1. Linear Search

C

#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, 0, 1, 9};

int x = 1;

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

int result linearSearch(arr, n, x);

if (result == -1)

printf("Element not found\n");

else

printf("Element found at index %d\n", result);

return 0;

}

2. Recursive Binary Search

C

#include <stdio.h>

int binarySearch(int arr[], int left, int right, int x) {

if (right >= left) {

int mid left (right left) / 2;

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return binarySearch(arr, left, mid 1, x);

return binarySearch(arr, mid+1, right, x);

}

return -1;

}

int main() {

int arr[] = {2, 3, 4, 10, 40};

int x = 10;

int n sizeof(arr) / sizeof(arr[0]);

int result binarySearch(arr, e, n1, x);

if (result == -1)

printf("Element not found\n");

else

printf("Element found at index %d\n", result);

return 0;

}

3. Naive Pattern Search

C

#include <stdio.h>

#include <string.h>

void naiveSearch(char\* txt, char\* pat) {

int M strlen(pat);

int N strlen(txt);

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

int j;

for (j=0; j < M; j++) {

if (txt[i + j] != pat[j])

break;

}

if (j- M)

printf("Pattern found at index %d\n", i);

}

}

int main() {

char txt[]= "AABAACAADAABAAABAA";

char pat[] "AABA";

naiveSearch(txt, pat);

return 0;

}

4a. Insertion Sort

C

#include <stdio.h>

void insertionSort(int arr[], int n) {

for (int i=1; i < n; i++) {

int key arr[i];

int ji-1;

while (j> && arr[j] > key) {

arr[j+1] arr[j];

j-j-1;

}

arr[j+1] key;

}

}

void printArray(int arr[], int n) {

for (int i=0; i < n; i++)

printf("%d", arr[i]);

printf("\n");

}

int main() {

int arr[] = {12, 11, 13, 5, 6);

int n sizeof(arr) / sizeof(arr[8]);

insertionSort(arr, n);

printArray(arr, n);

return 0;

}

4b. Heap Sort

Copy code

Rinclude <stdio.h>

void swap(int a, int b) {

int ta;

:

\*b=t;

}

void heapify(int arr[], int n, int i) {

int largest- 1;

int left-2\*1+1;

int right2\*1+2;

If (left < n && arr[left] > arr[largest])

largest - left;

if (right < n && arr[right] > arr[largest])

largest right;

heapify(arr, n, largest);

}

If (largest 1-1) {

swap(&arr[i], &arr[largest]);

void heapSort(int arr[], int n) {

for (int in/2-1; 1; 1--)

heapify(arr, n, 1);

for (int in-1;10; i--) (

swap(&arr[], &arr[i]);

heapify(arr, 1, 0);

}

void printArray(int arr[], int n) (

for (int 1; 1 < n; i++)

printf("arr[i]);

printf("\n");

}

int main() {

Int arr[] = {12, 11, 13, 5, 6, 7};

int n sizeof(arr) / sizeof(arr[0]);

heapSort(arr, n);

printArray(arr, n);

return e

}

. Depth First Search

Copy zade

#include <stidis.kr

#include <stdlib.h

struct Node (

Int vertex;

struct Node next;

};

struct Graph (

Int nusVertices;

struct Node\* adjlists;

int visited;

struct Node created(int v) (

struct Node nesiode malloc(sizeof(struct Node));

newNode->vertex vi

newNode->next;

return newllode;

struct Graph" createGraph(int vertices) {

struct Graph graph malloc(sizeof(struct Graph));

graph->nu Vertices vertices;

graph->adjLists malloc(vertices sizeof(struct Node\*));

graph->visited malloc(vertices sizeof(int)); for (int i; i < vertices; i++) (

graph->adjLists[i]-MILL;

graph->visited[i];

return graph;

void additge(struct Graph" graph, int src, int dest) (

struct Node nodiode createNode(dest);

newNode->next graph->ad Lists[src];

graph->ad Lists [src] = newNode;

newflode createllode(src);

newdiode->nextgraph->adjlists[dest];

graph->ad Lists [dest] newNode;

vold DFS(struct Graph graph, int vertex) (

struct Node adjlist graph->adjLists[vertex);

struct Node tomp adjList;

graph->visited[vertex] = 1;

printf("Visited du", vertex);

while (temp INMALL) (

int connectedVertex temp-overtex;

if(graph-visited[connectedVertex]) (

DFS(graph, connectedVertex);

)

}

temp temp->next;

}

int main() {

0

struct Graph graph createGraph();

addEdge(graph,, 1);

addEdge (graph, 0, 2);

addEdge (graph, 1, 2);

addEdge(graph, 2, 3);

DFS(graph,0

return 0

}

7. Dijkstra's Algorithm

#include <stdio.ho

#include <limits.to

#define V9

Int indistance(int dist[], int sptSet[]) (

int min INT MAX, min Index; for (int vvV; ++)

If (sptSet[v] && dist[v] <- min) min dist[v], min\_Index = v;

return min index;

void printSolution(int dist[], int n) (

printf("Vertex itit Distance from Source\n"); for (int 101 < V; 1++)

printf("\t\t", i, dist[1]);

:

Copy code

}

void printSolution(int dist[], int n) {

printf("Vertas Atit Distance from Source'");

for (int 1; 1 < V; I++)

printf("\t\t d", 1, dist[1]);

}

vold dijkstra(int graph [V][V], int src) {

Int dist[V];

Int sptSet[V];

for (int 16;1<V; I++)

dist[1] INT MAX, sptSet[1]=;

dist[src]=;

for (int count; count V 1; count++) {

Int u minDistance(dist, sptSet);

sptSet[u] = 1;

dist[v] dist[u] + graph[u][v];

printSolution(dist, V);

int main() (

int graph[V][V],,,,,,,,),

(0, 0, 0, 31, 0), (0, 0, 4, 14, 10, 0, 2, 0, 0), (0, 10, 0, 0, 0, 0, 1, 0,7),

,,,,,,, 7, 6}};

ㅈ

dijkstra(graph, 0);

return 61

}

9. Floyd's Algorithm

#include <stdio.h

Copy code

#define V4

#define INF 99999

void printselution(int dist[][V]);

vold Floydiarshall (int graph[][V]) {

Ant dist[V][V], 1, 3, k;

for (10; 1 < V; 1++)

for (j <V; j++)

dist[1][3]graph[1][3]:

for (k; k < V; k++) (

for (11 <V; I++) (

for j <V; j++) (

if (dist[i][k]+ dist[k][j] < dist[1][3])

dist[1][3] dist[i][k] dist[k][j];

printSolution(dist);

vald printSolution (ist dist[][V]) (

printf( "Shortest distances between every pair of vertices");

for (int 1 ->

11cV; 1++) (

for (int j <V; j++) {

if (dist[1][3] INF)

printf(");

else

printf(", dist[1][j]);

}

printf("");

}

Int main() (

Int graph [V][V], S, INF, 10), (INF,,, INF),

(INF, INF,, 1),

(INF, INF, INF, }};

floydwarshall (graph);

return

11. Finding Maximum and Minimum Numbers in an Array

Copy code

}

}

#include <stdio.h>

void findMinMax(int arr[], int n, int min, int max) {

\*min - arr[0];

\*max arr[0];

for (int i = 1; i < n; i++) {

if (arr[i] < \*min)

\*min arr[i];

if (arr[i] > \*max)

\*max = arr[i];

int main() {

int arr[] = {12, 1234, 45, 67, 1};

Int n sizeof(arr) / sizeof(arr[0]);

int min, max;

findMinMax(arr, n, &min, &max);

printf("Minimum element is %d\n", min);

printf("Maximum element is %d\n", max);

return 8;

}

12b. Quick Sort

#include <stdio.h

vold swap(int a, int b) {

ntta

Int partition (int arr[], int low, ist high) {

Int pivot arr[high];

It 1 (low-1);

for (int low; j< high j++) [

1F (arr[j] < pivot) (

1++;

swap(&arr[1], &arr[1]);

swap(&arr[1], &arr[high]);

return (1+1);

velit quicksort(int arr[], int low, int high) {

(low high) (

pi partition(arr, Inw, high);

quickSort(arr, low, pi-1);

quickSort(arr, pi, high);

vale printArray(int arr[], int size) { for (St 1 1 < size; i++) printf(", arr[1]);

printf("");

int main() {

Int arr[] (1995);

int n sizeof(arr) / sizeof(arr[0]);

quicksort(arr, 0, 1);

printArray(arr, n);

return

..................................................I

13. N-Queens Problem

fine

seid priirsalutoost board[][]){

(int)( printf board[1][1]):

arint

Exat board[] [], row, col) (

(171ccal; i++)

(board[row][1]]

return false

(1 row, col; 1)

(board[1][1])

Peturn false

For (1 row, col: j, j)

(board[1][1])

return falseg

Bool board [M] [N], col) (

(col N

Petur true

for (int -1; ++) (

(IsSafe(board, i, col)) (

board[i][col]

(solvelutil(board, col+1))

Peturn true

hoard[1][col]

return False;

bool()(

In board[N][]).

()

(solveNQUtil (board,al) (

());

printf("ne does not exist");

return false;

printSolution(board);

evture trung

1

solve();