**A given set of N integer elements using Heap Sort technique and compute its time taken**.

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

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);

for (int i = N - 1; i >= 0; i--) {

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

void main(){

int a[100000],n,i,j,ch,temp;

clock\_t start,end;

while(1){

printf("\n1:For manual entry of N value and array elements");

printf("\n2:To display time taken for sorting number of elements N in the range 500 to 14500");

printf("\n3:To exit");

printf("\nEnter your choice:");

scanf("%d", &ch);

switch(ch){

case 1:

printf("\nEnter the number of elements: ");

scanf("%d",&n);

printf("\nEnter array elements: ");

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

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

}

start=clock();

heapSort(a,n);

end=clock();

printf("\nSorted array is: ");

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

printf("%d\t",a[i]);

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

break;

case 2:

n=7500;

while(n<=14500) {

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

a[i]=n-i;

}

start=clock();

heapSort(a,n);

for(j=0;j<500000;j++){

temp=38/600;

}

end=clock();

printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS\_PER\_SEC));

n=n+1000;

}

break;

case 3:

exit(0);

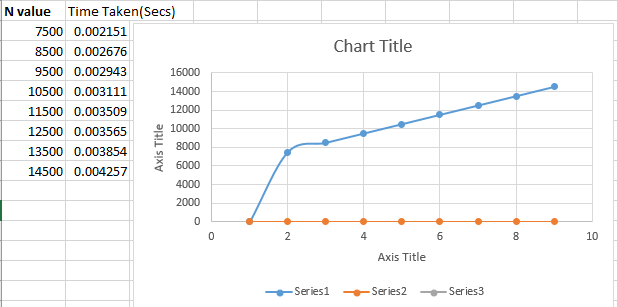
}

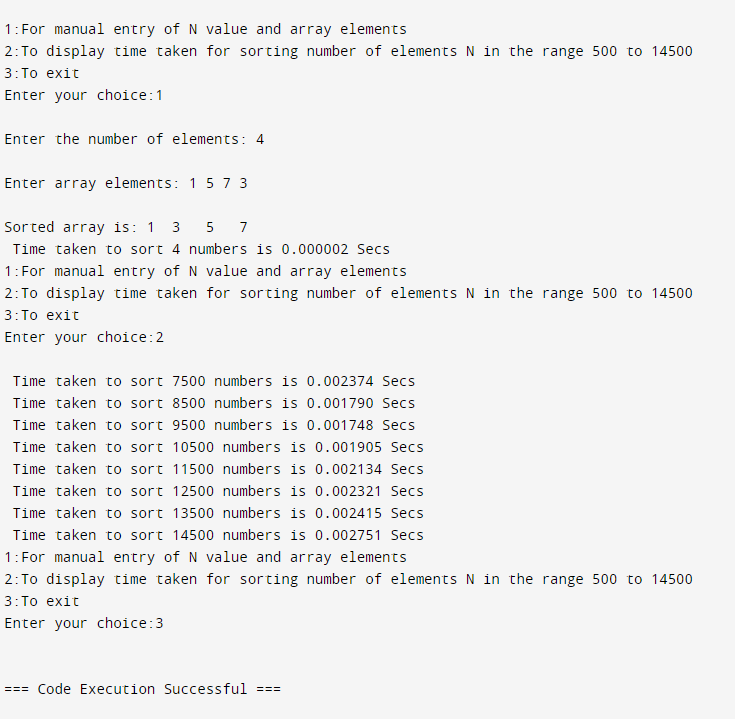
getchar();

}

}

**Output**

****

****

**Implement All Pair Shortest paths problem using Floyd’s algorithm**

#include <stdio.h>

#include <limits.h>

int INF = 1e5;

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

printf("The following matrix shows the shortest distances between every pair of vertices (-1 = infinity):\n");

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

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

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

printf("-1 ");

else

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

}

printf("\n");

}

}

void floydWarshall(int v, int graph[v][v]) {

int dist[v][v], i, j, k;

for (i = 0; i < v; i++)

for (j = 0; j < v; j++)

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

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(v, dist);

}

int main() {

int v;

printf("Enter no. of vertices: ");

scanf("%d", &v);

int graph[v][v];

printf("Enter weighted adjacency matrix (Enter -1 for inf): \n");

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

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

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

if (graph[i][j] == -1) graph[i][j] = INF;

}

}

floydWarshall(v, graph);

return 0;

}

**Output**

