INDEX

NAME: Sohan A R STD.: _____ SEC.: _____ ROLL NO.: 1BM22CS285 SUB.: ADA LABS

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
1	3-5-24	Linear, binary search, selection, bubble sort	10	Sohan 7/6/24
2	31-5-24	GCT, TOH, TOP, OFS, SR, CM	10	
3	7/6/24	Merge sort, Quicksort	10	Sohan 13/6/24
4	13/6/24	Johnson Trotter, Warshall, Floyds	10	Sohan 21/6/24
5	21/6/24	Heap sort, Horspool Algorithm	10	Sohan 5/7/24
6	5/7/24	Kruskals, dijkstras, prims Algo	10	Sohan 16/7/24
7	12/7/24	N Queens Algorithm		

classmate
Date
Page

3-5-24

LAB-1

1) Linear Search

```
#include <stdio.h>
int linear search (int arr[], int size, int key) {
    for (int i=0 ; i<size ; i++) {
        if (arr[i] == key) {
            return i;
        }
    }
    return -1;
}
int main () {
    int size ;
    printf ("Enter the size of the array");
    scanf ("%d", &size);
    int arr[size];
    printf ("Enter %d elements of the array: \n", size);
    for (int i=0 ; i<size ; i++)
        scanf ("%d", &arr[i]);
    }
int key ;
printf ("Enter the key ");
scanf ("%d", &key);
int index = linear search (ra, size, key);
if (index != -1) {
    printf ("Element %d found at index %d \n", key, index);
}
else {
    printf ("Element %d not found in the array \n", key);
} return 0; }
```

2) Binary Search

```
#include <stdio.h>
int binarySearch(int arr[], int size, int key);
int main() {
    int size;
    printf("Enter the size of the array");
    scanf("%d", &size);
    int arr[size];
    printf("Enter elements in the array");
    for(int i=0; i<size; i++) {
        scanf("%d", &arr[i]);
    }
    int key;
    printf("Enter the key to be searched");
    scanf("%d", &key);
    int index = binarySearch(arr, size, key);
    if(index != -1) {
        printf("Elements %d found at index %d \n", key, index);
    }
    else {
        printf("Element %d not found in the array \n", key);
    }
    return 0;
}
int binarySearch(int arr[], int size, int key) {
    int low = 0, high = size - 1;
    while(low <= high) {
        int mid = low + (high - low)/2;
```

```
if (arr[mid] == key)
    return mid;
else if (arr[mid] < key)
    low = mid + 1;
else
    high = mid - 1;
}
return -1;
}
```

3) Selection sort

```
#include <stdio.h>
void selection sort (int arr[], int size){
  for(int i=0; i<size-1; i++){
      int min_index = i;
  for(int j=i+1; j<size; j++){
      if(arr[j] < arr[min_index]){
        min_index = j;
      }
      if(min_index != j){
        int temp = arr[i];
        arr[i] = arr[min_index];
        arr[min_index] = temp;
      }
  }
}

int main(){
    int size
    printf("Enter the size of the array");
    scanf("%d", &size);
```

```
int arr[size];
printf("Enter %d elements of the array: \n", size);
for(int i=0; i<size; i++){
    scanf("%d", &arr[i]);
}
selectionSort(arr, size);
printf("Sorted array");
for(int i=0; i<size; i++){
    printf("%d ", arr[i]);
}
printf("\n");
return 0;
}
```

4 Bubble sort

```
#include <stdio.h>
void bubbleSort(int arr[], int size){
    for(int i=0; i<size-1; i++){
        for(int j=0; j<size-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 size;
    printf("Enter the size of the array");
    scanf("%d", &size);
```

```
int arr[size];
printf("Enter %d elements of the array \n", size);
for(int i=0; i<size; i++){
    scanf("%d", &arr[i]);
}
bubble sort(arr, size);
printf("Sorted array \n");

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

LAB - 2 31-5-24

1) GCD of two number

```
#include <stdio.h>
int gcd (int a, int b) {
    if (b == 0)
        return a;
    else
        return gcd (b, a % b);
}
int main () {
    int num1, num2
    printf ("Enter two integers");
    scanf ("%d %d", &num1, &num2);
    printf ("GCD of %d and %d is %d \n", num1,
            num2, gcd (num1, num2));
    return 0;
}
```

Output

```
Enter two integers : 5
7
GCD of 5 and 7 is 1
```

2 Tower of Hanoi

```
#include <stdio.h>
void toh (int n, char source, char dest, char temp) {
    if (n == 1) {
        printf ("move disk from rod %c to rod %c \n",
                source, dest);
        return;
    }
}
```

```
toh (n-1, source, temp, dest);
printf("Move disk %d from rod %c to %c\n", n, source, dest);
toh (n-1, temp, dest, source);
}
int main() {
    int n;
printf("Enter the number of disks");
scanf("%d", &n);
toh(n, 'A', 'C', 'B');
return 0;
}
```

Output:

Enter the number of disks: 2
Move disk 1 from A to B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C

Topological Sorting by DFS

Output:

Enter number of vertices: 3
Enter the adjacency matrix:
1 1 2
1 2 1
1 2 4
Topological sorting: 0 1 2

Topological sort by source removal.

Output:

Enter the number of vertices: 3

Enter the adjacency matrix

1 2 3

4 5 6

7 8 9

Topological sorting: 2 1 0

Quick select: Computing Median

Enter the size of the array

5

Enter the array elements

10 3 4 5 8

Enter the value of k

3

The 3rd smallest element is: 5

LAB-3

i) Merge ~~sort~~ Quick sort)

```
#include <stdio.h>
void merge(int [], int low, int mid, int high, int c[ ]) {
    int i = low;
    int j = mid+1;
    int k = low;

while (i<=mid && j<=high) {
    if (a[i]<a[j])
        c[k++] = a[i++];
    else
        c[k++] = a[j++];
}
while (i<=mid) {
    c[k++] = a[i++];
}
while (j<=high) {
    c[k++] = a[j++];
}
for (i=low; i<=high; i++) {
    a[i] = c[i];
}
}

int partition (int a[], int low, int high) {
    int pivot = a[low];
    int i = low+1;
    int j = high;
```

```
do {
while (a[i] <= pivot) {
    i++;
}
while (a[j] > pivot) {
    j--;
}
if (i < j) {
    swap(&a[i], &a[j]);
}
} while (i < j);
swap(&a[low], &a[j]);
return j;
}

void quick_sort(int a[], int low, int high) {
if (low < high) {
    int mid = partition(a, low, high);
    quick_sort(a, low, mid);
    quick_sort(a, mid+1, high);
}
}

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

```
int main() {
    int a[] = {5, 2, 8, 1, 9, 3, 7, 6, 4};
    int n = sizeof(a) / sizeof(a[0]);
    int c[n];
    printf("Original array: ");
    print_array(a, n);

    merge_sort(a, 0, n-1, c);
    printf("Sorted array using merge sort: ");
    print_array(a, n);

    int b[] = {5, 2, 8, 1, 9, 3, 7, 6, 4};
    quick_sort(b, 0, n-1);
    printf("Sorted array using quick sort: ");
    print_array(b, n);
    return 0;
}
```

Output:

Original array:
5, 2, 8, 1, 9, 3, 7, 6, 4

Sorted array using quick sort
1 2 3 4 5 6 7 8 9

ii) Merge sort:

```
#include<stdio.h>
void merge(int a[], int low, int mid, int high, int c[]){
    int i = low;
    int j = mid+1;
    int k = low;

    while(i<=mid && j<=high){
        if(a[i]<a[j])
            c[k++] = a[i++];
        else
            c[k++] = a[j++];
    }
    while(i<=mid){
        c[k++] = a[i++];
    }
    while(j<=high){
        c[k++] = a[j++];
    }
    for(i=low; i<=high; i++){
        a[i] = c[i];
    }
}

void merge_sort(int a[], int low, int high, int c[]){
    int mid;
    if(low<high){
        mid = (low+high)/2;
        merge_sort(a, low, mid, c);
        merge_sort(a, mid+1, high, c);
```

```
    merge(a, low, mid, high, c);
  }
}
void print_array(int a[], int n) {
    int i;
    for(i=0; i<n; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int a[n];
    int c[n];
    printf("Enter the elements: ");
    for(int i=0; i<n; i++) {
        scanf("%d", &a[i]);
    }
    printf("Before sorting: ");
    print_array(a, n);
    merge_sort(a, 0, n-1, c);
    printf("After sorting: ");
    print_array(a, n);

    return 0;
}
```

Output

Enter the number of elements: 5

Enter the elements: 23

45

1

68

32

Before sorting: 23 45 1 68 32

After Sorting: 1 23 32 45 68

7/6/24

LAB-4

i) Floyd's Algorithm

```
#include <stdio.h>
#define V 4
#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, INF, 3, INF},
                        {2, 0, INF, INF},
                        {INF, 6, 0, 1},
                        {7, INF, INF, 0} };
    floyd warshall (graph);
    return 0;
}
```

Output:

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

0	9	3	4
2	0	5	6
8	6	0	1
7	16	10	0

2) Warshall's algorithm

```
#include <stdio.h>
void warshall (int a[][10], int n) {
    int p[10][10];

    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            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][n]==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");
}
int main(){
    int n;
    printf("Enter the number of vertices:");
    scanf("%d", &n);
    int a[10][10];
    printf("Enter the number of vertices:");
    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: 4
Enter the adjacency matrix:
0
1
0
0
0
0
0
1
0
0
0
0
1
0
1
0
```

Transitive closure:

```
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

3) Johnson Trotter

```
#include <stdio.h>
#include <stdbool.h>
bool LEFT_TO_RIGHT = true;
bool RIGHT_TO_LEFT = false;

void swap(int *a, int *b){
  int temp = *a;
  *a = *b;
  *b = temp;
}
int searchArr(int a[], int n, int mobile)
{
   for(int i=0; i<n; i++)
      if(a[i]==mobile)
        return i+1;
}

int getMobile(int a[], bool dir[], int n)
{
   int mobile_prev=0, mobile=0;
   for(int i=0; i<n; i++){
      if(dir[a[i]-1]==RIGHT_TO_LEFT && i!=0){
        if(a[i]>a[i-1] && a[i]>mobile_prev)
        {
          mobile = a[i];
          mobile_prev = mobile;
        }
      }
   }
}
```

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

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

    if (dir[a[pos-1]-1] == RIGHT_TO_LEFT)
        swap(&a[pos], &a[pos-1]);

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

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

int fact(int n)
{
    int res = 1;
```

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

void printPermutation (int n)
{
    int a[n];
    bool dir[n];
    for (int i = 0; i < n; i++){
        a[i] = i + 1;
        printf ("%d", a[i]);
    }
    printf ("\n");

    for (int i = 0; i < n; i++)
        dir[i] = RIGHT_TO_LEFT;

    for (int i = 1; i < fact(n); i++)
        printOnePerm (a, dir, n);
}

int main()
{
    int n = 4;
    printPermutation (n);
    return 0;
}
```

Output:

1234
124 3
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
1234

13/6/24

LAB-5 (21-6-24)

i) Heap Sort

```
#include <stdio.h>
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) {
        int temp = 0;
        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);
    }
}
```

```
void printarray (int arr[], int N)
{
    for (int i=0; i<N; i++)
        printf ("%d", arr[i]);
    printf ("\n");
}
int main()
{
    int N;
    printf ("Enter number of elements: ");
    scanf ("%d", &N);
    int arr = (int) malloc (N sizeof (int));
    printf ("Enter %d element", N);
    for (int i=0; i<N; i++)
        scanf ("%d", &arr[i]);
    heapSort (arr, N);
    printf ("sorted array")
    print array (arr, N);
    free (arr);
    return 0;
}
```

Output:

Enter the no of elements

5

Enter 5 elements : 5

26

48

32

2) Horspool:

```
#include <stdio.h>
#include <string.h>
#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[j]] =
            patternLen-1-j;
    }
}

void horspool(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 = patternLen-1;
    while(i<textLen){
        int k=0
```

if (i == pattern len) {
printf ("pattern found at position - %d \n",
i - pattern len + 1);

}

i = shift table (unsigned) Text [i]);

}

}

int main () {

char text [] = "this is a simple example";

char pattern [] = "example";

horspool search (text, pattern);

return 0;

}

Output:

Pattern found at 17

21/6/24

Dijkstras 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;
}

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

}

printSolution(dist, n);

int main()
{
    int graph[V][V];
    int n;
    int src;

    printf("Enter the number of vertices in the graph
        (maximum %d): ", V);
    scanf("%d", &n);

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

    printf("Enter the source vertex (between 0 and %d): ",
        n - 1);
    scanf("%d", &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
0 1
1 2
1 5
2 4
2 5
3 6
4 7
5 8
6 9
Enter the source vertex (between 0 and 2 : 2

Vertex	distance from source
0	7
1	8
2	0

Kruskal's algorithm

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

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

struct Graph {
    int V, E;
    struct Edge* edge;
};

struct Subset {
    int parent;
    int rank;
};
struct Graph* createGraph(int V, int E);
void freeGraph(struct Graph* graph);
int find(struct Subset subsets[], int i);
void Union(struct Subset subsets[], int x, int y);
int myComp(const void* a, const void* b);
void KruskalMST(struct Graph* graph);
void printMST(struct Edge result[], int e);

int main() {
    int V, E;
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &V);
    printf("Enter the number of edges in the graph: ");
    scanf("%d", &E);
```

struct Graph *graph = createGraph(V, E);

for (int i = 0; i < E; ++i) {

printf("Enter details for edge %d (source dest weight)

", i+1);

scanf("%d %d %d", &graph->edge[i].src,

&graph->edge[i].dest, &graph->edge[i].

weight);

}

KruskalMST(graph);

freeGraph(graph);

return 0;

}

struct Graph* createGraph(int V, int E) {

struct Graph* graph = (struct Graph*) malloc(sizeof(struct

Graph

graph->V = V;

graph->E = E;

graph->edge = (struct Edge*) malloc(E * sizeof(struct Edge

return graph;

}

void freeGraph(struct Graph* graph)

free(graph->edge);

free(graph);

}

```
void kruskalMST(struct Graph* graph) {
    int V = graph->V;
    struct Edge result[V];
    int e = 0;

    qsort(graph->edge, graph->E, sizeof(graph->edge[0]),
          myComp);

    struct Subset *subsets = (struct Subset*) malloc(V *
                                 sizeof(struct Subset));

    while (e < V - 1 && e < graph->E) {
        struct Edge next_edge = graph->edge[e++];
        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        if (x != y) {
            result[e-1] = next_edge;
            Union(subsets, x, y);
        }
    }

    printMST(result, e);
    free(subsets);
}

void printMST(struct Edge result[], int e) {
    printf("Edges in MST\n");
    for (int i = 0; i < e; ++i)
        printf("%d -- %d == %d\n", result[i].src,
               result[i].dest, result[i].weight);
}
```

Output:

Enter the number of vertices: 2

Enter the number of edges: 2

Enter details of edge 1: 1

22

1

Enter details for edge 2 : 12

21

33

Edge in the MST :

1 -- 22 == 1

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])
{
    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, 9, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 9},
        {6, 8, 0, 0, 4},
        {0, 5, 7, 4, 0}
    };
    primMST(graph);
```

O/p:

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

5/7/24

12-7-24

LAB-7

Q N Queen's algorithm

```
#include <stdio.h>
#include <stdbool.h>
void printSolution(char board[][10], int N);
bool isSafe(char board[][10], int row, int col, int N);
bool solveQueens(char board[][10], int col, int N);

bool solveQueens(char board[][10], int col, int N){
    if(col >= N)
        return true;
    for(int i=0; i<N; i++){
        if(isSafe(board, i, col, N)){
            board[i][col] = 'Q';
            if(solveNQueens(board, col+1, N))
                return true;
            board[i][col] = '-';
        }
    }
    return false;
}

bool isSafe(char board[][10], int row, int col, int N){
    int i, j;
    for(i=0; i<col; i++)
        if(board[row][i] == 'Q')
            return false;
    for(i=row, j=col; i>=0 && j>=0; i--, j--)
        if(board[i][j] == 'Q')
            return false;
```

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

void printSolution(char board[][10], int N){
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++)
            printf("%c ", board[i][j]);
        printf("\n");
    }
}

int main(){
    int N;
    printf("Enter the size of the chess board(N):");
    scanf("%d", &N);
    char board[10][10];
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            board[i][j] = '_';
        }
    }

    if (!solveNQueens(board, 0, N)){
        printf("Solution does not exist\n");
        return 0;
    }

    printf("One possible solution:\n");
    printSolution(board, N);
    return 0;
}
```

Output:

Enter the size of the chess board N: 4

One possible solution:

. . Q .

Q . . .

. . . Q

. Q . .

16/7/24