USN: 1BM22CS328

INDEX

Name: S Pranav Reddy Std: Sec:

Roll No.: Subject: ADA - LAB School/College:

School/College Tel. No.: Parents Tel. No.:

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1	31/5/24	Linear, binary search, selection, bubble sorting	10	3/6/24
2	31/5/24	GCD, TOH, Topo - DFS, SR, EM	10	7/6/24
3	7/6/24	Merge sort, Quick sort		
4	13/06/24	Floyd's, warshall, Schwarfitten	10	13/6/24
5	21/6/24	Leap sort, Hurspool	10	21/6/24
6	05/7/24	Krushkal's, Bijkstra's, Prims	10	5/7/24
7	12/7/24	N Queens Algorithm	10	16/7/24

LAB-1

03-05-24

1) Linear Search

```
#include <stdio.h>
int linearsearch(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 = linearsearch(arr, 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;
}
```

3) 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 search: ");
    Scanf("%d", &key);
    int index = binarySearch (arr, 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;
}
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 selectionSort (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 != i) {
            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 in", size);
    for (int i=0; i<size; i++) {
        scanf("%d", &arr[i]);
    }
    selectionSort(arr, size);
```

2)

```
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]);
    }
    bubbleSort(arr, size);
    printf("sorted array:\n");
```

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

2/5/24

Pgm 2

```
#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;
}
```

o/p

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

2) Towers of Hanoi

```
#include <stdio.h>
void toh(int n, char source, char dest, char temp) {
    if (n == 1) {
        Printf("move disk 1 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;
}
```

O/P: 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

c) Topo sort

Enter the no of vertices 3

Enter the adjacency matrix

1 1 0
1 0 1
0 1 1

Topological sorting using DFS 1 0 2

a) topo-sr

Enter no of vertices 3

Enter the adjacency matrix

1 1 0
1 0 1
0 1 1

Sorting using SL : 1 0 2

f) Computing Median

Enter the size
5
Enter the array elements
10 3 4 5 8
Enter the value of k
3
The 3rd smallest element is : 5

LAB 1

Algo 1

1) Merge sort

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

void merge (int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i=0; i<n1; i++)
        L[i] = arr[l+i];
    for (j=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++;
    }
```

```
    arr[k] = L[i];
    i++;
    k++;
  }

while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;
  }
}

void mergeSort (int arr[], int l, int r)
{ if (l<r) {
      int m = l+(r-1)/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 = sizeof(arr) / sizeof(arr[0]);

    printf("Given array");
    printArray(arr, arr_size);
    mergeSort(arr, 0, arr_size - 1);

    printf("Sorted Array");
    printArray(arr, arr_size);
    return 0;
}
```

O/P: Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13

2) Quick sort:

```
#include <stdio.h>

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 - 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);
    }
}

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

o/p Sorted Array

15 7 8 9 10

7/6/24

LAB - 11

1) Floyd 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} };

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

O/P: The following matrix shows the shortest distances between every pair of vertices

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

2) warshall :

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

```
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][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");
}
int main(){
    int n;
    Printf("Enter the number of vertices:");
    Scanf("%d", &n);
    int a[10][10];
    Printf("Enter the adjacency matrix:\n");
```

```
for (i = ...; i ... ; i++){
    for (j = ...; j ... ; j++){
        scanf("%d", &a[i][j]);
    }
}
warshall(a, n);
return 0;
}
```

```
Enter the number of vertices : 4
Enter the adjacency matrix:
0
1
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
```

5) Sheet-bottles

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#define maxN 20

int p[maxN];
int pi[maxN];
int dir[maxN];

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

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

void printAllPermutations(int n){
    for (int i=0; i<n; i++){
        p[i] = i+1;
        pi[i] = i;
        dir[i] = -1;
    }
    printPermutation(n);
```

```
int main() {
    int n;
    printf("Enter the value of n (max 20): ");
    scanf("%d", &n);
    if (n <= 0 || n > 20) {
        printf("Invalid input. Please enter a value between 1 and 20.\n");
        return 1;
    }
    printPermutations(n);
    return 0;
}
```

```
scanf("%d", &n);
if (n<=0 || n>10)
{
    printf("Invalid input.\n");
    return 1;
}
PrintAllPermutations(n);
return 0;
}
```

o/p:

Enter the value of n : 4

1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2341
2431
4231
4213
2413
2143
2134

13/6/24

```
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);
```

```
        }
    }

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 elements: ", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    heapSort(arr, n);
    printf("Sorted array: ");
    printArray(arr, n);
    free(arr);
    return 0;
}
```

o/p: Enter the no of elements

5

Enter 5 elements: 5

1
2 6
4 8
3 2

sorted array is

1 5 2 6 3 2 4 8

2) Horspool

I/P:

```
#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 horspoolSearch(char *text, char *pattern){
    int textLen = strlen(text);
    int patternLen = strlen(pattern);
    int shiftTable[MAX_CHAR];
```

```
void badChar(char pattern[], int patternLen, int badChar[256]) {
    int i;
    for (i = 0; i < 256; i++)
        badChar[i] = patternLen;
    for (i = 0; i < patternLen - 1; i++)
        badChar[pattern[i]] = patternLen - 1 - i;
}

void boyerMoore(char text[], char pattern[]) {
    int textLen = strlen(text);
    int patternLen = strlen(pattern);
    int badChar[256];
    badChar(pattern, patternLen, badChar);
    int i = patternLen - 1;
    while (i < textLen) {
        int j = patternLen - 1;
        while (j >= 0 && pattern[j] == text[i]) {
            i--;
            j--;
        }
        if (j < 0) {
            printf("Pattern found at position %d\n", i + 1);
            i += patternLen;
        } else {
            i += badChar[text[i]];
        }
    }
}

int main() {
    char text[] = "This is a simple example";
    char pattern[] = "example";
    boyerMoore(text, pattern);
    return 0;
}
```

o/p pattern found at 17

21/6/24

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;
}
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])
            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 adj matrix representation of the graph

    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 & %d): ", n-1);
    scanf("%d", &src);
    dijkstra(graph, src, n);

    return 0;
}
```

o/p Enter the number of vertices in the graph 3

Enter the adjacency matrix representation of the graph

1
2
3
4
5
6
7
8
9

Enter the source vertex (between 0 and 2) : 2

vertex	Distance from source
0	7
1	8
2	0

→ prims

```
I/p: #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];
            }
    }
}
```

```
int graph[V][V] = {
    {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}
};
primMST(graph);
return 0;
}
```

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

- 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 printMST(struct Edge result[], int e);
int main() {
    int V, E;
    printf("Enter the no. of vertices in the graph: ");
    scanf("%d", &V);
    printf("Enter no no. of edges in the graph: ");
    scanf("%d", &E);
    struct Graph* graph = createGraph(V, E);
    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);
}

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);
}
}

int comp(const void* a, const void* b) {
    struct Edge* a1 = (struct Edge*) a;
    struct Edge* b1 = (struct Edge*) b;
}

void KruskalMST(struct Graph* graph) {
    int V = graph->V;
    struct Edge result[V];
    int e = 0;
    for (int v = 0; v < V; ++v) {
        subsets[v].parent = v;
        subset[v].rank = 0
    }
```

```
while (e < V - 1 && e < graph->E) {
    struct Edge next_edge = graph->edge[i++];
    int x = find(subsets, next_edge.src);
    int y = find(subsets, next_edge.dest);
    if (x != y) {
        result[e++] = next_edge;
        Union(subsets, x, y);
    }
}
printMST(result, e);
free(subsets);
}
void printMST(struct Edge result[], int e) {
    printf("Edges in the MST:\n");
    for (int i = 0; i < e; ++i)
        printf("%d -- %d == %d\n", result[i].src,
               result[i].dest, result[i].weight);
}
```

o/p:

```
Enter the no. of vertices in the graph: 3
Enter the no of edges in the graph: 3
Enter details for edge 1 of 3
2 1
3 5
Enter details for edge 2 of 3
1 2
3
Enter details for edge 3 of 3
1 2
5 5
2 2
Edges in the MST
0 -- 0 == 1618565607
```

5/7/24

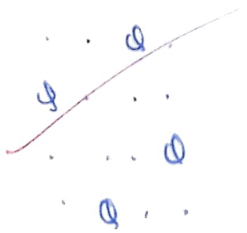
```
nqueens

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

void printSolution(int **board, int N) {
    for (int i=0; i<N; i++) {
        for (int j=0; j<N; j++) {
            printf("%d", board[i][j] ? 'Q' : '.');
        }
        printf("\n");
    }
}

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

```
return true;
}

bool solveNQUtil(int **board, 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]=1;
        if(solveNQUtil(board, col+1, N)){
            return true;
        }
        board[i][col]=0;
    }
}
return false;
}

bool solveNQ(int N){
    int **board=(int **)malloc(N*sizeof(int*));
for (int i=0; i<N; i++){
    board[i]=(int *)Malloc(N*sizeof(int));
    for(int j=0; j<N; j++){
        board[i][j]=0;
    }
}
```

```
if (!solveNQUtil(board, 0, N)) {
        Printf("Solution does not exist");
        return false;
    }

    printSolution(board, N);

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

    free(board);
    return true;
}

int main() {
    int N;
    printf("Enter the value of N: ");
    scanf("%d", &N);
    solveNQ(N);
```

O/P:

Enter the value of N = 4



16/7/24