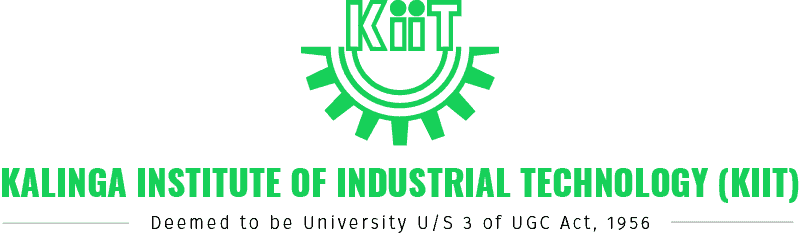
****

**NAME : SOHAM SAMANTA**

**ROLL NUMBER : 20051107**

**SECTION : CSE-14**

**DAA LAB ASSIGNMENT-3**

**3.1** Rewrite the program no-2.3 (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.

**Solution:**

//SOHAM SAMANTA CODES

#include<bits/stdc++.h>

using namespace std;

#define ll long long int

#define mod 1000000007

#define PI 3.1415926535897932384626433832

#define ss ios\_base::sync\_with\_stdio(false);cin.tie(NULL);

void swap(int arr[], int i, int j) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

static void insertion(int arr[], int n) { // 6 3 8 9 1

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

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

if (arr[j] < arr[j-1]) {

swap(arr, j, j-1);

} else {

break;

}

}

}

}

int32\_t main(){

ss;

int n;

cin>>n;

int arr[n];

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

cin>>arr[i];

}

cout<<"Before Sorting: "<<endl;

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

cout<<arr[i]<<" ";

}

insertion(arr,n);

cout<<endl;

cout<<"After Sorting: "<<endl;

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

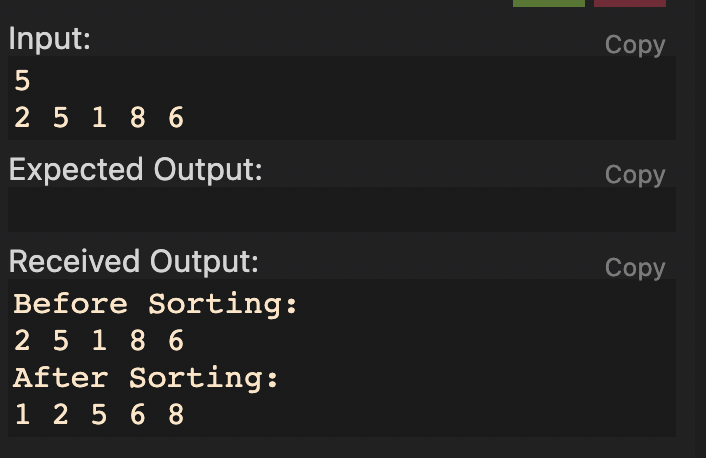
cout<<arr[i]<<" ";

}

return 0;

}

**Output:**



**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.

**Solution:**

//SOHAM SAMANTA CODES

#include<bits/stdc++.h>

using namespace std;

#define ll long long int

#define mod 1000000007

#define PI 3.1415926535897932384626433832

#define ss ios\_base::sync\_with\_stdio(false);cin.tie(NULL);

int find\_num1(int A[], int n)

{

for ( int i = 0; i<= n-2 ;i++)

{

int count = 1;

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

{

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

count = count + 1;

}

if ( count > n/2 )

return A[i];

}

return -1;

}

int findCandidate(int a[], int size)

{

int maj\_index = 0, count = 1;

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

if (a[maj\_index] == a[i])

count++;

else

count--;

if (count == 0) {

maj\_index = i;

count = 1;

}

}

return a[maj\_index];

}

/\* Function to check if the candidate

occurs more than n/2 times \*/

bool isMajority(int a[], int size, int cand)

{

int count = 0;

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

if (a[i] == cand)

count++;

if (count > size / 2)

return 1;

else

return 0;

}

/\* Function to print Majority Element \*/

void find\_num2(int a[], int size)

{

/\* Find the candidate for Majority\*/

int cand = findCandidate(a, size);

/\* Print the candidate if it is Majority\*/

if (isMajority(a, size, cand))

cout << " " << cand << " ";

else

cout << "No Majority Element";

}

int32\_t main(){

ss;

int n;

cout<<"Enter size";

cin>>n;

int a[n];

sort(a,a+n);

cout<<"Enter array"<<endl;

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

cin>>a[i];

}

cout<<"Number repeating more than n/2 times using O(N2) = "<<find\_num1(a,n)<<endl;

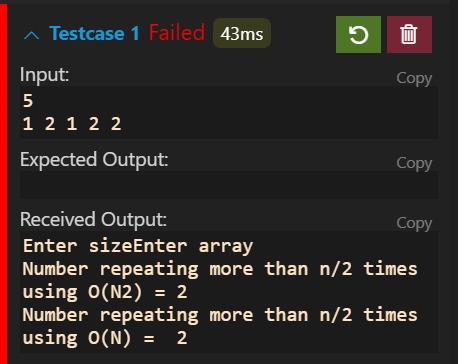
cout<<"Number repeating more than n/2 times using O(N) = ";

find\_num2(a,n);

return 0;

}

**Output:**

****

**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.

**Solution:**

//SOHAM SAMANTA CODES

#include<bits/stdc++.h>

using namespace std;

#define ll long long int

#define mod 1000000007

#define PI 3.1415926535897932384626433832

#define ss ios\_base::sync\_with\_stdio(false);cin.tie(NULL);

int sqrtOp(int n) {

int l = 0, r = n/2, ans = 0, m = 0;

while(l <= r){

m = (l+r) / 2;

if(m\*m == n)

return m;

else if(m\*m < n){

l = m+1;

ans = l;

} else{

r = m-1;

ans = r;

}

}

return ans;

}

int32\_t main(){

ss;

int n;

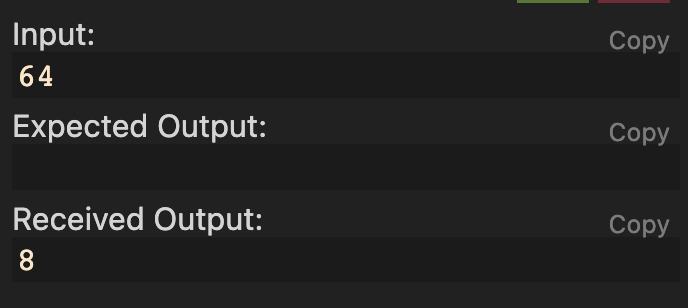
cin>>n;

cout<<sqrtOp(n);

return 0;

}

**Output:**



**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 programby 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. a. For example, for the array A[]={1, 3, 7, 2, 1, 5, 1, 2, 4, 6, 3}. b. This maximum sum is S = S3-8 = 2+( 1)+5+( 1)+( 2)+4 = 7.

**Solution:**

//SOHAM SAMANTA CODES

#include<bits/stdc++.h>

using namespace std;

#define ll long long int

#define mod 1000000007

#define PI 3.1415926535897932384626433832

#define ss ios\_base::sync\_with\_stdio(false);cin.tie(NULL);

int maxSubsum(int arr[], int n){

int current\_sum;

int max\_sum = INT\_MIN;

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

{

current\_sum = 0;

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

{

current\_sum += arr[j];

max\_sum = max(max\_sum, current\_sum);

}

}

return max(max\_sum, 0);

}

int32\_t main(){

ss;

int n;

cin >> n;

int arr[n];

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

cin >> arr[i];

}

cout << maxSubsum(arr, n);

return 0;

}

**Output:**

