Week 9

Heap Sort

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to heapify a subtree rooted at node i

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 = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

heapify(arr, N, largest);

}

}

// Heap Sort function

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--) {

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

int main() {

FILE \*file;

int N;

file = fopen("C:/Users/student/Desktop/Numbers500.txt", "r");

if (file == NULL) {

printf("Error opening file.\n");

return 1;

}

fscanf(file, "%d", &N);

int \*arr = (int \*)malloc(N \* sizeof(int));

if (arr == NULL) {

printf("Memory allocation failed.\n");

fclose(file);

return 1;

}

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

fscanf(file, "%d", &arr[i]);

}

fclose(file);

clock\_t start = clock();

heapSort(arr, N);

clock\_t end = clock();

double time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;

//printf("Sorted array:\n");

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

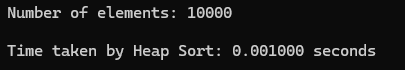
// printf("%d ", arr[i]);

printf("Number of elements: %d\n",N);

printf("\nTime taken by Heap Sort: %f seconds\n", time\_taken);

free(arr);

return 0;

}  


Floyd’s

#include <stdio.h>

#define INF 99999 // A large value to represent infinity

#define V 100 // Maximum number of vertices (adjust as needed)

void floydWarshall(int graph[V][V], int n) {

int dist[V][V];

// Initialize distance matrix same as input graph

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

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

dist[i][j] = graph[i][j];

// Floyd-Warshall algorithm

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

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

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

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

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

// Print the shortest distance matrix

printf("\nAll Pairs Shortest Path Matrix:\n");

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

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

if (dist[i][j] == INF)

printf("%7s", "INF");

else

printf("%7d", dist[i][j]);

}

printf("\n");

}

}

int main() {

int n;

printf("Enter the number of vertices: ");

scanf("%d", &n);

int graph[V][V];

printf("Enter the adjacency matrix (use %d for INF):\n", INF);

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

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

scanf("%d", &graph[i][j]);

floydWarshall(graph, n);

return 0;

}

