BCSE204P- Design and Analysis of Algorithms Lab

E1-Slot-L43-L44 Lab

In-Lab Practice (IPS) Exercise-1

Name: R. Muhmmad Abrar

Register no: 21BRS1713

Any one question Output:

2. Design related problems

   (a) Translate the Insertion-sort algorithm discussed in the class into a program which takes n numbers (real or integers). Inputs for the program is n and the n numbers.

Insertion Sort > insertion-2a.cpp > ...

```cpp
#include <iostream>
using namespace std;

void printArray(int array[], int size) {
  for (int i = 0; i < size; i++) {
    cout << array[i] << " ";
  }
  cout << endl;
}

void insertionSort(int array[], int size) {
    for (int step = 1; step < size; step++) {
        int key = array[step];
        int j = step - 1;
        while (key < array[j] && j >= 0) {
        array[j + 1] = array[j];
        --j;
        }
        array[j + 1] = key;
    }
}

int main() {
    int data[] = {5,7,4,3,6,2};
    cout << "Given array: \n";
    int size = sizeof(data) / sizeof(data[0]);
    printArray(data,size);
    insertionSort(data, size);
    cout << "Insertion sort in ascending order:\n";
    printArray(data, size);
}
```

```
PS D:\SEMESTER 4\DAA\LAB\IPS FILES\Insertion Sort> cd "d:\SEMESTER 4\DAA\LAB\IPS FILES\Insertion Sort"
PS D:\SEMESTER 4\DAA\LAB\IPS FILES\Insertion Sort> & .\"insertion-2a.exe"
Given array:
5 7 4 3 6 2
Insertion sort in ascending order:
2 3 4 5 6 7
PS D:\SEMESTER 4\DAA\LAB\IPS FILES\Insertion Sort>
```

(b) Given a sequence of n numbers (real or integers) and a number k (k is one among the n numbers), write an algorithm and the corresponding code to compute the position of k if the given n numbers are arranged in an increasing order, using insertion-sort. If the $2, -1, 3, 0, 7$ and 3 are the input , your program should output 4 since 3 will be in the fourth position (starting from 1), in the sorted (increasing) order. You are expected to code the problem two different ways, say, c_1 , c_2 using two different approaches. Decide whether c_1 is efficient or c_2 is efficient based on the running time $T(n)$ of the resepctive codes.

Approach c1 [LINEAR SEARCH]

Insertion Sort > insertion-2b-linear.cpp > ...

```cpp
#include<iostream>
#include<ctime>
using namespace std;

int main(){

    int n,k,ele;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++){
        cin>>ele;
        arr[i]=ele;
    }
    cin>>k;
    clock_t st=clock();
    for(int i=1;i<n;i++){
        int key;
        int j=i-1;
        key=arr[i];
        while((j>=0)&&(arr[j]>key)){
            arr[j+1]=arr[j];
            j=j-1;
        }
        arr[j+1]=key;
    }

    for(int i=0;i<n;i++){
        if(arr[i]==k){
            cout<<"Element position is"<<i+1<<'\n';
        }
    }

    double time1=(double)(clock() - st)/CLOCKS_PER_SEC;
    cout<<"Time taken is "<<time1<<endl;
}
```

Approach c2 [BINARY SEARCH]

Insertion Sort > insertion-2b-binary.cpp > ...

```cpp
#include<iostream>
#include<ctime>
using namespace std;
int bSearch(int arr[], int l, int r, int x){
    if (r >= l) {
        int mid = l + (r - l) / 2;

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

        if (arr[mid] > x)
            return bSearch(arr, l, mid - 1, x);

        return bSearch(arr, mid + 1, r, x);
    }
    return -1;
}
int main(){
    int n,k,ele;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++){
        cin>>ele;
        arr[i]=ele;
    }
    cin>>k;
    clock_t st=clock();
    for(int i=1;i<n;i++){
        int key;
        int j=i-1;
        key=arr[i];
        while((j>=0)&&(arr[j]>key)){
            arr[j+1]=arr[j];
            j=j-1;
        }
        arr[j+1]=key;
    }
    int index = bSearch(arr,0,n-1,k);
    if(index != -1){
        cout<<"Element not found.";
    }
    else{
        cout<<"Element position is "<<index+1;
    }
    double time1=(double)(clock() - st)/CLOCKS_PER_SEC;
    cout<<"Time taken is "<<time1<<endl;
}
```

Outputs:

C1[linear search]

```
┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Insertion Sort]
└─$ g++ insertion-2b-linear.cpp

┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Insertion Sort]
└─$ ./a.out
5
12
11
5
6
3
5
Element position is2
Time taken is 5.1e-05
```

C2[binary search]

```
┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Insertion Sort]
└─$ g++ insertion-2b-binary.cpp

┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Insertion Sort]
└─$ ./a.out
5
12
11
5
6
3
5
Element position is 2
Time taken is 3.6e-05
```

For the same inputs, c2 method seems to provide better results than c1 method.
Thus, Binary search method is more efficient in terms of time complexity.

(c) All the alphabets(lower case) of english language $a, b, c, ..., y, z$ are assigned values $1, 2, 3, ..., 25, 26$. Given a sequence of n symbols from english alphabet (only lower case), write an insertion-sort based algorithm to arrange the given n symbols, in an increasing order of their values. . You are expected to code the problem two different ways, say, c_1 , c_2 using two different approaches. Decide whether c_1 is efficient or c_2 is efficient based on the running time $T(n)$ of the resepctive codes.

CODE

Approach c1

Insertion Sort > insertion-alpha-sort.cpp > main()

```cpp
#include <iostream>
#include <cstring>
#include <string>
#include <ctime>
using namespace std;

void printArray(char array[], int size) {
  for (int i = 0; i < size; i++) {
    cout << array[i] << " ";
  }
  cout << endl;
}

void insertionSort(char array[], int size) {
    for (int step = 1; step < size; step++) {
        int key = array[step];
        int j = step - 1;
        while (key < array[j] && j >= 0) {
        array[j + 1] = array[j];
        --j;
        }
        array[j + 1] = key;
    }
}

int main() {
    string s;
    cin>>s;
    int len_st = s.length();
    char* data = new char[len_st + 1];
    strcpy(data, s.c_str());
    int size = sizeof(data) / sizeof(data[0]);
    clock_t timeStart = clock();
    insertionSort(data, size);
    float time=(float)(clock() - timeStart)/CLOCKS_PER_SEC;
    cout<<"The number of inputs: "<<len_st<<endl;
    cout << "Sorted string in ascending order:\n";
    printArray(data, size);
    cout<<"Time Taken is "<<time<<"s";
}
```

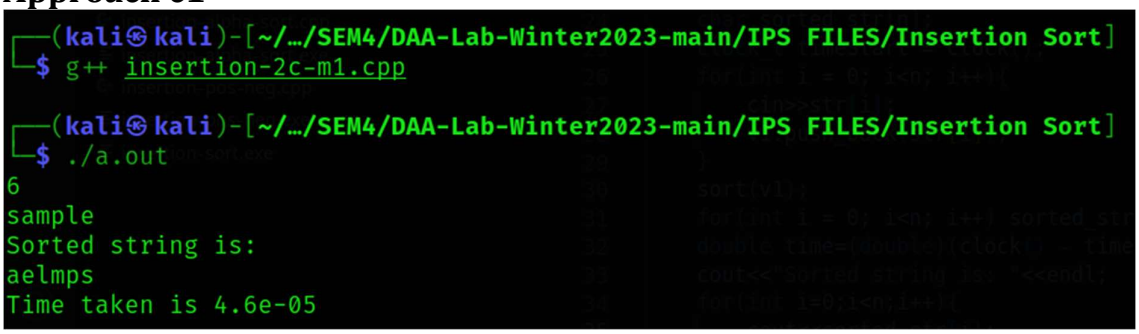
Approach c2

insertion-2c-m2.cpp > ...

```cpp
#include <iostream>
#include<ctime>
using namespace std;
int main(){
    int n;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++){
        char a;
        cin>>a;
        int b=a;
        arr[i]=b-96;
    }
    clock_t timeStart = clock();
    for(int j=1;j<n;j++){
        int key=arr[j];
        int i=j-1;
        while((i>=0)&&key<arr[i]){
            arr[i+1]=arr[i];
            i-=1;
        }
        arr[i+1]=key;
    }
    cout<<"Sorted string in ascending order:"<<'\n';
    for(int i=0;i<n;i++){
        char a=(arr[i]+96);
        cout<<(a)<<'\n';
    }
    double time=(double)(clock() - timeStart)/CLOCKS_PER_SEC;
    cout<<"\nTime taken is "<<time<<endl;
}
```

OUTPUT:

Approach c1



Approach c2

By looking at the time taken by the 2 approaches, I feel that my approach c2 is taking lesser time than approach c1 for the same input. So, approach 2 is more efficient.

(d) Given a sequence of n numbers (real or integers), write an algorithm and the corresponding code to arrange the given n numbers are arranged in such a way that all the negative numbers (if any) are arranged in a descending order and all the positive numbers are arranged in an increasing order with zero (if it is in the input) appearing between the smallest negative number and the smallest positive number. If $7, 3, 2, 4$ the output should be $2, 3, 4, 7$. If $-7, -3, 2, 4$ the output should be $-3, -7, 2, 4$ should be the output. If $7, 3, -1, 0, 2, 4$ the output should be $-1, 0, 3, 4, 7$.

CODE

Insertion Sort > insertion-2d-m1.cpp > ...

```cpp
#include<iostream>
#include<vector>
#include<ctime>
using namespace std;
int main(){
    clock_t tStart = clock();
    vector<int> arr1;
    vector<int> arr2;
    int n,ele;
    cin>>n;
    for(int i=0;i<n;i++){
        cin>>ele;
        if(ele<0){
            arr1.push_back(ele);
        }
        else{
            arr2.push_back(ele);
        }
    }
    for(int j=1;j<arr1.size();j++){
        int key=arr1[j];
        int i=j-1;
        while(key>arr1[i]&&(i>=0)){
            arr1[i+1]=arr1[i];
            i=i-1;
        }
        arr1[i+1]=key;
    }
    for(int j=1;j<arr2.size();j++){
        int key = arr2[j];
        int i = j-1;
        while((i>=0)&&(arr2[i]>key))
        {
            arr2[i+1] = arr2[i];
            i = i-1;
        }
        arr2[i+1] = key;
    }
double time1=(double)(clock() - tStart)/CLOCKS_PER_SEC;
    cout<<"Time taken is "<<time1<<endl;
   for(int i=0;i<arr1.size();i++){
    cout<<arr1[i];
   }
   for(int i=0;i<arr2.size();i++){
    cout<<arr2[i];
   }
}
```

OUTPUT

```
┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Insertion Sort]
└─$ g++ insertion-2d-m1.cpp

┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Insertion Sort]
└─$ ./a.out
4
-4
-2
3
1
Time taken is 0.000145s
-2 -4 1 3
```

(e) Given n points $P_1, P_2, ..., P_n$ with the coordinates $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ respectively, write an insertion-sort based algorithm and the corresponding code to arrange the points in an increasing order of the distance of the point from the origin (with $(0, 0)$ as the coordinates. Distance between any two points (a_1, b_1) and (a_2, b_2) is $\sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2}$. Input for the code is : n and the coordinates of the n points entered with $x-$coordinate first and then the $y-$coordinate.

Insertion Sort > insertion-2e.cpp > ...

```cpp
#include<iostream>
#include<cmath>
using namespace std;

void input(int arr[],int n){
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
}

void print(int A[],int B[],int n){
    for(int u=0;u<n;u++){
        cout<<A[u]<<","<<B[u]<<endl;
    }
    cout<<endl;
}

void dist(int A[],int B[],int D[],int n ){
    for(int i=0;i<n;i++){
        D[i]=sqrt(pow(A[i],2)+pow(B[i],2)) ;
    }
}

void insertion_sort(int arr[],int A[],int B[],int n){
    int k;
    for(int u=1;u<n;u++){
        int key=arr[u];
        int k2=A[u];
        int k3=B[u];
        int v=u-1;
        while(key<arr[v] && v>=0){
            if(arr[v]==k)
            {
                int pos=v;
            }
            arr[v+1]=arr[v];
            A[v+1]=A[v];
            B[v+1]=B[v];
            --v;
        }
        arr[v+1]=key;
        A[v+1]=k2;
        B[v+1]=k3;
    }
}

int main(){
    int n;
    cout<<"Enter the number of elements"<<endl;
    cin>>n;
    int X[n],Y[n],D[n];
    input(X,n);
    input(Y,n);
    dist(X,Y,D,n);
    insertion_sort(D,X,Y,n);
    print(X,Y,n);
}
```

OUTPUT

```
┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Insertion Sort]
└─$ g++ insertion-2e.cpp

┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Insertion Sort]
└─$ ./a.out
Enter the number of elements
4
Enter X Coordinates:
1
3
5
7
Enter Y Coordinates:
2
4
6
8
1,2
3,4
5,6
7,8
```

```
┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Insertion Sort]
└─$ ./a.out
Enter the number of elements
5
Enter X Coordinates:
1
7
3
5
4
Enter Y Coordinates:
2
8
4
6
0
1,2
4,0
3,4
5,6
7,8
```