# DAA LAB FILE

###### Submitted By:

###### Purvav Punyani

###### 1805231

###### CS-G1

# LAB-1

# /\*1.1Write a program to store random numbers into an array of n integers and then find out the smallest and largest number stored in it. n is the user input.\*/

#include<stdio.h>

#include<stdlib.h>

int main()

{

int i,n;

printf("Enter a number\n");

scanf("%d",&n);

int a[n];

int max=0,min=32767;

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

{

a[i]=rand();

if(a[i]>max)

max=a[i];

if(a[i]<min)

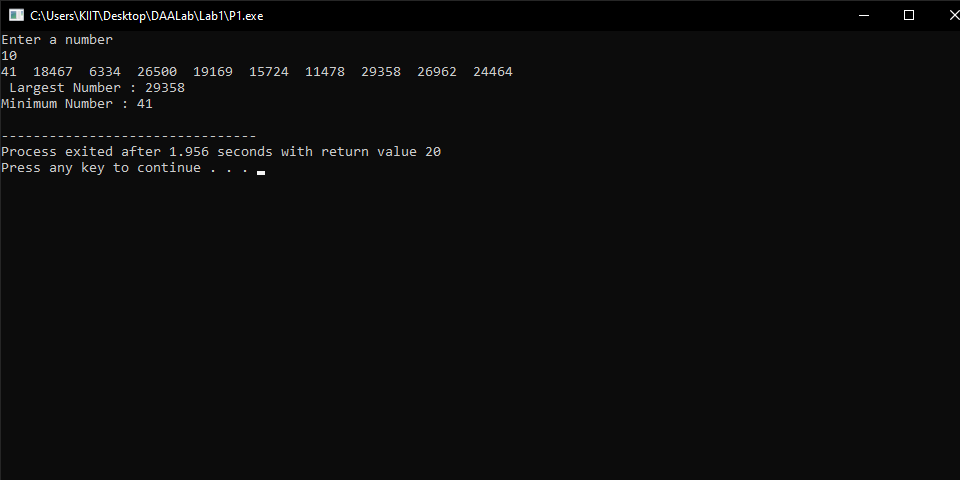
min=a[i];

}

printf("Largest Number : %d\n",max);

printf("Minimum Number : %d\n",min);

}



**/\* 1.2** Write a program to store random numbers into an array of n integers, where the array must cotains some duplicates. Do the following:

a) Find out the total number of duplicate elements.

b) Find out the most repeating element in the array\*/

#include<stdio.h>

#include<stdlib.h>

int main()

{

int i,n;

printf("Enter a number\n");

scanf("%d",&n);

int a[100];

int key[n];

for( i=0;i<100;i++)

a[i]=0;

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

{

int k = rand()%10;

printf("%d ",k);

a[k]++;

key[i]=k;

}

int m=0,c=0,k;

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

{

if(a[key[i]]>1)

{

c++;

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

}

if(a[key[i]]>m)

{

m=a[i];

k=i;

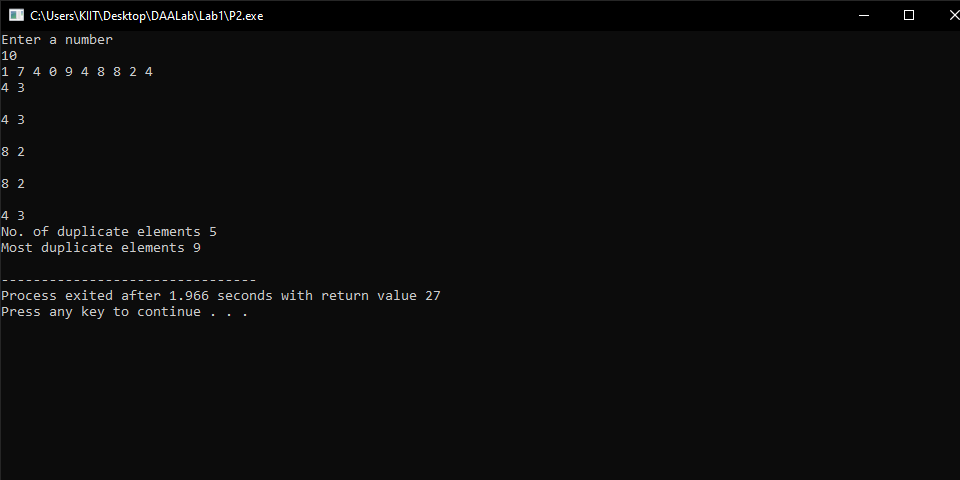
}

}

printf("No. of duplicate elements %d \n",c);

printf("Most duplicate elements %d \n",k);

}



**/\*1.3** Write a program to rearrange the elements of an array of n integers such that all even numbers are followed by all odd numbers. How many different ways you can solve this problem. Write your approaches & strategy for solving this problem.\*/

#include <stdio.h>

#include<stdlib.h>

int main()

{

int i,n;

printf("Enter a number\n");

scanf("%d",&n);

int a[n];

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

{

a[i] = rand()%100;

}

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

{

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

}

printf("\n");

for(i=0;i<n-1;i++)

{

if(a[i]%2!=0)

{

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

{

if(a[j]%2==0)

{

int t=a[i];

a[i]=a[j];

a[j]=t;

break;

}

}

}

}

printf("\n After seperating odd and even: \n");

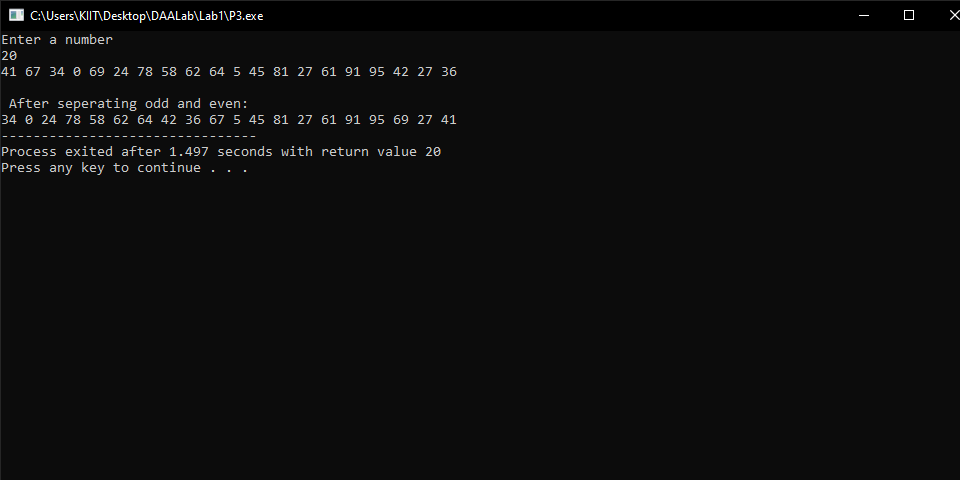
for(i=0;i<n;i++)

{

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

}

}



**/\*1.4** Write a program that takes three variable (a, b, c) as separate parameters and rotates the values stored so that value a goes to be, b, b to c and c to a by using SWAP(x,y) function that swaps/exchanges the numbers x & y.\*/

#include<stdio.h>

#include<stdlib.h>

void swapThree(int \*a, int \*b, int \*c)

{

// Store sum of all in a

\*a = \*a + \*b + \*c; // (a = 60)

// After this, b has value of a

\*b = \*a - (\*b+\*c); // (b = 60 – (20+30) =10)

// After this, c has value of b

\*c = \*a - (\*b+\*c); // (c = 60 – (10 + 30) = 20)

// After this, a has value of c

\*a = \*a - (\*b+\*c); //(a = 60 – (10 + 20) = 30)

}

int main()

{

int a = 10, b = 20, c = 30;

printf("Value before swapping:\n");

printf("a = %d \nb = %d \nc = %d\n", a, b, c);

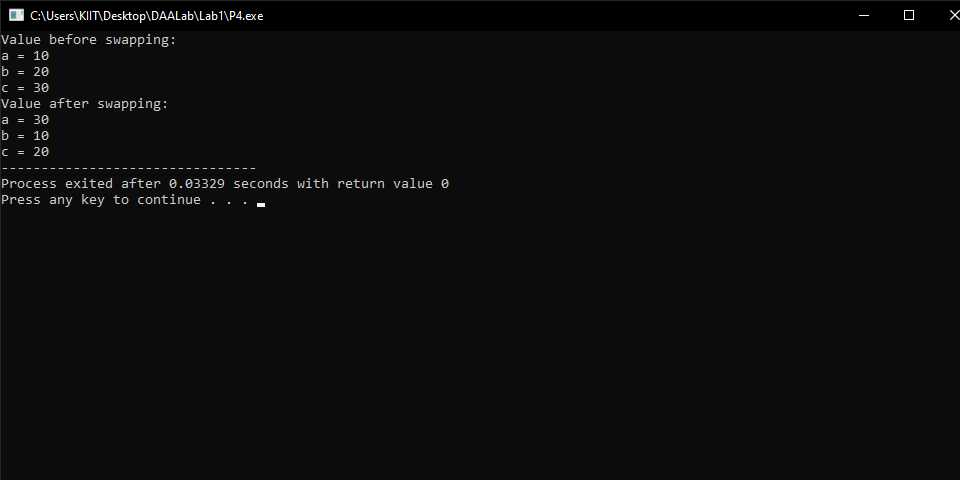
swapThree(&a,&b,&c);

printf("Value after swapping:\n");

printf("a = %d \nb = %d \nc = %d", a, b, c);

return 0;

}



**/\*1.5** Let A be n\*n square matrix array. WAP by using appropriate user defined functions for the following:

a) Find the number of nonzero elements in A

b) Find the sum of the elements above the leading diagonal.

c) Display the elements below the minor diagonal.

d) Find the product of the diagonal elements.\*/

#include <stdio.h>

#include<stdlib.h>

int main(){

int n, i, j;

printf("\n Enter the size of the matrix: ");

scanf("%d", &n);

int a[n][n];

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

{

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

{

a[i][j] = rand() % 10;

}

}

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

{

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

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

}

printf("\n");

}

int nonZero = 0;

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

{

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

{

if(a[i][j] != 0)nonZero++;

}

}

int sum= 0;

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

{

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

{

if(j>i)

sum+=a[i][j];

}

}

printf("a) Number of non-zero elements: %d\n",nonZero);

printf("b) Sum of elements above the leading diagonal: %d\n",sum);

printf("c) Elements below the minor diagonal: ");

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

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

if(i+j >= n){

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

}

}

}

printf("\n");

int major\_product =1;

int minor\_product =1;

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

major\_product \*= a[i][i];

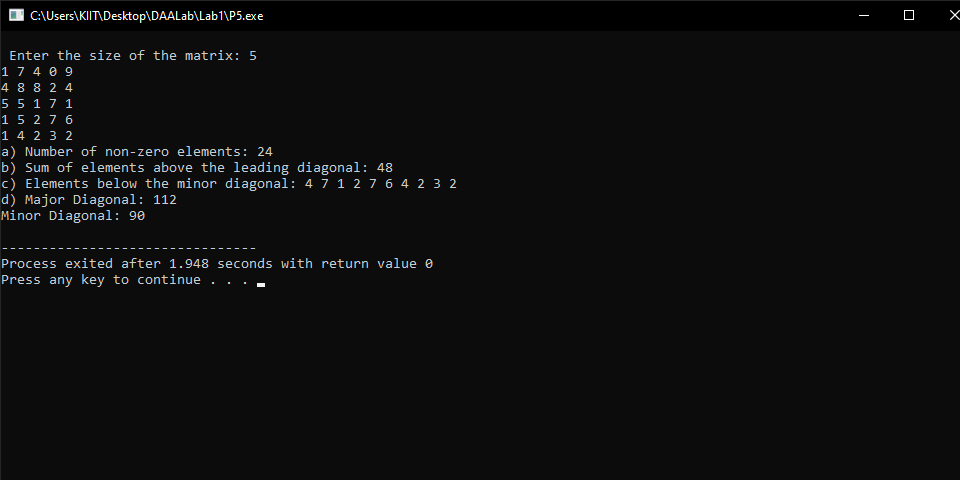
minor\_product \*= a[i][n-i-1];

}

printf("d) Major Diagonal: %d\nMinor Diagonal: %d\n", major\_product, minor\_product);

return 0;

}



**/\*1.6** Write a program to find out the second smallest and second largest element stored in an array of n integers. n is the user input. The array takes input through random number generation within a given range. How many different ways you can solve this problem.

Write your approaches & strategy for solving this problem.\*/

#include <stdio.h>

#include<stdlib.h>

void main () {

int number[30];

int i, j, a, n;

printf("Enter the value of n:\n");

scanf("%d", &n);

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

number[i]=rand()%10;

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

}

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

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

if (number[i] < number[j]){

a = number[i];

number[i] = number[j];

number[j] = a;

}

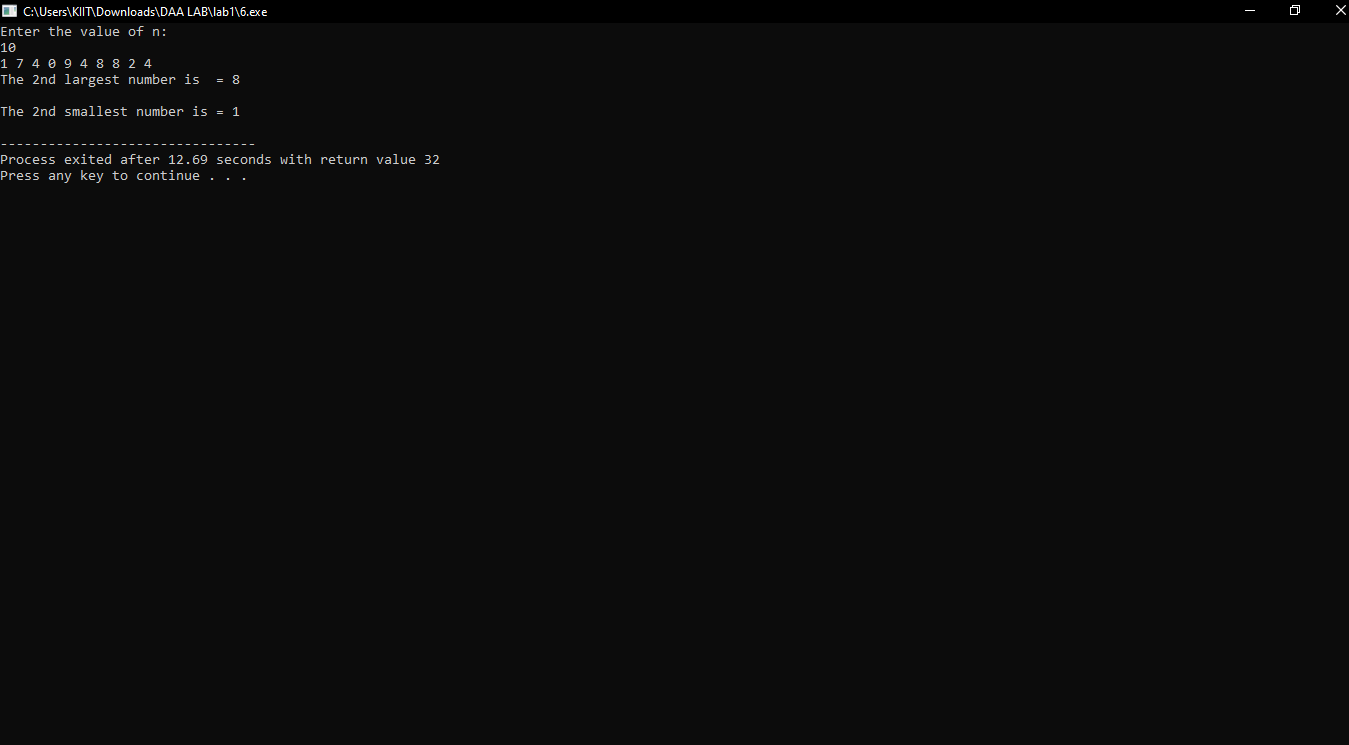
}

}

printf("\nThe 2nd largest number is = %d\n", number[1]);

printf("\nThe 2nd smallest number is = %d\n", number[n - 2]);

}



**/\*1.7** Write a program to swap pair of elements of an array of n integers from starting. If n is odd, then last number will be remain unchanged.\*/

#include<stdio.h>

#include<stdlib.h>

void swap(int \*a, int \*b);

void segregateEvenOdd(int arr[], int size)

{

int left = 0, right = size-1;

while (left < right)

{

while (arr[left]%2 == 0 && left < right)

left++;

while (arr[right]%2 == 1 && left < right)

right--;

if (left < right)

{

swap(&arr[left], &arr[right]);

left++;

right--;

}

}

}

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

int arr[] = {12, 34, 45, 9, 8, 90, 1};

int arr\_size = sizeof(arr)/sizeof(arr[0]);

int i = 0;

printf("\n Array is: ");

for (i = 0; i < arr\_size; i++)

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

segregateEvenOdd(arr, arr\_size);

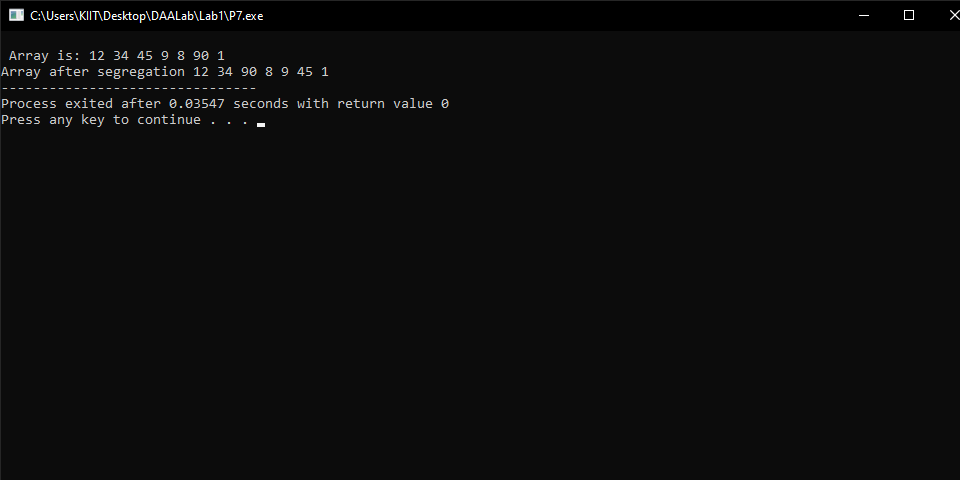
printf("Array after segregation ");

for (i = 0; i < arr\_size; i++)

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

return 0;

}



**/\*1.8** Write a program to display an array of n integers (n>1), where at every index of the array should contain the product of all elements in the array except the element at the given index. Solve this problem by taking single loop and without an additional array\*/

#include<stdio.h>

#include<stdlib.h>

public int[] product\_without\_self(int[] arr) {

int[] result = new int[arr.length];

result[result.length-1] = 1;

int i;

for(i=arr.length-2; i>=0; i--) {

result[i] = result[i+1] \* arr[i+1];

}

int left = 1;

for(i=0; i<arr.length; i++) {

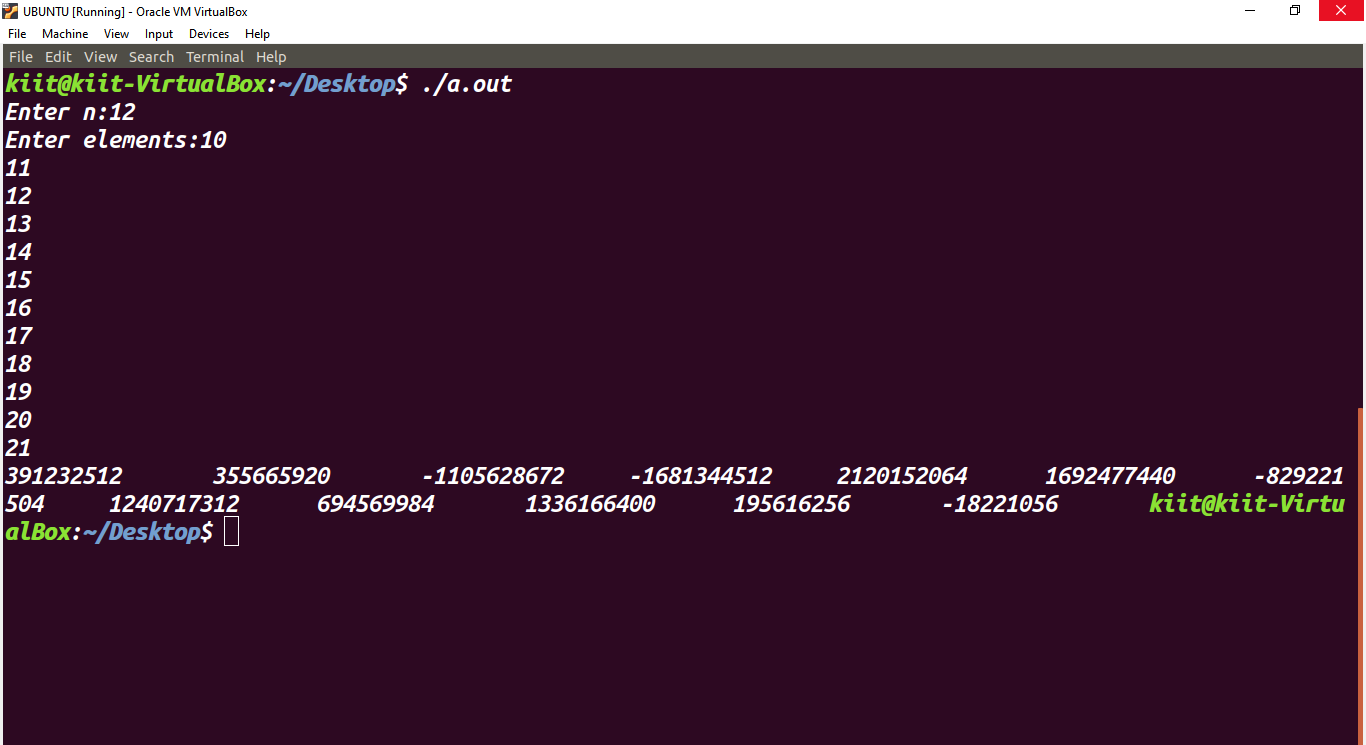
result[i] \*= left;

left \*= arr[i];

}

return result;

}



# LAB 2

**/\*2.1** Write a program to **test whether a number n, entered through keyboard is prime or not** by using different algorithms you know for atleast 10 inputs and note down the time complexity by step/frequency count method for each algorithm & for each input. Finally make a comparision of time complexities found for different inputs, plot an appropriate graph & decide which algothm is faster\*/

#include <stdio.h>

#include<stdlib.h>

int algo1(int n)

{

int i;

int counter=1;

int c=0;

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

{

counter++;

if(n%i==0)

{

c++;

counter++;

}

}

if(c==2){

counter++;

}

else{

counter++;

}

return counter;

}

int algo2(int n)

{

int i;

int counter=1,c=0;

for(i=2;i<n;i++)

{

counter++;

if(n%i==0)

{

counter++;

c++;

}

}

if(c==0){

counter++;

}

else

{

counter++;

}

return counter;

}

int algo3(int n)

{

int i, p;

int counter=1,prime[n+1];

for(i=0;i<n+1;i++)

{

prime[i]=1;

counter++;

}

for (p=2; p\*p<=n; p++)

{

counter++;

if (prime[p] == 1)

{

for(i=p\*p; i<=n; i += p)

prime[i] = 0;

counter++;

}

}

if(prime[n]==1){

counter++;

}

else{

counter++;

}

return counter;

}

int main()

{

int i;

printf("Number Algo1 Algo2 Algo3 Faster\n");

for(i=0;i<10;i++)

{

int n=rand()%100;

int p1=algo1(n);

int p2=algo2(n);

int p3=algo3(n);

printf("%d\t%d\t%d\t%d\t",n,p1,p2,p3);

if(p1<p2 && p1<p3)

{

printf(" algo1 is having least step\n");

}

else if(p2<p3 && p2<p1)

{

printf(" algo2 is having least step\n");

}

else

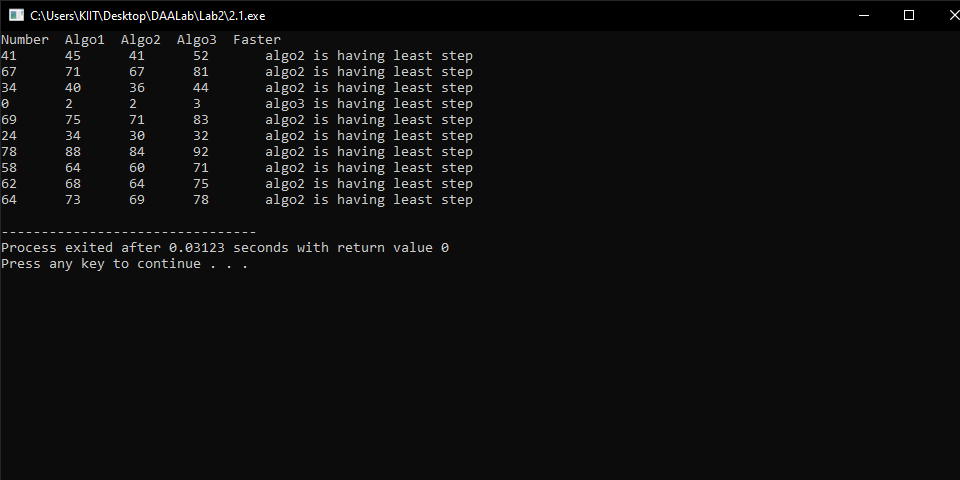
{

printf(" algo3 is having least step\n");

}

}

}



**/\*2.2** Write a program **to find out GCD (greatest common divisor)** using the following three algorithms.

a) Euclid’s algorithm

b) Consecutive integer checking algorithm.

c) Middle school procedure which makes use of common prime factors. For finding list of primes implement sieve of Eratosthenes algorithm.

Write a program to find out which algorithm is faster for the following data. Estimate how many times it will be faster than the other two by step/frequency count method in each case.

i. Find **GCD of two numbers when both are very larg**e i.e.GCD(31415, 14142) by applying each of the above algorithms.

ii. Find **GCD of two numbers when one of them can be very large** i.e.GCD(56, 32566) or GCD(34218, 56) by applying each of the above algorithms.

iii. Find **GCD of two numbers when both are very smal**l i.e.GCD(12,15) by applying each of the above algorithms.

iv. Find **GCD of two numbers when both are same** i.e.GCD(31415, 31415) or GCD(12, 12) by applying each of the above algorithms.

Write the above data in the following format and decide which algorithm is faster for the particular data.\*/

#include <stdio.h>

// Euclid's Algorithm

int counter1=0,counter2=0,counter3=0;

int algo1(int a, int b)

{

if (a == 0)

{

counter1++;

return b;

}

return algo1(b%a, a);

counter1++;

}

// Consecutive Algorithm checking

int algo2(int m,int n)

{

int t; counter2++;

if(m<n)

{

counter2++;

t=m;

counter2++;

}

else

{

counter2++;

t=n;

counter2++;

}

while(t>0)

{

counter2++;

if(m%t==0)

{

counter2++;

if(n%t==0)

{

counter2++;

return t;

}

}

t--; counter2++;

}

}

//Prime Number Method

int algo3(int a, int b)

{

if (a == 0)

{

counter3++;

return b;

}

if (b == 0)

{

counter3++;

return a;

}

if (a == b)

{

counter3++;

return a;

}

if (a > b)

{

counter3++;

return algo3(a-b, b);

}

return algo3(a, b-a);

counter3++;

}

int main()

{

int a[]={31415,56,34218,12,31415,12};

int b[]={14142,32566,56,15,31415,12};

int i;

printf("Numbers\t\t\tGCD by algo1\tSteps of Algo1\tGCD by algo2\tSteps of Algo2\tGCD by algo3\tSteps of Algo3\tFaster\n");

for(i=0;i<6;i++)

{

counter1=0;

counter2=0;

int p1=algo1(a[i],b[i]);

int p2=algo2(a[i],b[i]);

int p3=algo3(a[i],b[i]);

printf("GCD of (%d,%d)\t\t%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d",a[i],b[i],p1,counter1,p2,counter2,p3,counter3);

if(counter1<counter2 && counter1<counter3)

{

printf("\t\talgo1 is having least step\n");

}

else if(counter2<counter3 && counter2<counter1)

{

printf("\t\talgo2 is having least step\n");

}

else

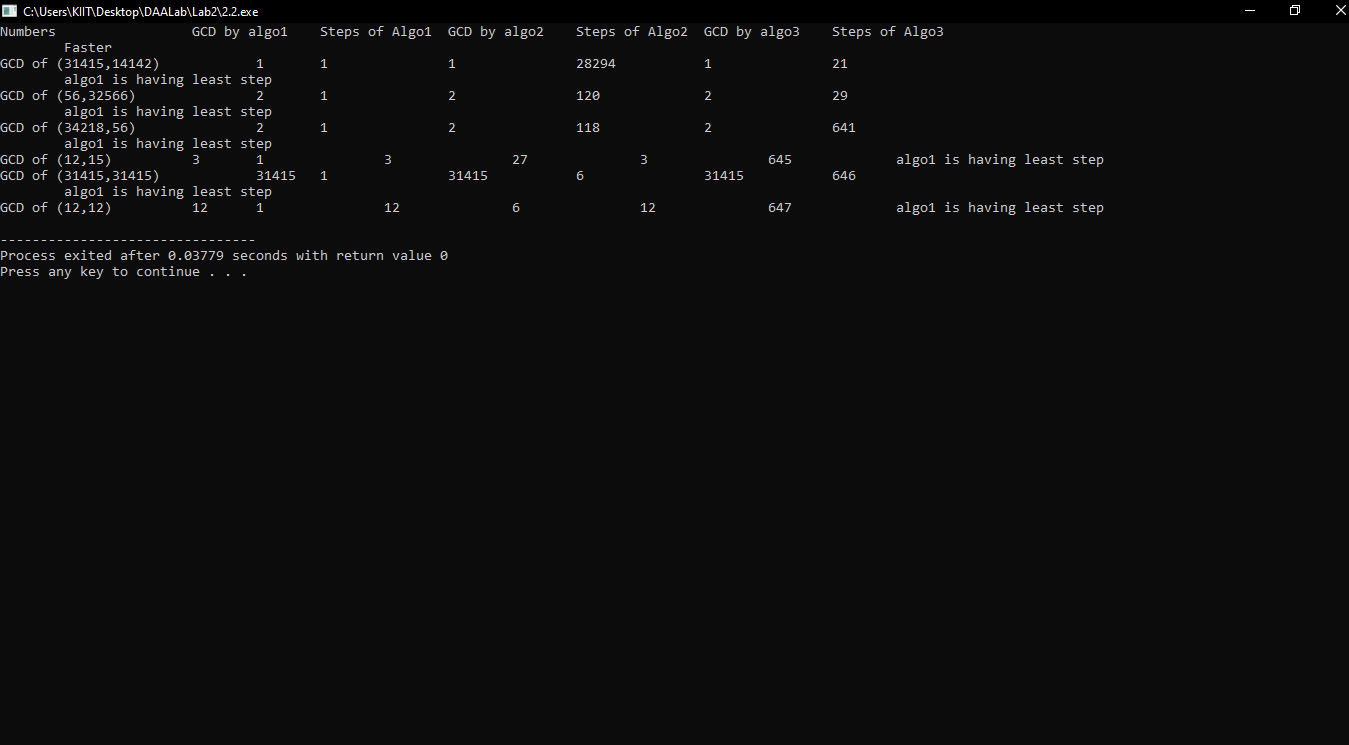
{

printf("\t\talgo3 is having least step\n");

}

}

}



**/\*2.3** Write a menu driven program as given below, to sort an array of n integers in ascending order by **insertion sort algorithm** and determine the **time required (in terms of step/frequency count)** to sort the elements. Repeat the experiment for different values of n and different nature of data (i.e.apply insertion sort algorithm on the data of array that are already sorted, reversly sorted and random data). Finally plot a graph of the time taken versus n for each type of data. The elements can be read from a file or can be generated using the random number generator.

#include<stdio.h>

#include<stdlib.h>

int count=1;

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

count=1;

int i,j,temp;

count++;

count++;

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

temp=arr[i];

//count++;

for(j=i-1; j>=0; j--){

//count++;

if(arr[j]>temp){

count++;

arr[j+1]=arr[j];

}

else{

count++;

break;

}

}

arr[j+1]=temp;

}

}

void reverseArray(int arr[], int start, int end)

{

count=0;

int temp;

count++;

count++;

while (start < end)

{

temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

start++;

end--;

count++;

}

}

int main(){

int n,a,as,ds,rnd,\*p;

char ch='y';

int \*arr;

do{

printf("\n Press 0 to exit\n Press 1 to Input Random Numbers\n Press 2 to Display the Data\n Press 3 to Sort data in Ascending Order\n Press 4 to Sort data in Descending order\n Press 5 to find Time Complexity for randomly sorted data\n Press 6 to find Time Complexity for ascending sorted data\n Press 7 to find Time Complexity for descending sorted data\n Press 8 to find Time Complexity for all cases of sorted data");

scanf("%d",&n);

if(n==0){

exit(0);

}

else if(n==1){

printf("Enter the desired size of the array");

scanf("%d",&a);

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

printf("\n %d ",a);

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

arr[i]=rand()%100;

}

else if(n==2){

printf("\n %d ",a);

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

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

}

}

else if(n==3){

insertion(arr,a);

}

else if(n==4){

insertion(arr,a);

reverseArray(arr,0,a-1);

}

else if(n==5){

insertion(arr,a);

printf("\n Average case Time Complexity is %d",count);

}

else if(n==6){

insertion(arr,a);

insertion(arr,a);

printf("\nBest Case Time Complexity is %d",count);

}

else if(n==7){

//insertion(arr,a);

reverseArray(arr,0,a-1);

insertion(arr,a);

printf("\nWorst Case Time Complexity is %d",count);

}

else if(n==8){

printf("\n\tAverage Best Worst");

for(int k=5000;k<=50000;k=k+5000){

p=(int \*)malloc(k\*sizeof(int));

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

\*(p+j)=rand()%100;

}

printf("\n%d",k);

insertion(p,k);

printf("\t%d",count);

insertion(p,k);

insertion(p,k);

printf(" %d",count);

reverseArray(p,0,k-1);

insertion(p,k);

printf(" %d",count);

}

}

//}\*/

else{

printf("\nNot in Option !!!!!");

}

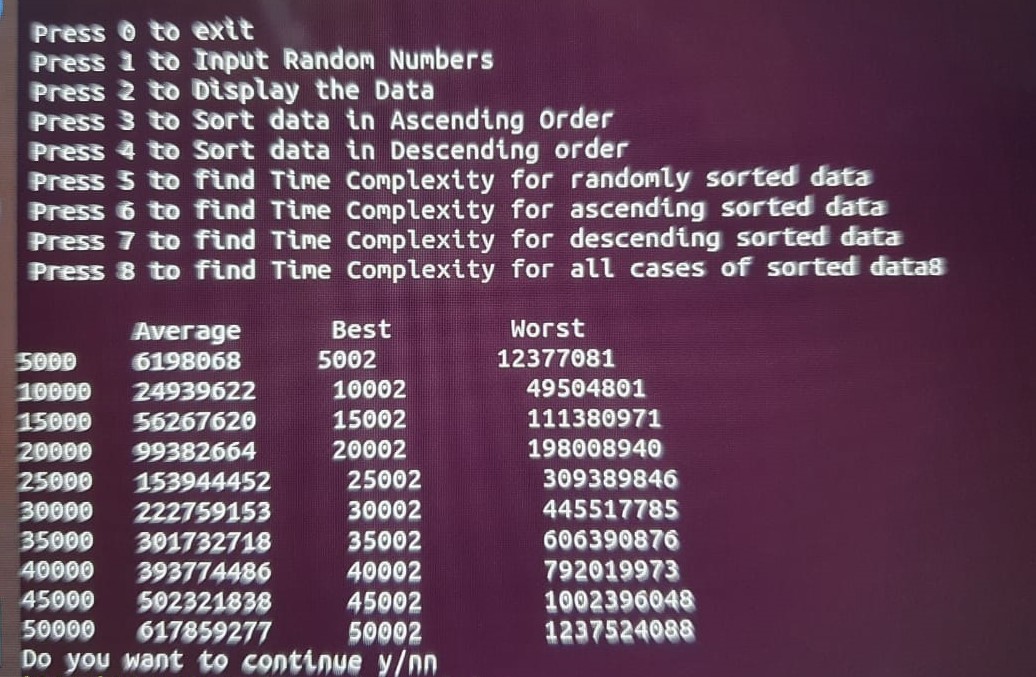
printf("\nDo you want to continue y/n");

scanf(" %c",&ch);

}while(ch=='y' || ch=='Y');

return 0;

}



# LAB 3

**/\*3.1** Rewrite the program no-2.1 **(Insertion Sort)** with the following details.

i. Compare the best case, worst case and average case time complexity with the same

data except time complexity will count the cpu clock time.

ii. Plot a graph showing the above comparison (n, the input data Vs. CPU times for

best, worst & average case)

iii. Compare manually program no-2.1 graph vs program no-3.1 graph and draw your

inference. \*/

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

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

int i,j,temp;

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

temp=arr[i];

for(j=i-1; j>=0; j--){

if(arr[j]>temp){

arr[j+1]=arr[j];

//arr[j+1]=temp;

}

else{

break;

}

}

arr[j+1]=temp;

}

}

void reverseArray(int arr[], int start, int end)

{

int temp;

while (start < end)

{

temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

start++;

end--;

}

//for(int i=start;i<=end;i++)

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

}

int main()

{

int i;

clock\_t start\_time,end\_time;

double total\_time;

int n,a,as,ds,rnd,\*p;

char ch='y';

int \*arr;

do{

printf("\n Press 0 to exit\n Press 1 to Input Random Numbers\n Press 2 to Display the Data\n Press 3 to Sort data in Ascending Order\n Press 4 to Sort data in Descending order\n Press 5 to find Time Complexity for randomly sorted data\n Press 6 to find Time Complexity for ascending sorted data\n Press 7 to find Time Complexity for descending sorted data\n Press 8 to find Time Complexity for all cases of sorted data");

scanf("%d",&n);

if(n==0){

exit(0);

}

else if(n==1){

printf("Enter the desired size of the array");

scanf("%d",&a);

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

printf("\n %d ",a);

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

arr[i]=rand()%100;

}

else if(n==2){

printf("\n %d ",a);

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

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

}

}

else if(n==3){

insertion(arr,a);

}

else if(n==4){

insertion(arr,a);

reverseArray(arr,0,a-1);

}

else if(n==5){

start\_time=clock();

insertion(arr,n);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

printf("\nAverage Case time = %f",total\_time);

}

else if(n==6){

insertion(arr,n);

start\_time=clock();

insertion(arr,n);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

printf("\nBest Case time = %f",total\_time);

}

else if(n==7){

insertion(arr,n);

reverseArray(arr,0,n-1);

start\_time=clock();

insertion(arr,n);

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

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

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

printf("\nWorst case time is %f",total\_time);

}

else if(n==8){

printf("\n\tAverage \tBest \t\tWorst");

for(int k=5000;k<=50000;k=k+5000){

p=(int \*)malloc(k\*sizeof(int));

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

\*(p+j)=rand()%100;

}

printf("\n%d",k);

start\_time=clock();

insertion(p,k);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

//printf("\nAverage Case time for %d = %f",k,total\_time);

printf("\t%f",total\_time);

start\_time=clock();

insertion(p,k);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

//printf("\nBest Case time = %f",total\_time);

printf("\t%f",total\_time);

reverseArray(p,0,k-1);

start\_time=clock();

insertion(p,k);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

printf("\t%f",total\_time);

}

}

//}\*/

else{

printf("\nNot in Option !!!!!");

}

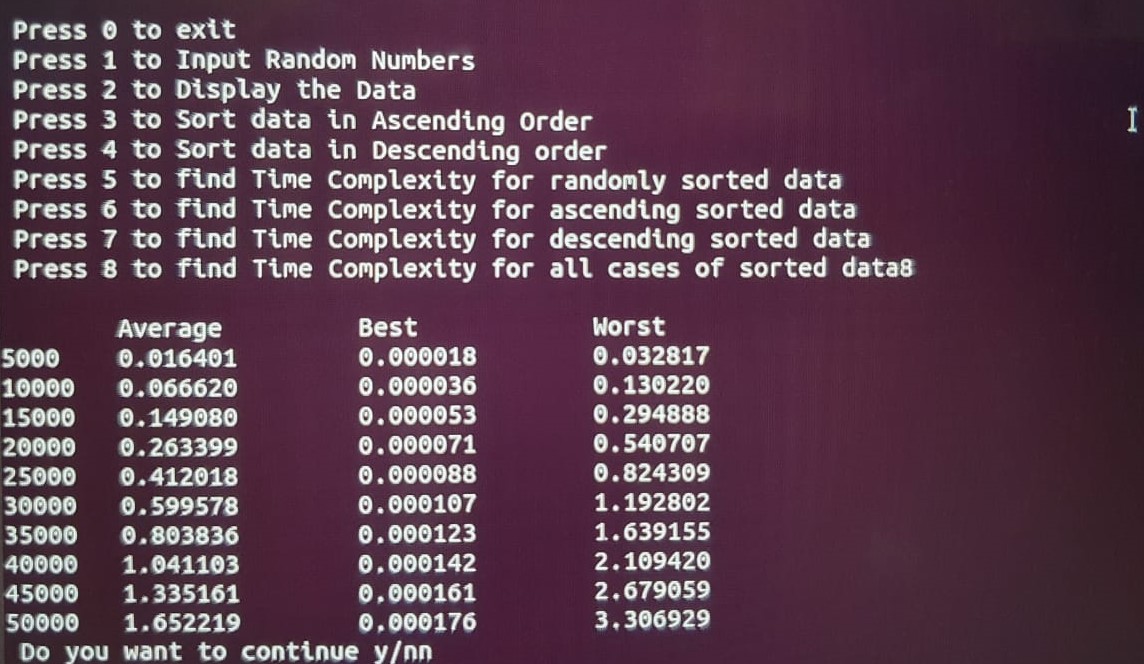
printf("\n Do you want to continue y/n");

scanf(" %c",&ch);

}while(ch=='y' || ch=='Y');

return 0;

}



**/\*3.2** Let A be a list of n (not necessarily distinct) integers. Write a program by using User Defined Function(UDF)s to test whether any item occurs more than ⌈ n/2⌉ times in A. a) UDF should take O(n2) time and use no additional space. b) UDF should take O(n) time and use O(1) additional space.\*/

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#include <time.h>

void udfn2(int arr[], int n)

{

int i, j;

int count =0,breakCase = ceil(n/2);

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

{

for(j=i+1;j<n;j++)

if(arr[i]==arr[j])

count++;

if(count>breakCase)

{

printf("More than %d occured",breakCase);

exit(0);

}

}

}

int main()

{

printf("Enter array size \t:");

int n, i, j;

scanf("%d",&n);

int arr[n];

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

{

arr[i] = rand()%n;

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

}

printf("\n");

clock\_t start\_t, end\_t;

double total\_t;

start\_t=clock();

udfn2(arr,n);

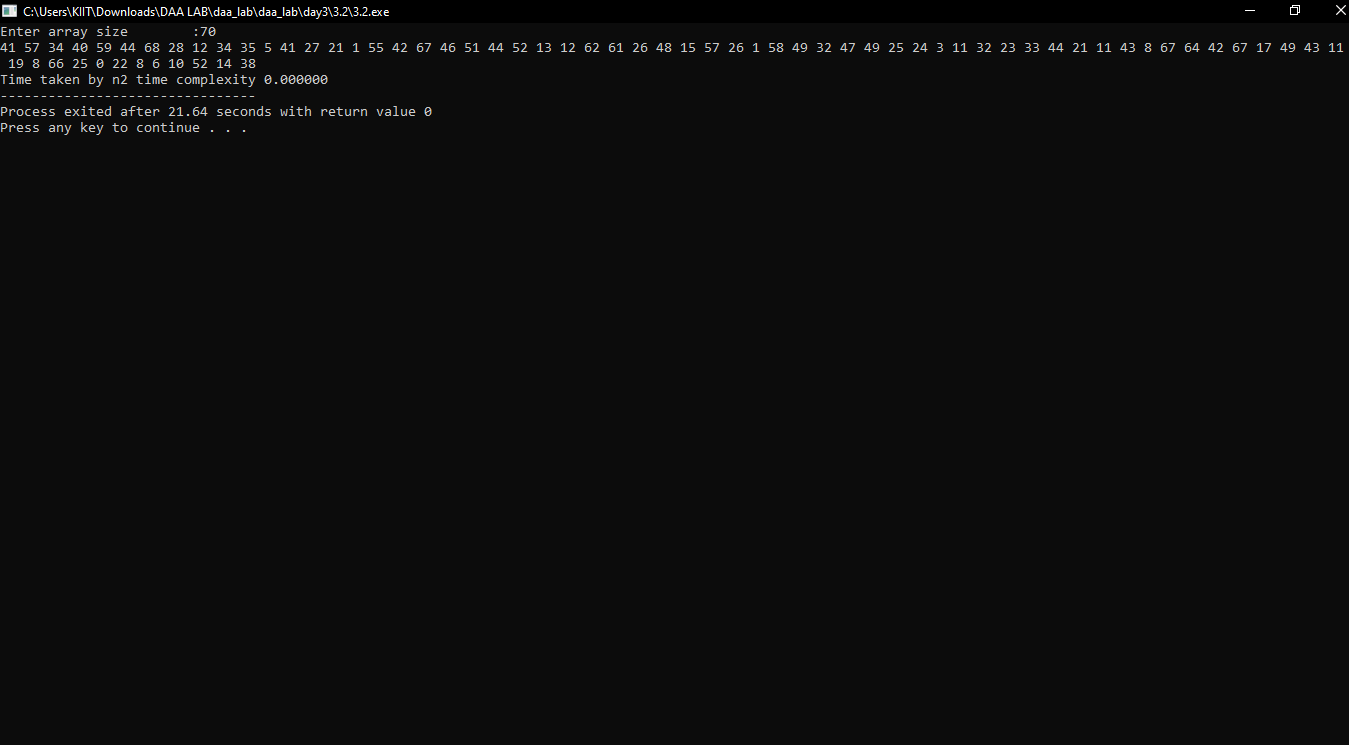
end\_t=clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("Time taken by n2 time complexity %f",total\_t);

return 0;

}



**/\*3.3** Write a program by using an user defined function for computing ⌊ √n⌋ for any

positive integer n. Besides assignment and comparison, your algorithm may only use the four basic arithmetical operations.\*/

#include<stdio.h>

#include<math.h>

double squareroot(int number)

{

float temp, sqrt;

// store the half of the given number e.g from 256 => 128

sqrt = number / 2;

temp = 0;

// Iterate until sqrt is different of temp, that is updated on the loop

while(sqrt != temp){

// initially 0, is updated with the initial value of 128

// (on second iteration = 65)

// and so on

temp = sqrt;

// Then, replace values (256 / 128 + 128 ) / 2 = 65

// (on second iteration 34.46923076923077)

// and so on

sqrt = ( number/temp + temp) / 2;

}

printf("The square root of '%d' is '%f'", number, sqrt);

return sqrt;

}

int main(){

int num;

double root;

printf("\n Enter the number to compute: ");

scanf("%d", &num);

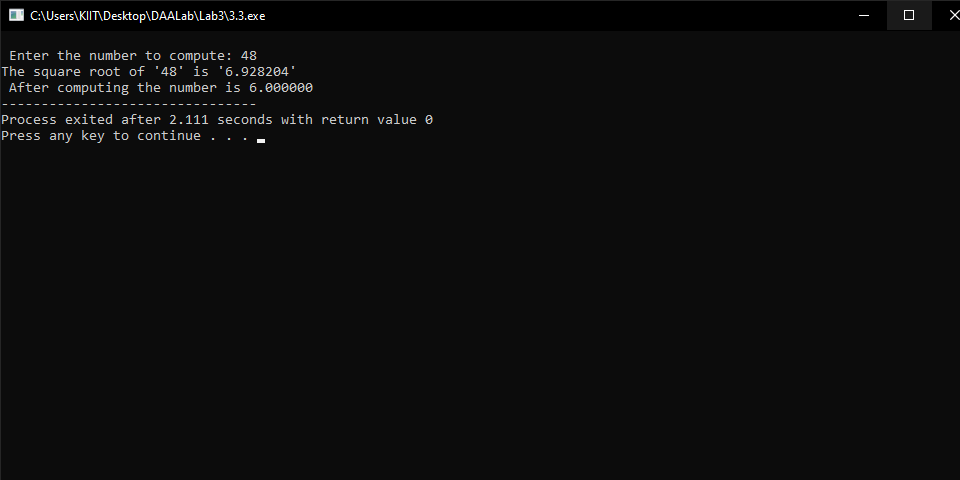
root=squareroot(num);

root=floor(root);

printf("\n After computing the number is %f", root);

return 0;

}



**/\*3.4** Let A be an array of n integers a0,a1,... ,an-1 (negative integers are allowed), denoted, by A[i... j], the sub-array ai, ai+1,... ,aj for i≤j. Also let Si-j denote the sum ai + ai+1 +· · · + aj.

**Write a program by using an udf that must run in O(n2) time to find out the**

**maximum value of Si-j for all the pair i, j with 0 ≤ i ≤ j ≤ n-1. Call this maximum**

**value S. Also obtains the maximum of these computed sums.** Let j < i in the

notation A[i... j] is also allowed. In this case, A[i... j] denotes the empty sub-array (that is, a sub-array that ends before it starts) with sum Si-j = 0. Indeed, if all the elements of A are negative, then one returns 0 as the maximum sub-array sum.

For example, for the array A[]={1, 3, 7, -2, -1, -5, -1, -2, -4, 6, 2}.

This maximum sum is S = S0-2 = 1+3+7=11.\*/

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

int maxSumOn2(int \*p, int n)

{

int i, j;

int max = INT\_MIN;

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

{

int sum = 0;

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

{

sum += p[j];

if (sum > max)

{

max = sum;

}

}

}

return max;

}

void main()

{

int n, i;

printf("Enter Number of Elements : ");

scanf("%d", &n);

int \*p = (int \*)calloc(n, sizeof(int));

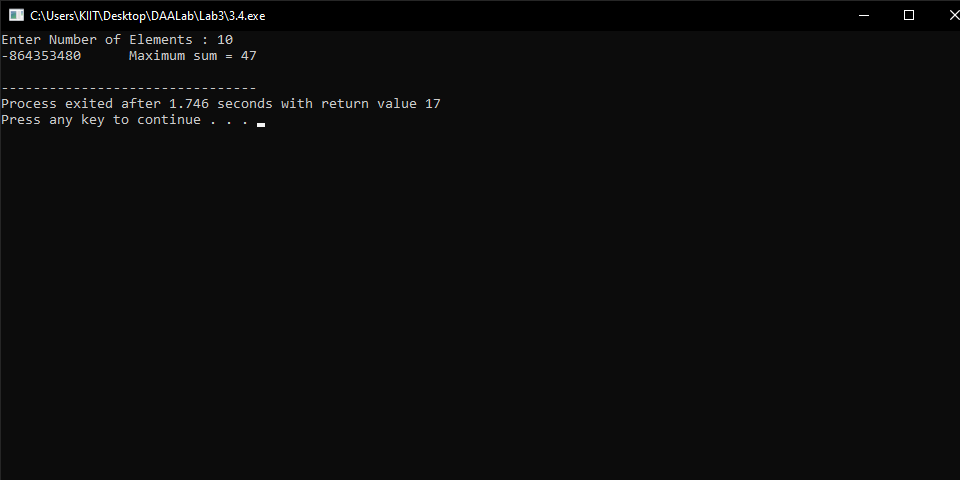
for (i = 0; i < n; i++)

\*(p + i) = rand() % 10;

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

printf("Maximum sum = %d\n", maxSumOn2(p, n));

}



# LAB 4

**/\*4.1** Write a program to search an element x in an array of n integers using **binary search** algorithm that uses divide and conquer technique. Find out the best case, worst case and average case time complexities for different values of n and plot a graph of the time taken versus n. The n integers can be generated randomly and x can be choosen randomly, or any element of the array or middle or last element of the array depending on type of time complexity analysis. \*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int binSearch(int arr[], int l, int h, int k)

{

if(h >= l)

{

int mid = (l+h)/2;

if(arr[mid] == k) return mid;

if(arr[mid] > k) return binSearch(arr,l,mid-1,k);

if(arr[mid] < k) return binSearch(arr,mid+1,h,k);

}

else

{

return -1;

}

}

int main()

{

printf("Array size\tWorst Case\tAverage Case\tBest case\n");

clock\_t start\_t, end\_t;

float total\_t;

int n=0,key;

while(n<=4500){

n+=500;

printf("n\t");

int arr[n];

int low = 0, high = n-1;

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

{

arr[i] = i;

}

//printf("\nWorst Case Analysis.......\n");

key = n;

start\_t = clock();

binSearch(arr,low,high,key);

end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("%f\t",total\_t);

//printf("Average Case Analysis.......\n");

key = random()%n;

start\_t = clock();

binSearch(arr,low,high,key);

end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("%f\t",total\_t);

//printf("Best Case Analysis.......\n");

key = arr[(low+high)/2];

start\_t = clock();

binSearch(arr,low,high,key);

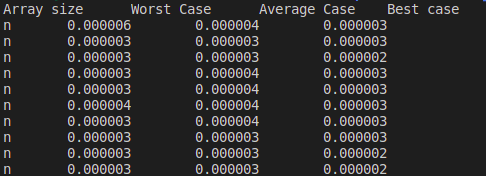
end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("%f\n",total\_t);

}

}





**/\*4.2** Write a program to **sort a list** of n elements using the **merge sort** method and determine the time required to sort the elements. Repeat the experiment for different values of n and different nature of data (random data, sorted data, reversely sorted data) in the list. n is the user input and n integers can be generated randomly. Finally plot a graph of the time taken versus n. \*/

/\*Write a program to sort a list of n elements using the merge sort method and determine

the time required to sort the elements. Repeat the experiment for different values of n

and different nature of data (random data, sorted data, reversely sorted data) in the list.

n is the user input and n integers can be generated randomly. Finally plot a graph of the

time taken versus n.\*/

#include <stdio.h>

#include <stdlib.h>

#include <time.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++;

}

while (i < n1) {

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 - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

int main()

{

clock\_t start\_t, end\_t;

double total\_t;

int n,key,i;

printf("Enter array size:\t");

scanf("%d",&n);

int arr[n];

int low = 0, high = n-1;

printf("\nSorted....\n");

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

{

arr[i] = i;

}

start\_t = clock();

mergeSort(arr,low,high);

end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("Time Taken = %f\n\n",total\_t);

printf("\nRandom....\n");

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

{

arr[i] = rand()%n;

}

start\_t = clock();

mergeSort(arr,low,high);

end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("Time Taken = %f\n\n",total\_t);

printf("\nReverse Sorted....\n");

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

{

arr[i] = n-i;

}

start\_t = clock();

mergeSort(arr,low,high);

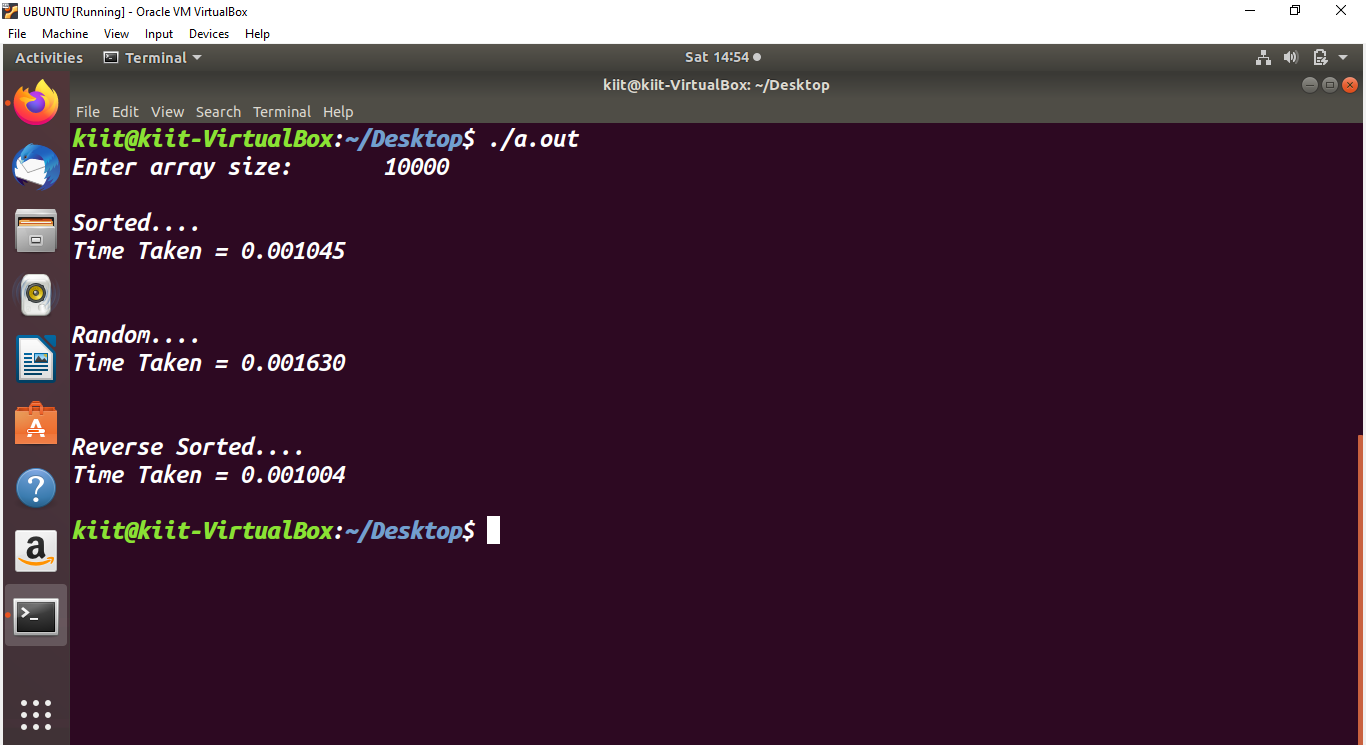
end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("Time Taken = %f\n\n",total\_t);

return 0;

}



|  |  |  |  |
| --- | --- | --- | --- |
| Array size | Sorted | Random | Reverse |
| 25000 | 0.0034 | 0.0055 | 0.0033 |
| 50000 | 0.007 | 0.0138 | 0.009 |
| 75000 | 0.0109 | 0.018 | 0.0106 |
| 100000 | 0.0147 | 0.0286 | 0.0154 |
| 125000 | 0.0198 | 0.0371 | 0.01977 |

**/\*4.3** Write a program to use divide and conquer method to determine the time required to find the maximum and minimum element in a list of n elements. The data for the list can be generated randomly. Compare this time with the time taken by straight forward algorithm or brute force algorithm for finding the maximum and minimum element for the same list of n elements. Show the comparison by plotting a required graph for this problem. \*/

#include <stdio.h>

#include <limits.h>

#include <stdlib.h>

#include <time.h>

int calmin(int arr[], int s,int e,int min)

{

int min1;

if(s==e) //one element

{

if(min>arr[s])

min=arr[s];

return arr[s];

}

if(s-e==1) //two elements

{

if(arr[s]<arr[e])

{

if(min>arr[s])

min = arr[s];

}

else

{

if(min>arr[e])

min = arr[e];

}

return 0;

}

int middle = (s+e)/2;

calmin(arr,s,middle,min);

min1=min;

calmin(arr,middle+1,e,min);

if(min>min1)

min=min1;

return 0;

}

int calmax(int arr[], int s,int e, int max)

{

int max1;

if(s==e) //one element

{

if(max<arr[s])

max=arr[s];

return arr[s];

}

if(s-e==1) //two elements

{

if(arr[s]<arr[e])

{

if(max<arr[e])

max = arr[e];

}

else

{

if(max<arr[s])

max = arr[s];

}

return 0;

}

int middle = (s+e)/2;

calmax(arr,s,middle,max);

max1=max;

calmax(arr,middle+1,e,max);

if(max>max1)

max=max1;

return 0;

}

void bruteForceCalMin(int arr[],int n)

{

int minim = arr[0], i;

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

{

if(minim>arr[i])

minim = arr[i];

}

printf("Min:%d",minim);

}

void bruteForceCalMax(int arr[],int n)

{

int i;

int maxim = arr[0];

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

{

if(maxim<arr[i])

maxim = arr[i];

}

printf("Max:%d",maxim);

}

int main()

{ int in=1;

int i;

int n=0,maxim;

while(n<=200000){

printf("enter the size of array\t");

n+=20000;

printf("%d\n",n);

int arr[n];

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

arr[i]=rand()%n;

clock\_t start\_t, end\_t;

double total\_t;

start\_t = clock();

bruteForceCalMin(arr,n);

end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("\nTime Taken with Brute Force = %f\n\n",total\_t);

start\_t = clock();

bruteForceCalMax(arr,n);

end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("\nTime Taken with Brute Force = %f\n\n",total\_t);

start\_t = clock();

maxim = calmin(arr,0,n-1,INT\_MAX);

end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("Time Taken with Dn max= %f\n\n",total\_t);

start\_t = clock();

maxim = calmax(arr,0,n-1,INT\_MIN);

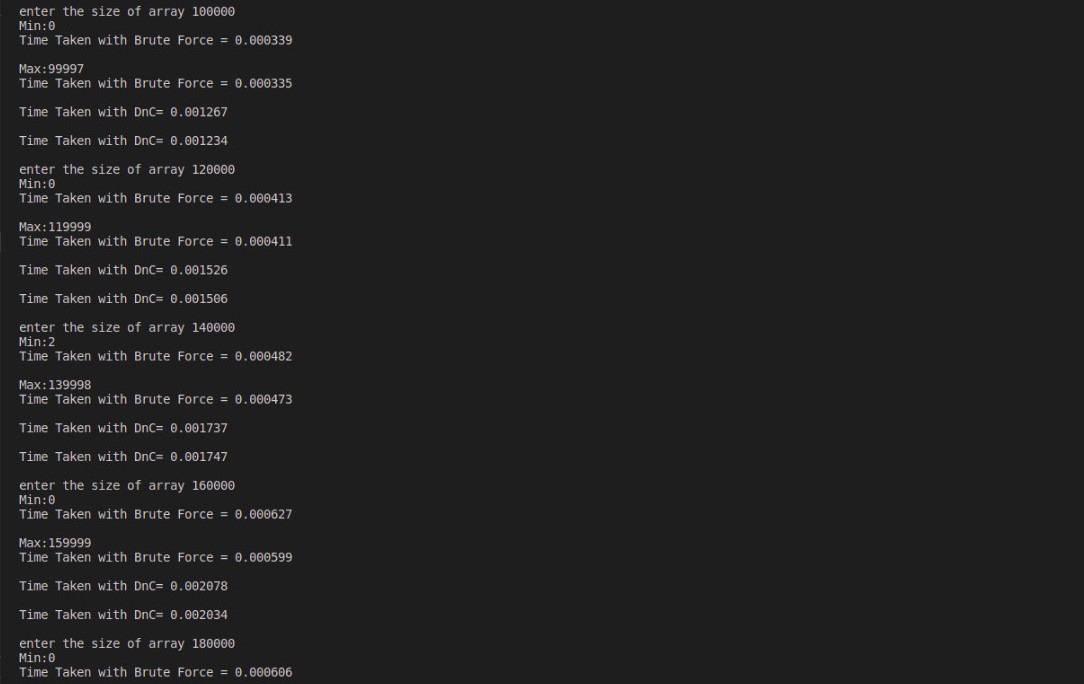
end\_t = clock();

total\_t = (double)(end\_t - start\_t) / CLOCKS\_PER\_SEC;

printf("Time Taken with DnC min= %f\n\n",total\_t);

}

}



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Array size** | **Brute Force minimum** | **Brute Force maximum** | **DnC minimum** | **DnC maximum** |
| **20000** | 0.000214 | 0.000208 | 0.000701 | 0.000281 |
| **40000** | 0.000142 | 0.00014 | 0.000531 | 0.000534 |
| **60000** | 0.000213 | 0.000215 | 0.000783 | 0.000784 |
| **80000** | 0.000391 | 0.000327 | 0.00135 | 0.001452 |
| **100000** | 0.000445 | 0.000405 | 0.001337 | 0.001306 |
| **120000** | 0.000414 | 0.000407 | 0.001553 | 0.00155 |
| **140000** | 0.000521 | 0.000485 | 0.001747 | 0.002138 |
| **160000** | 0.000613 | 0.000575 | 0.002087 | 0.002252 |
| **180000** | 0.000619 | 0.000625 | 0.002993 | 0.002688 |
| **200000** | 0.000701 | 0.000674 | 0.002373 | 0.002397 |

**/\*4.4**

Write a program that uses a divide-and-conquer algorithm/user defined function for the exponentiation problem of computing an where a > 0 and n is a positive integer. How does this algorithm compare with the brute-force algorithm in terms of number of multiplications made by both algorithms.\*/

#include <stdio.h>

int count1 = 0;

int count2 = 0;

int powerDivideAndConquer(int x, int n)

{

if(n == 0)

return 1;

int temp = powerDivideAndConquer(x, n/2);

count1++;

temp = temp \* temp;

if(n % 2 == 0)

return temp;

else

{

count1++;

return temp \* x;

}

}

int powerBruteForce(int x, int n)

{

int i;

int res = 1;

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

count2++;

res = res \* x ;

return res;

}

int main()

{

int x, n;

printf("Enter x & n :");

scanf("%d %d", &x, &n);

printf("power(x, n)= %d\n", powerDivideAndConquer(x, n));

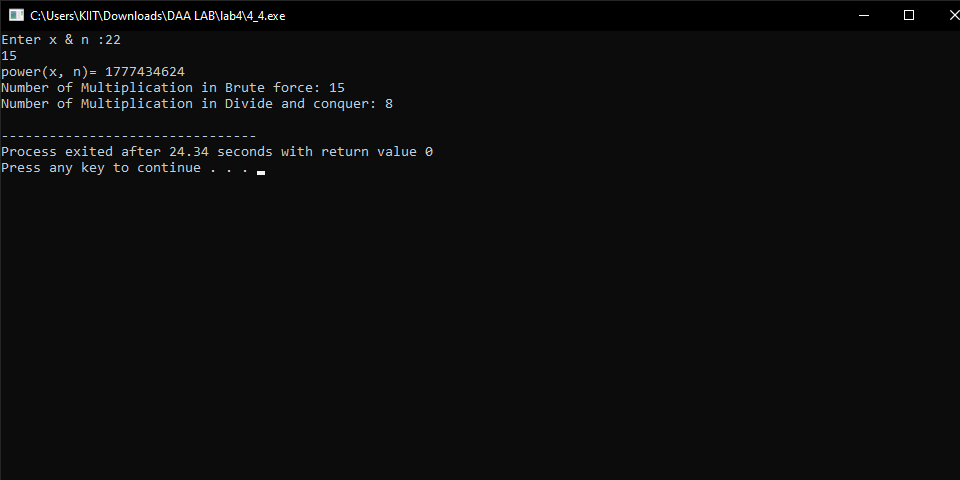
int res = powerBruteForce(x, n);

printf("Number of Multiplication in Brute force: %d\n",count2);

printf("Number of Multiplication in Divide and conquer: %d\n",count1);

return 0;

}



# LAB 5

**/\*5.1** Write a menu (given as follows) driven program to sort an array of n integers in ascending order by heap sort algorithm and perform the operations on max heap.

Determine the time required to sort the elements. Repeat the experiment for different

values of n, the number of elements in the array to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.\*/

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int n1;

void reverseArray(int arr[], int start, int end)

{

int temp;

while (start < end)

{

temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

start++;

end--;

}

}

void max\_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 (i!= largest)

{

int temp = arr[i];

arr[i]= arr[largest];

arr[largest] = temp;

max\_heapify(arr,n,largest);

}

}

void heap\_create(int arr[],int n){

int i;

for(i =((n/2)-1);i>=0;i--)

max\_heapify(arr,n,i);

}

void heap\_sort(int arr[],int n){

heap\_create(arr,n);

while(n>1){

int temp=arr[0];

arr[0]=arr[n-1];

arr[n-1]=temp;

n--;

max\_heapify(arr,n,0);

}

}

int heap\_extract\_max(int arr[],int n){

int max=arr[0];

arr[0]=arr[n-1];

n--;

heap\_create(arr,n);

//max\_heapify(arr,n,0);

return max;

}

void heap\_replace(int arr[],int n,int key,int pos){

arr[pos]=key;

max\_heapify(arr,n,0);

}

void heap\_insert(int arr[],int \*n,int new\_val){

\*n=\*n+1;

//heap\_replace(arr,n,new\_val,n-1);

arr[\*n-1]=new\_val;

int i=\*n-1;

while(i>=0 && arr[i]>arr[(i-1)/2]){

int temp=arr[i];

arr[i]=arr[(i-1)/2];

arr[(i-1)/2]=temp;

i=(i-1)/2;

}

}

int heap\_delete(int arr[],int\* n,int val){

//n1=n;

int p=\*n;

int i;

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

if(arr[i]==val){

int temp=arr[i];

arr[i]=arr[p-1];

arr[p-1]=temp;

p--;

//printf("%d",p);

max\_heapify(arr,p,i);

break;

}

}

return p;

}

int main(){

int i,j, k;

clock\_t start\_time,end\_time;

double total\_time;

int n,size,as,ds,rnd,\*p;

char ch='y';

int \*arr;

//int arr[100];

int pos,val,del;

do{

printf("\n Press 0 to exit\n Press 1 to Input Random Numbers\n Press 2 to Display the Data\n Press 3 to Sort data in Ascending Order using max-heap\n Press 4 to Sort data in Descending order using any algorithm\n Press 5 to find Time Complexity for ascending order of randomly sorted data\n Press 6 to find Time Complexity for ascending of ascending sorted data\n Press 7 to find Time Complexity for ascending of descendingly sorted data\n Press 8 to find Time Complexity for all cases of sorted data\n Press 9 to extract the maximum element\n Press 10 to replace the value of a node with a new value\n Press 11 to insert an element\n Press 12 to delete an element");

scanf("%d",&n);

if(n==0){

exit(0);

}

else if(n==1){

printf("Enter the desired size of the array");

scanf("%d",&size);

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

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

arr[i]=rand()%100;

}

heap\_create(arr,size);

}

else if(n==2){

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

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

}

}

else if(n==3){

heap\_sort(arr,size);

}

else if(n==4){

heap\_sort(arr,size);

reverseArray(arr,0,size-1);

}

else if(n==5){

start\_time=clock();

heap\_sort(arr,size);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

printf("\nTime taken to sort data in ascending of randomn data= %f",total\_time);

}

else if(n==6){

heap\_sort(arr,size);

start\_time=clock();

heap\_sort(arr,size);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

printf("\nTime taken to sort data in ascending of ascendingly sorted data= %f",total\_time);

}

else if(n==7){

heap\_sort(arr,size);

reverseArray(arr,0,size-1);

start\_time=clock();

heap\_sort(arr,size);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

printf("\nTime taken to sort data in ascending of descendingly sorted data= %f",total\_time);

}

else if(n==8){

printf("\n\t\t Worst Case\t Average Case \tBest Case\n");

for(k=5000;k<=50000;k=k+5000){

p=(int \*)malloc(k\*sizeof(int));

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

\*(p+j)=rand()%100;

}

printf("%d",k);

heap\_sort(p,k);

reverseArray(p,0,k-1);

start\_time=clock();

heap\_sort(p,k);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

//printf("\nWorst Case time = %f",total\_time);

printf("\t\t%f",total\_time);

start\_time=clock();

heap\_sort(p,k);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

//printf("\nAverage Case time for %d = %f",k,total\_time);

printf("\t%f",total\_time);

start\_time=clock();

heap\_sort(p,k);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

//printf("\nBest Case time = %f",total\_time);

printf("\t%f",total\_time);

printf("\n");

}

}

else if(n==9){

heap\_create(arr,size);

int max=heap\_extract\_max(arr,size);

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

}

else if(n==10){

printf("\nEnter the position where you want to replace");

scanf("%d",&pos);

printf("\nEnter the new value");

scanf("%d",&val);

heap\_create(arr,size);

heap\_replace(arr,size,val,pos);

}

else if(n==11){

printf("\nEnter the element you want to insert");

int new\_val;

scanf("%d",&new\_val);

//size++;

heap\_insert(arr,&size,new\_val);

}

else if(n==12){

printf("\nEnter the element you want to delete");

scanf("%d",&del);

int d;

d=heap\_delete(arr,&size,del);

size=d;

printf("\nDeleted %d",size);

}

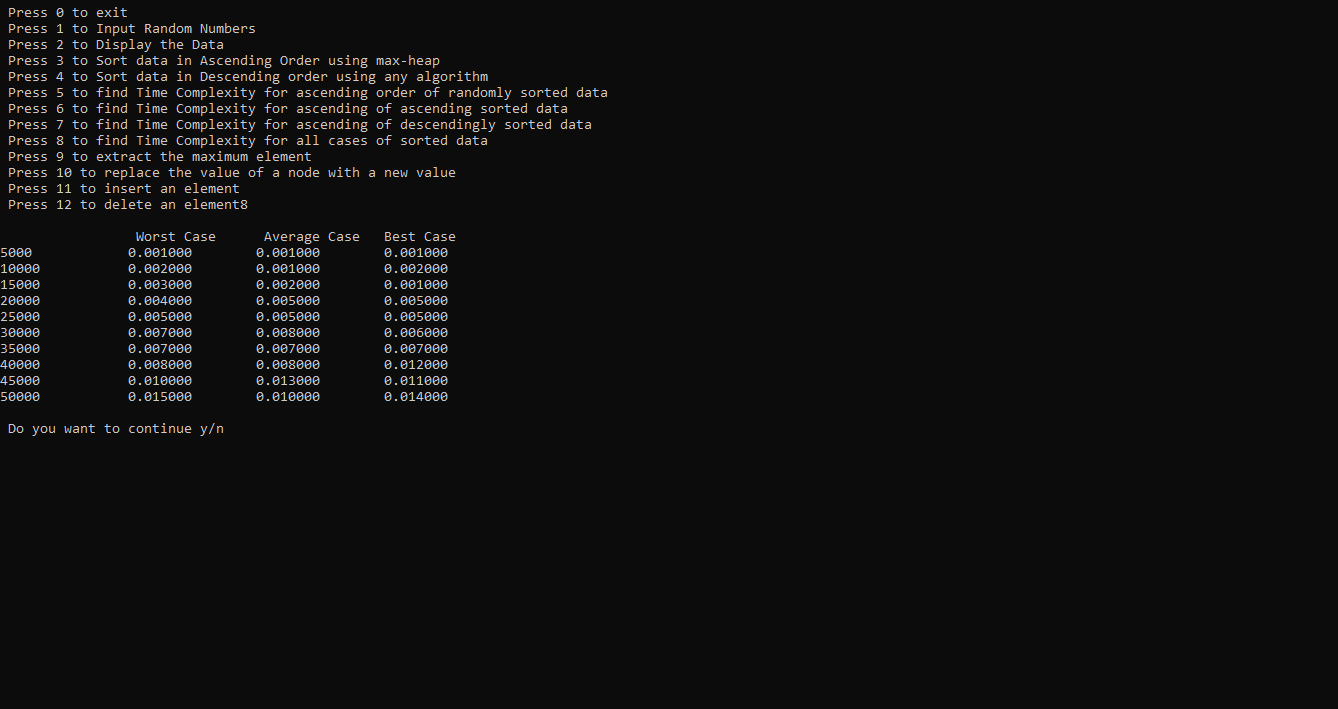
printf("\n Do you want to continue y/n");

scanf(" %c",&ch);

}while(ch=='y' || ch=='Y');

return 0;

}



**/\*5.2** Similar to above program no.5.1, write a menu driven program to sort an array of n integers in **descending order by heap sort algorithm.** Hints: Use min heap and accordingly change the menu options.\*/

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int n1;

void reverseArray(int arr[], int start, int end)

{

int temp;

while (start < end)

{

temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

start++;

end--;

}

}

void min\_heapify(int arr[], int n, int i)

{

int min=i;

int left=(2\*i)+1;

int right=(2\*i)+2;

if (left<n && arr[left]<arr[min])

min =left;

if (right<n && arr[right]<arr[min])

min = right;

if (i!= min)

{

int temp = arr[i];

arr[i]= arr[min];

arr[min] = temp;

min\_heapify(arr,n,min);

}

}

void heap\_create(int arr[],int n){

int i;

for(i =((n/2)-1);i>=0;i--)

min\_heapify(arr,n,i);

}

void heap\_sort(int arr[],int n){

heap\_create(arr,n);

while(n>1){

int temp=arr[0];

arr[0]=arr[n-1];

arr[n-1]=temp;

n--;

min\_heapify(arr,n,0);

}

}

int heap\_extract\_min(int arr[],int n){

int min=arr[0];

arr[0]=arr[n-1];

n--;

heap\_create(arr,n);

//min\_heapify(arr,n,0);

return min;

}

void heap\_replace(int arr[],int n,int key,int pos){

arr[pos]=key;

min\_heapify(arr,n,0);

}

void heap\_insert(int arr[],int \*n,int new\_val){

\*n=\*n+1;

//heap\_replace(arr,n,new\_val,n-1);

arr[\*n-1]=new\_val;

int i=\*n-1;

while(i>=0 && arr[i]>arr[(i-1)/2]){

int temp=arr[i];

arr[i]=arr[(i-1)/2];

arr[(i-1)/2]=temp;

i=(i-1)/2;

}

}

int heap\_delete(int arr[],int\* n,int val){

//n1=n;

int p=\*n;

int i;

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

if(arr[i]==val){

int temp=arr[i];

arr[i]=arr[p-1];

arr[p-1]=temp;

p--;

//printf("%d",p);

min\_heapify(arr,p,i);

break;

}

}

return p;

}

int main(){

int i,j, k;

clock\_t start\_time,end\_time;

double total\_time;

int n,size,as,ds,rnd,\*p;

char ch='y';

int \*arr;

//int arr[100];

int pos,val,del;

do{

printf("\n Press 0 to exit\n Press 1 to Input Random Numbers\n Press 2 to Display the Data\n Press 3 to Sort data in Descending Order using min-heap\n Press 4 to Sort data in Ascending order using any algorithm\n Press 5 to find Time Complexity for ascending order of randomly sorted data\n Press 6 to find Time Complexity for ascending of ascending sorted data\n Press 7 to find Time Complexity for ascending of descendingly sorted data\n Press 8 to find Time Complexity for all cases of sorted data\n Press 9 to extract the minimum element\n Press 10 to replace the value of a node with a new value\n Press 11 to insert an element\n Press 12 to delete an element");

scanf("%d",&n);

if(n==0){

exit(0);

}

else if(n==1){

printf("Enter the desired size of the array");

scanf("%d",&size);

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

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

arr[i]=rand()%100;

}

heap\_create(arr,size);

}

else if(n==2){

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

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

}

}

else if(n==3){

heap\_sort(arr,size);

}

else if(n==4){

heap\_sort(arr,size);

reverseArray(arr,0,size-1);

}

else if(n==5){

start\_time=clock();

heap\_sort(arr,size);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

printf("\nTime taken to sort data in ascending of randomn data= %f",total\_time);

}

else if(n==6){

heap\_sort(arr,size);

start\_time=clock();

heap\_sort(arr,size);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

printf("\nTime taken to sort data in ascending of ascendingly sorted data= %f",total\_time);

}

else if(n==7){

heap\_sort(arr,size);

reverseArray(arr,0,size-1);

start\_time=clock();

heap\_sort(arr,size);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

printf("\nTime taken to sort data in ascending of descendingly sorted data= %f",total\_time);

}

else if(n==8){

printf("\n\t\t Worst Case\t Average Case \tBest Case\n");

for(k=5000;k<=50000;k=k+5000){

p=(int \*)malloc(k\*sizeof(int));

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

\*(p+j)=rand()%100;

}

printf("%d",k);

heap\_sort(p,k);

reverseArray(p,0,k-1);

start\_time=clock();

heap\_sort(p,k);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

//printf("\nWorst Case time = %f",total\_time);

printf("\t\t%f",total\_time);

start\_time=clock();

heap\_sort(p,k);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

//printf("\nAverage Case time for %d = %f",k,total\_time);

printf("\t%f",total\_time);

start\_time=clock();

heap\_sort(p,k);

end\_time=clock();

total\_time=((double)(end\_time-start\_time))/CLOCKS\_PER\_SEC;

//printf("\nBest Case time = %f",total\_time);

printf("\t%f",total\_time);

printf("\n");

}

}

else if(n==9){

heap\_create(arr,size);

int min=heap\_extract\_min(arr,size);

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

}

else if(n==10){

printf("\nEnter the position where you want to replace");

scanf("%d",&pos);

printf("\nEnter the new value");

scanf("%d",&val);

heap\_create(arr,size);

heap\_replace(arr,size,val,pos);

}

else if(n==11){

printf("\nEnter the element you want to insert");

int new\_val;

scanf("%d",&new\_val);

//size++;

heap\_insert(arr,&size,new\_val);

}

else if(n==12){

printf("\nEnter the element you want to delete");

scanf("%d",&del);

int d;

d=heap\_delete(arr,&size,del);

size=d;

printf("\nDeleted %d",size);

}

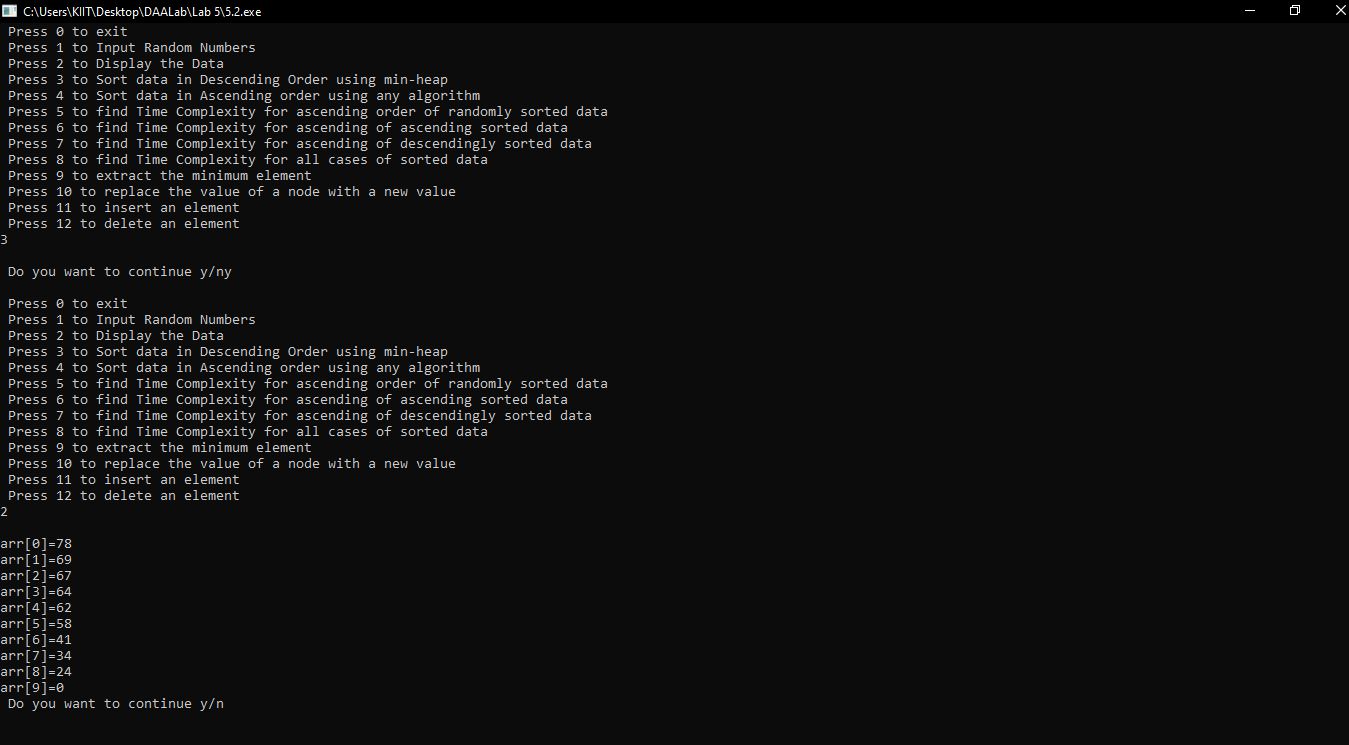
printf("\n Do you want to continue y/n");

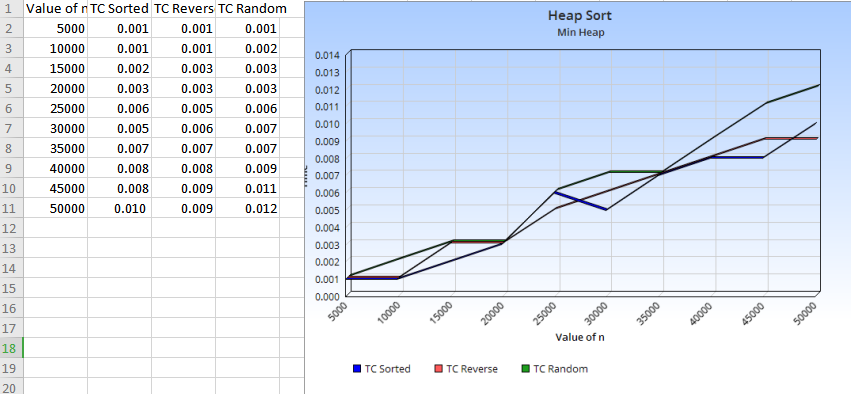
scanf(" %c",&ch);

}while(ch=='y' || ch=='Y');

return 0;

}





# LAB 6

/\*6.1 Write a program to implementation of Fractional Knapsack algorithm.\*/

#include <stdio.h>

#include<stdio.h>

void knapsack(int n, int c[], int v[], int W)

{

int curr\_w;

float tot\_v;

int i, obj;

int used[1000];

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

used[i] = 0;

curr\_w = W;

while (curr\_w > 0)

{

obj= -1;

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

if ((used[i] == 0) && ((obj == -1) || ((float)v[i]/c[i] > (float)v[obj]/c[obj])))

obj = i;

used[obj] = 1;

curr\_w -= c[obj];

tot\_v += v[obj];

if (curr\_w >= 0)

printf("Added object %d (%dRs, %dKg) completely in the bag. Space left: %d.\n", obj + 1, v[obj], c[obj], curr\_w);

else

{

printf("Added %d%% (%dRs, %dKg) of object %d in the bag.\n", (int)((1 + (float)curr\_w/c[obj]) \* 100), v[obj], c[obj], obj + 1);

tot\_v -= v[obj];

tot\_v += (1 + (float)curr\_w/c[obj]) \* v[obj];

}

}

printf("Filled the bag with objects worth %.2fRs.\n", tot\_v);

}

int main(int argc, char \*argv[])

{

int n, v[1000], c[1000], W, i;

printf("\nEnter number of objects: ");

scanf("%d",&n);

printf("\n Enter weight of the objects: ");

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

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

printf("\n Enter the value of each object: ");

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

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

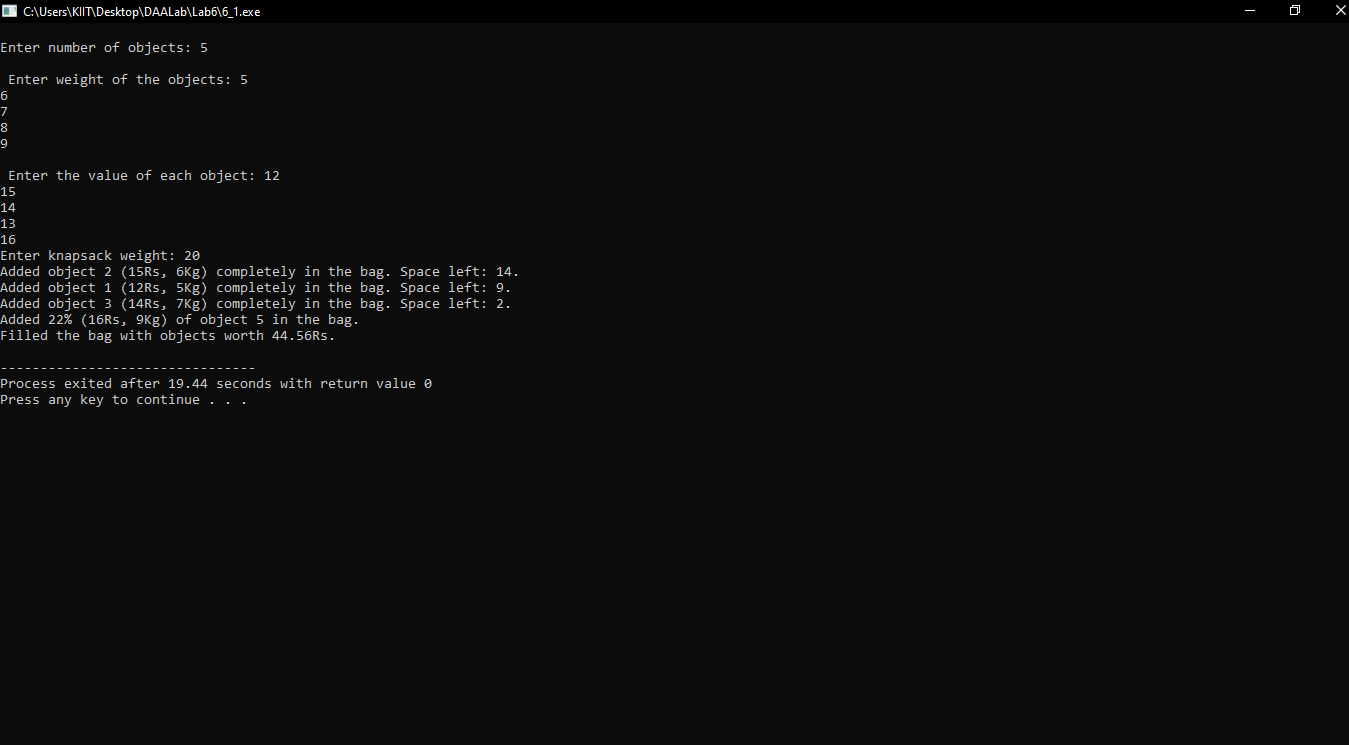
printf("Enter knapsack weight: ");

scanf("%d",&W);

knapsack(n,c,v,W);

return 0;

}



**/\*6.2** Write a program to implement the **activity-selection problem** stated as follows: You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time. Example: Consider the following 6 activities ( 0, 1, 2, 3, 4, 5). start[] = {1, 3, 0, 5, 8, 5}; finish[] = {2, 4, 6, 7, 9, 9}; The maximum set of activities that can be executed by a single person is {0, 1, 3, 4}.\*/

#include<stdio.h>

int main(){

int start[1000];

int finish[1000],n, k;

printf("\nEnter no of activities: ");

scanf("%d",&n);

printf("\nEnter start time: ");

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

scanf("%d",&start[k]);

printf("\nEnter finish time: ");

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

scanf("%d",&finish[k]);

int activities = n;

int i, j;

printf ("\nFollowing activities are selected \n\t");

i = 0;

printf("%d\t", i);

for (j = 1; j < activities; j++){

if (start[j] >= finish[i]){

printf ("%d ", j);

i = j;

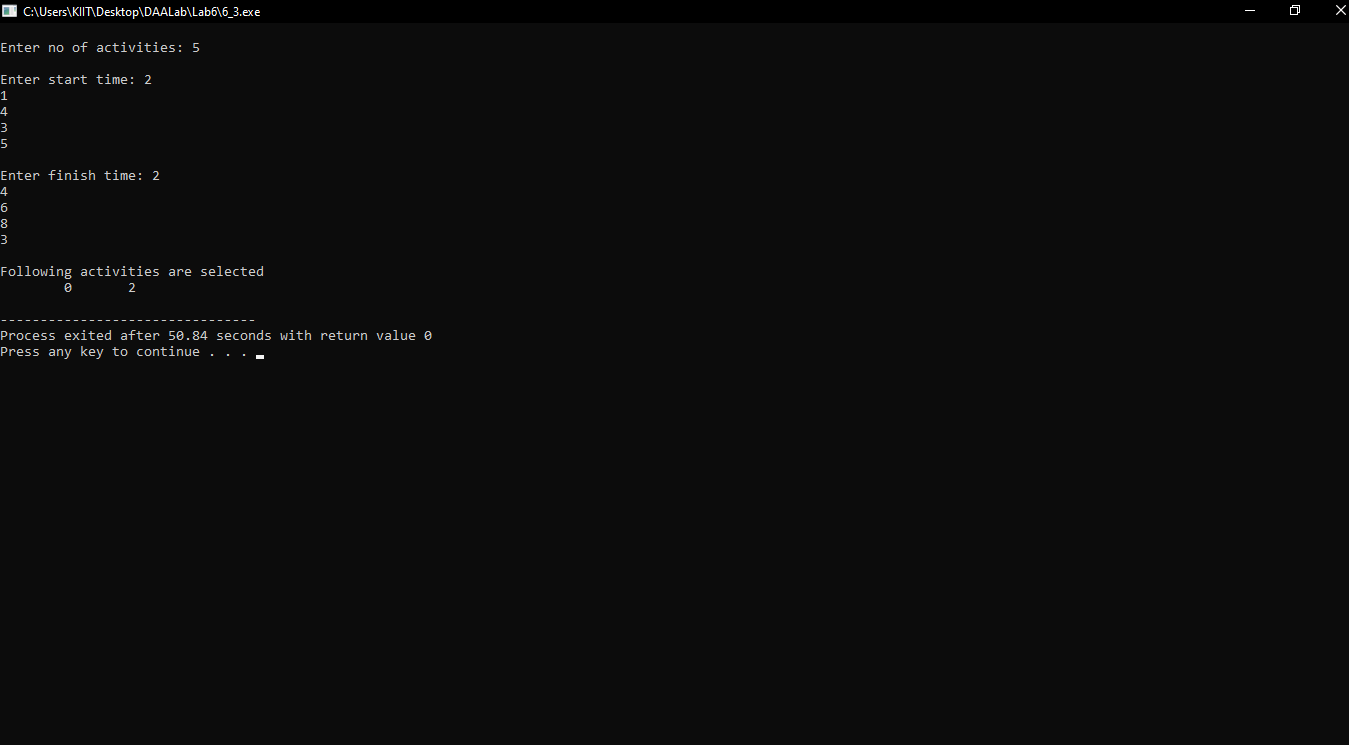
}

}

printf("\n");

return 0;

}



**/\*6.3** Write a program to implement the file or code compression using **Huffman‟s algorithm.\*/**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX\_TREE\_HT 100**

**struct MinHeapNode {**

**char data;**

**unsigned freq;**

**struct MinHeapNode \*left, \*right;**

**};**

**struct MinHeap {**

**unsigned size;**

**unsigned capacity;**

**struct MinHeapNode\*\* array;**

**};**

**struct MinHeapNode\* newNode(char data, unsigned freq)**

**{**

**struct MinHeapNode\* temp = (struct MinHeapNode\*)malloc**

**(sizeof(struct MinHeapNode));**

**temp->left = temp->right = NULL;**

**temp->data = data;**

**temp->freq = freq;**

**return temp;**

**}**

**struct MinHeap\* createMinHeap(unsigned capacity)**

**{**

**struct MinHeap\* minHeap = (struct MinHeap\*)malloc(sizeof(struct MinHeap));**

**minHeap->size = 0;**

**minHeap->capacity = capacity;**

**minHeap->array = (struct MinHeapNode\*\*)malloc(minHeap->**

**capacity \* sizeof(struct MinHeapNode\*));**

**return minHeap;**

**}**

**void swapMinHeapNode(struct MinHeapNode\*\* a,**

**struct MinHeapNode\*\* b)**

**{**

**struct MinHeapNode\* t = \*a;**

**\*a = \*b;**

**\*b = t;**

**}**

**void minHeapify(struct MinHeap\* minHeap, int idx)**

**{**

**int smallest = idx;**

**int left = 2 \* idx + 1;**

**int right = 2 \* idx + 2;**

**if (left < minHeap->size && minHeap->array[left]->**

**freq < minHeap->array[smallest]->freq)**

**smallest = left;**

**if (right < minHeap->size && minHeap->array[right]->**

**freq < minHeap->array[smallest]->freq)**

**smallest = right;**

**if (smallest != idx) {**

**swapMinHeapNode(&minHeap->array[smallest],**

**&minHeap->array[idx]);**

**minHeapify(minHeap, smallest);**

**}**

**}**

**int isSizeOne(struct MinHeap\* minHeap)**

**{**

**return (minHeap->size == 1);**

**}**

**struct MinHeapNode\* extractMin(struct MinHeap\* minHeap)**

**{**

**struct MinHeapNode\* temp = minHeap->array[0];**

**minHeap->array[0]**

**= minHeap->array[minHeap->size - 1];**

**--minHeap->size;**

**minHeapify(minHeap, 0);**

**return temp;**

**}**

**void insertMinHeap(struct MinHeap\* minHeap,**

**struct MinHeapNode\* minHeapNode)**

**{**

**++minHeap->size;**

**int i = minHeap->size - 1;**

**while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq) {**

**minHeap->array[i] = minHeap->array[(i - 1) / 2];**

**i = (i - 1) / 2;**

**}**

**minHeap->array[i] = minHeapNode;**

**}**

**void buildMinHeap(struct MinHeap\* minHeap)**

**{**

**int n = minHeap->size - 1;**

**int i;**

**for (i = (n - 1) / 2; i >= 0; --i)**

**minHeapify(minHeap, i);**

**}**

**void printArr(int arr[], int n)**

**{**

**int i;**

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

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

**printf("\n");**

**}**

**int isLeaf(struct MinHeapNode\* root)**

**{**

**return !(root->left) && !(root->right);**

**}**

**struct MinHeap\* createAndBuildMinHeap(char data[], int freq[], int size)**

**{**

**int i;**

**struct MinHeap\* minHeap = createMinHeap(size);**

**for (i = 0; i < size; ++i)**

**minHeap->array[i] = newNode(data[i], freq[i]);**

**minHeap->size = size;**

**buildMinHeap(minHeap);**

**return minHeap;**

**}**

**struct MinHeapNode\* buildHuffmanTree(char data[], int freq[], int size)**

**{**

**struct MinHeapNode \*left, \*right, \*top;**

**struct MinHeap\* minHeap = createAndBuildMinHeap(data, freq, size);**

**while (!isSizeOne(minHeap)) {**

**left = extractMin(minHeap);**

**right = extractMin(minHeap);**

**top = newNode('$', left->freq + right->freq);**

**top->left = left;**

**top->right = right;**

**insertMinHeap(minHeap, top);**

**}**

**return extractMin(minHeap);**

**}**

**void printCodes(struct MinHeapNode\* root, int arr[], int top)**

**{**

**if (root->left) {**

**arr[top] = 0;**

**printCodes(root->left, arr, top + 1);**

**}**

**if (root->right) {**

**arr[top] = 1;**

**printCodes(root->right, arr, top + 1);**

**}**

**if (isLeaf(root)) {**

**printf("%c: ", root->data);**

**printArr(arr, top);**

**}**

**}**

**void HuffmanCodes(char data[], int freq[], int size)**

**{**

**struct MinHeapNode\* root**

**= buildHuffmanTree(data, freq, size);**

**int arr[MAX\_TREE\_HT], top = 0;**

**printCodes(root, arr, top);**

**}**

**int main()**

**{**

**char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };**

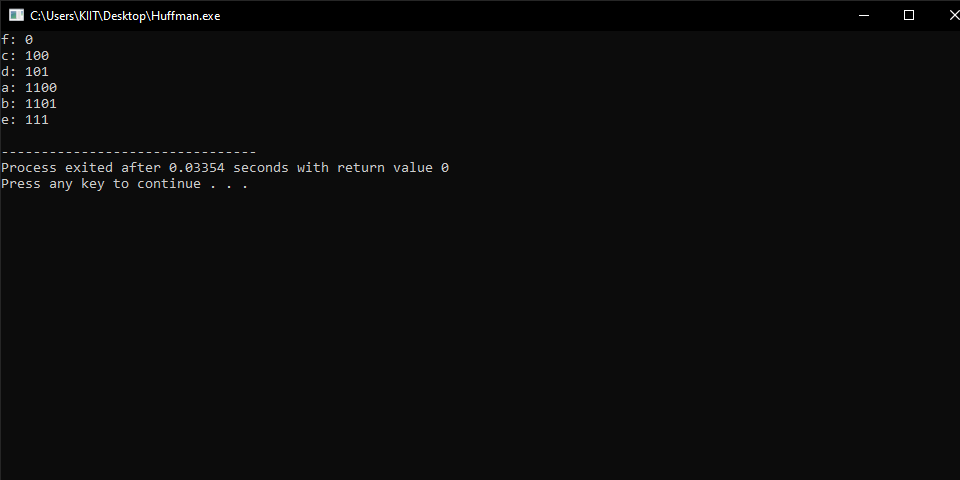
**int freq[] = { 5, 9, 12, 13, 16, 45 };**

**int size = sizeof(arr) / sizeof(arr[0]);**

**HuffmanCodes(arr, freq, size);**

**return 0;**

**}**

****

###### Submitted By:

###### Purvav Punyani

###### 1805231

###### CS-G1