

CSE2012
DAA LAB
CSE 2012- Design and Analysis of Algorithms
Practice Problem Sheet-1

Name: Preyash

Registration Number: 20BPS1022

Q1. Consider arranging n numbers stored in an array A , by first finding the smallest element of A and exchanging it with the element in $A[1]$. Then find the second smallest element of A and exchange it with $A[2]$ continue in this manner for the first $(n-1)$ elements. Based on the approach described above, write an algorithm for arranging the given n numbers in an increasing order of the numbers. Compute the best-case running time, worst-case running time and the average-case running time of the algorithm. Compare this algorithm with that of the insertion-sort algorithm, based on the respective $T(n)$. Based on your analysis, conclude which algorithm performs better for which type of inputs etc.

Algorithm:

start

selectionSort(array, size)

(size - 1) repetitions

make the initial unsorted element the smallest

for each of the items that haven't been sorted

if element currentMinimum is true

create a new minimum element

exchange the initial unsorted position with the minimum

end of selectionSort

Code:

```
#include <iostream>
#include <vector>
using namespace std;
void swap(int *a, int *b)
{
    int temp = *a;
    *b = temp;
}
int main()
{
    int n;
    cout << "Enter number of elements in the array: " << endl;
    cin >> n;
    vector<int> vec1;
    for (int i = 0; i < n; i++)
    {
        int a;
        cin >> a;
        vec1.push_back(a);
    }
    cout << "The vector before sorting the array: " << endl;
    for (int i = 0; i < n; i++)
    {
        cout << vec1[i] << " ";
    }
    cout << endl;
    for (int i = 0; i < n - 1; i++)
    {
        int index = i;
        for (int j = i + 1; j < n; j++)
        {
            if (vec1[j] < vec1[index])
                index = j;
        }
        swap(vec1[i], vec1[index]);
    }
    cout << "The vector after sorting the array: " << endl;
    for (int i = 0; i < n; i++)
    {
        cout << vec1[i] << " ";
    }
    return 0;
}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

The vector before sorting the array:
1 6 3 5 2
The vector after sorting the array:
1 2 3 5 6
PS E:\Coding\C++\DAA_LABS\PPS1> █
```

Analysis of Algorithm:

Best Case Time Complexity: $O(N)$

Average Case Time Complexity: $O(N^2)$

Worst Case Time Complexity: $O(N^2)$

Insertion Sort Best Time Complexity: $O(N^2)$

Insertion Sort Average Time Complexity: $O(N^2)$

Insertion Sort Worst Time Complexity: $O(N^2)$

Insertion Sort is better than Selection sort for wide range of input because the number of comparisons in case of Insertion Sort is less than selection Sort.

Q2. Consider sorting (arranging in an increasing order) n numbers stored in an array A , by repeatedly by swapping the adjacent elements that are not in an increasing order. If $A[i]$ and $A[i + 1]$ are such that $A[i] > A[i + 1]$, then $A[i]$ and $A[i + 1]$ shall be swapped. Based on the approach described above, write an algorithm for arranging the given n numbers in an increasing order of the numbers. Compute the best-case running time, worst-case running time and the average-case running time of the algorithm. Compare this algorithm with that of the insertion-sort algorithm and conclude which algorithm performs better for which type of inputs etc.

Algorithm:

```
start
call BubbleSort(list)
for all elements of list
    if list[i] > list[i+1]
        swap(list[i], list[i+1])
    end if
end for
return list
end BubbleSort
```

Code:

```
#include <bits/stdc++.h>
using namespace std;
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
void bubbleSort(vector<int> &arr)
{
    int i, j;
    int n = arr.size();
    for (i = 0; i < n-1; i++){
        for (j = 0; j < n-i-1; j++){
            if (arr[j] > arr[j+1]){
                swap(&arr[j], &arr[j+1]);
            }
        }
    }
}
```

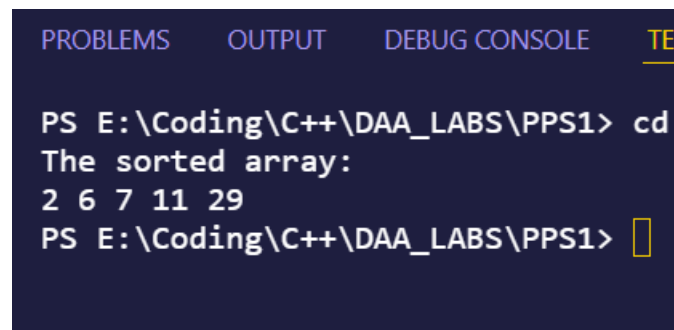
```

    }
}

int main()
{
    vector<int> arr = {02, 11, 29, 06, 07};
    bubbleSort(arr);
    cout<<"The sorted array: \n";
    for(auto i: arr)cout << i << " ";
    return 0;
}

```

Output:



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TEST
PS E:\Coding\C++\DAA_LABS\PPS1> cd
The sorted array:
2 6 7 11 29
PS E:\Coding\C++\DAA_LABS\PPS1> 

```

Analysis of Algorithm:

Best Case Time Complexity: $O(N)$

Average Case Time Complexity: $O(N^2)$

Worst Case Time Complexity: $O(N^2)$

Insertion Sort Best Time Complexity: $O(N)$

Insertion Sort Average Time Complexity: $O(N^2)$

Insertion Sort Worst Time Complexity: $O(N^2)$

Insertion Sort is better than Bubble sort for wide range of input because the number of comparison in case of Insertion Sort is less than bubble Sort.

Q3. Consider an array of $A[1, 2, 3, \dots, n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then we call the pair (i, j) as an inversion of A . For example, the five inversions in the array $A :< 2, 3, 8, 6, 1 >$ are $(1, 5), (2, 5), (3, 4), (3, 5), (4, 5)$. Given an array of numbers, design the pseudocode for computing the number of inversions in the array ('inversion algorithm'). Analyse the pseudocode of the inversion algorithm. Compare the running-time of insertion-sort and the running-time of inversion algorithm. Is there anything similar between the 'insertion-sort algorithm' and the 'inversion algorithm'?

Algorithm:

InverstionAlgo(array, size)

 Initialise a vector of pair of integer

 for each of the unsorted elements

 if $i < arr[j]$

 makepair of i and j and push it in vector

 print the contents of pair + 1

 end inversionAlgo

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    vector<int> arr = {2,3,8,6,1};
    vector<pair<int,int>> ans;
    for(int i=0; i<arr.size(); i++){
        for(int j=i+1; j<arr.size(); j++){
            if(i<j && arr[i]>arr[j]){
                ans.push_back(make_pair(i, j));
            }
            else {
                continue;
            }
        }
    }
    for(auto i:ans){
        cout << i.first + 1 << " " << i.second + 1 << endl;
    }
    return 0;
}
```

Output:

```
PS E:\Coding\C++\DAA_LABS\PPS1> cd "e:
1 5
2 5
3 4
3 5
4 5
PS E:\Coding\C++\DAA_LABS\PPS1> 
```

Analysis of Algorithm:

Time Complexity of Inversion Algorithm is: $O(N^2)$

Time Complexity of Insertion Sort is: $O(N^2)$

Q4. A nonnegative integer n is given. Design an algorithm to count all the solutions of the inequality $x^2 + y^2 < n$, where x and y are non-negative integers. The pseudocode should not use any operations with real numbers (square roots, etc.). Analyze the pseudocode designed by you.

Algorithm:

```
selectionSort(n)
initialize ans as 0
for loop till  $x^2 < n$ 
  for loop till  $x^2 + y^2 < n$ 
    increment ans
  return ans
end selectionSort
```

Code:

```
#include <iostream>
using namespace std;
int Sol(int n)
{
    int ans = 0;
    for (int x = 0; x*x < n; x++){
        for (int y = 0; x*x + y*y < n; y++){
            ans++;
        }
    }
}
```

```

        return ans;
    }
    int main()
    {
        int n;
        cout<<"Enter the value of n: ";
        cin>>n;
        cout << "The number of inequality solutions are: "<< Sol(n) << endl;
        return 0;
    }

```

Output:

```

PS E:\Coding\C++\DAA_LABS\PPS1> cd "e:\Coding\C-
unnerFile } ; if ($?) { .\tempCodeRunnerFile }
Enter the value of n: 5
The number of inequality solutions are: 6
PS E:\Coding\C++\DAA_LABS\PPS1>

```

Analysis of Algorithm:

Time Complexity: $O(n^2)$

Space Complexity: $O(n)$

Q5. Given two numbers a and b, we usually say that a is less than or equal to b (denoted by $a \leq b$) if the value of a is smaller than or equal to b. Here, we define a new relation $a \leq b$ in a different way, as follows. a is said to be less than or equal to b (denoted by $a' \leq b'$) if the last digit of a is less than or equal to the last digit of b. For example, 27 is less than 19, since 7 is less than 9. 29 and 39 are said to be equal. 38 is greater than 102. Given n numbers, design a pseudocode to arrange the given numbers in a decreasing order, as per the order defined above. For example: Given the numbers 27,39,1, 55, 124. Numbers in the decreasing order are : 39, 27,55,124,1 Also, analyze your algorithm with all the steps required.

Algorithm:

Step 1: Start

Step 2: Store the elements that has to be sorted in an array. Lets call it arr

Step 3: Divide the remainders of each element in separate array called arrange

Step 4: Now we apply insertion sort on the array arrange and simultaneously change the positions of elements in arr while changing position of elements in arrange

Step 5: End

Code:

```
#include<iostream>
using namespace std;
void sort(int a[],int arr[],int l){
    int key,j,order;
    for (int i=1;i<l;i++){
        key=a[i];
        order=arr[i];
        j=i-1;
        while(j>=0 && a[j]<key){
            a[j+1]=a[j];
            arr[j+1]=arr[j];
            j--;
        }
        a[j+1]=key;
        arr[j+1]=order;
    }
}
void display(int a[],int l){
    for (int i=0;i<l;i++){
        cout<<a[i]<<endl;
    }
}
void printArray(int arr[],int n){
    for (int i=0;i<n;i++)
```

```

    {
        cout<<arr[i]<<endl;
    }
}
int main()
{
    int i,arr[]={27,39,1, 55, 124},arrange[20];
    int size=sizeof(arr)/sizeof(arr[0]);
    for(i=0;i<size;i++)
    {
        arrange[i]=arr[i]%10;
    }
    sort(arrange,arr,size);
    printArray(arr,size);
}

```

Output:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
if ($?) { .\tempCodeRunnerFile }
39
27
55
124
1
PS E:\Coding\C++\DAA_LABS\PPS1>

```

Analysis of Algorithm:

Insertion sort is the logic utilised here. In insertion sort, an element from a sorted array is taken and inserted into an unsorted array at the appropriate point. We do this by comparing the element from the sorted array to all of the items in the unsorted array and then inserting it in the proper place. So, in the best-case scenario, the array is already sorted, and the time complexity is $O(n)$. The temporal complexity would be $O(n^2)$ in the case of reverse sorted input.