Added object 3 (5,15) completely in the bag
   Space left 4.
Added objects 6 (3,9) completely in the bag
 space left 1.
```

DATE: PAGE:

Added 33% (3,10) of objects 2 in the bag.

Filled the bag with objects worth 36.

Lab - 7

8 Implement "N-Queens Problem" using Backtracking.

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

bool place(int [], int);
void printSolution(int [], int);
void nQueens(int);
int main()
{
    int n;
    printf("Enter the number of queens");
    scanf("%d", &n);
    nQueens(n);
    return 0;
}
void nQueens(int n) {
    int x[10];
    int count = 0;
    int k = 1;
    while (k != 0) {
        x[k] = x[k] + 1;
        while (x[k] <= n && !place(x,k))
{
    x[k] = x[k] + 1;
}
if (x[k] <= n) {
    if (k == n) {
        printSolution(x, n);
        printf("Solution found \n");
```

DATE: PAGE:

```
count++;
}
else {
    k++
    x[k] = 0;
}
} else {
    k--;
}
}
printf("Total solutions %d \n", count);
}
bool place(int x[10], int k) {
    int i;
    for (i=1; i<k; i++) {
if (x[i] == x[k]) || (i-x[i] == k-x[k]) ||
    (i+x[i] == k+x[k]))
        {
            return false;
        }
    }
    return true;
}
void printSolution(int x[10], int n)
{
    int i;
    for (i=1; i<=n; i++) {
        printf("%d", x[i]);
    }
    printf("\n");
}
```

DATE: PAGE:

Output:

```
Enter the number of queens : 4
    2 4 1 3
Solution found
    3 1 4 2
Solution found
Total solutions : 2.
```

18/7/24