Name : Sonal Dilip Gholap

UID : 2021600020

Branch : CSE-AIML

Batch : B

Experiment : 1

Aim : Experiment on finding the running time of an algorithm (time complexity analysis)

- **Objectives :**

    1. To understand the algorithms of insertion sort and selection sort.
    2. To derive the best case and worst-case time complexities of insertion sort and selection sort.
    3. To sort the given arrays using both insertion and selection sort and present output along with a few comparisons and exchanges.
    4. To compare the time complexities of insertion sort and selection sort.

- **Theory**

1. Insertion Sort

    - Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands.
    - The array is virtually split into a sorted and an unsorted part.
    - Values from the unsorted part are picked and placed at the correct position in the sorted part.
    - Insertion sort is efficient for small data sets.
    - It is adaptive in nature i.e. it is appropriate for data sets that are already substantially sorted.
    - It is an in-place comparison sorting algorithm.
    - Best-case time complexity of insertion sort is **O(n)**
    - Average case time complexity of insertion sort is **$O(n^2)$**
    - Worst-case time complexity of insertion sort is **$O(n^2)$**
    - The space complexity of insertion sort is **O(1)**

2. Selection Sort

- Selection sort is a simple sorting algorithm that works by repeatedly selecting the smallest element from the unsorted portion of the list and moving it to the sorted portion of the list.
- In every iteration of the selection sort, the minimum element from the unsorted subarray is picked and moved to the beginning of unsorted subarray.
- After every iteration sorted subarray size increase by one and unsorted subarray size decrease by one.
- Sorted part is placed at the left, while the unsorted part is placed at the right.
- It is an in-place sorting algorithm.
- It is not suitable for large data sets.
- The best-case time complexity of selection sort is **$O(n^2)$**
- The average and worst-case complexity of selection sort is **$O(n^2)$**
- The space complexity of selection sort is **O(1)**

- **<u>Algorithm for Insertion Sort</u>**

```
for i = 2 to n do
   key = A[i];
   j = i – 1;
   while j > 0 and A[j] > key do
        A[j+1] = A[j];
        j = j – 1;
   A[j+1] = key;
```

- **<u>Algorithm for Selection Sort</u>**

```
for i = 1 to n - 1 do
   min = i;
   for j = i – 1 to n do
       if A[j] < A[min] then
           min = j;
   swap A[i] , A[min];
```

- **<u>Derivation of Time Complexity</u>**

    1. Insertion Sort

* Time Complexity Derivations

I) Insertion Sort (A)

```
1. for i=2 to length(A)
2. do key ← A[i]
3. // insert A[i] into sorted sequence A[i, . . . j-1]
4. j=i-1
5. while j>0 and A[j] > key do
6. A[j+1] = A[j]
7. j=j-1
8. A[j+1] = key
```

Statement	Cost	Time	Best Case Time	Worst Case Time
1.	c_1	n	n	n
2.	c_2	$n-1$	$n-1$	$n-1$
3.	c_3	0	0	0
4.	c_4	$n-1$	$n-1$	$n-1$
5.	c_5	$\sum_{j=2}^{n} t_j$	$n-1$	$\frac{n(n+1)}{2} - 1$
6.	c_6	$\sum_{j=2}^{n} t_{j-1}$	0	$\frac{n(n-1)}{2}$
7.	c_7	$\sum_{j=2}^{n} t_{j-1}$	0	$\frac{n(n-1)}{2}$
8.	c_8	$n-1$	$n-1$	$n-1$

FABER-CASTELL

Date ____

Page No. ____

A) Best Case Time = $c_1 n + c_2(n-1) + c_3(0) + c_4(n-1) + c_5(n-1) + c_6(0) + c_7(0) + c_8(n-1)$

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n + (-c_2 - c_4 - c_5 - c_8)$$

$$T(n) = A(n) + B$$

$\therefore$ Best Case Time Complexity = $O(n)$

B) Worst Case Time = $c_1 n + c_2(n-1) + c_3(0) + c_4(n-1) + c_5\left[\frac{n(n+1)}{2} - 1\right] + c_6\left[\frac{n(n-1)}{2}\right] + c_7\left[\frac{n(n-1)}{2}\right] + c_8(n-1)$

$$T(n) = c_1 n + c_2 n - c_2 + c_4 n - c_4 + c_5\left[\frac{n(n+1)}{2} - 1\right] + c_6\left[\frac{n(n-1)}{2}\right] + c_7\left[\frac{n^2}{2} - \frac{n}{2}\right] + c_8 n - c_8$$

$$T(n) = \left(c_5 + c_6 + c_7\right)\frac{n^2}{2} + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n + (-c_2 - c_4 - c_5 - c_8)$$

$$T(n) = A(n^2) + B(n) + C$$

$\therefore$ Worst Case Time Complexity = $O(n^2)$

2. Selection Sort

II) Selection Sort (A)

1. for $i = 1 \rightarrow n-1$
2. do $min \leftarrow i$
3. for $j = i+1 \rightarrow n$, do
4. if $A[j] < A[min]$, then
5. $min \leftarrow j$
6. swap $A[j]$, $A[min]$

Statement	Cost	Time	Best Case Time	Worst Case Time
1.	c_1	n	n	n
2.	c_2	$n-1$	$n-1$	$n-1$
3.	c_3	n	n	n
4.	c_4	$\sum_{j=1}^{n-1} t_j$	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$
5.	c_5	$\sum_{j=1}^{n-1} t_j$	0	$\frac{n(n-1)}{2}$
6.	c_6	$n-1$	0	$n-1$

A) Best Case Time $= c_1 n + c_2(n-1) + c_3(n) + c_4\left[\frac{n(n-1)}{2}\right] + c_5(0) + c_6(0)$

$$= \frac{c_4}{2}(n^2) + \left(c_1 + c_2 + c_3 - \frac{c_4}{2}\right)n + \left(-c_2 - \frac{c_4}{2}\right)$$

$$= An^2 + Bn + C$$

$\therefore$ Best Case Time Complexity $= O(n^2)$

FABER-CASTELL

Date

Page No.

B) Worst Case Time $= c_1 n + c_2(n-1) + c_3(n) + c_4\left[\frac{n(n-1)}{2}\right]$

$$+ c_5\left[\frac{n(n-1)}{2}\right] + c_6(n-1)$$

$$T(n) = \left(\frac{c_4}{2} + \frac{c_5}{2}\right)n^2 + \left(c_1 + c_2 + c_3 - \frac{c_4}{2} - \frac{c_5}{2} + c_6\right)n$$

$$+\left(-c_2 - \frac{c_4}{2} - \frac{c_5}{2} - c_6\right)$$

$$T(n) = An^2 + Bn + C$$

$\therefore$ Worst Case Time Complexity $= O(n^2)$

Code :

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

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

    int temp = *a;
    *a = *b;
    *b = temp;
}

void insertionsort(int a[],int n){
```

```
    int key;
    int swaps = 0;
    int comps = 0;
    for(int j=1;j<n;j++){
    key = a[j];
    int i = j-1;
    //comps++;
    while(i>=0 && a[i]>key){
        a[i+1]=a[i];
        swaps++;
        comps++;
        //a[i] = key;
        i--;
    }
    a[i+1]=key;

    if (a[i]<=key){
        comps++;
    }


    }

    cout<<endl;
    cout<<"Sorted array : ";

    for(int i=0;i<n;i++){
        cout<<a[i]<<" ";
    }
    cout<<endl<<"Number of Comparisons : "<<comps<<endl;
    cout<<"Number of Swaps : "<<swaps<<endl;

}

void selectionsort(int a[],int n){

    int minI;
    int swaps = 0;
    int comps = 0;

    for(int i=0;i<n-1;i++){
        minI = i;
        for(int j=i+1;j<n;j++){
            comps++;
            if(a[j]<a[minI]){
                minI = j;
            }
        }
        //swap a[i] and a[minI]
```

```
        if(minI!=i){
            swap(&a[i],&a[minI]);
            swaps++;
        }



    }

    cout<<endl;
    cout<<"Sorted array : ";

    for(int i=0;i<n;i++){
        cout<<a[i]<<" ";
    }
    cout<<endl<<"Number of Comparisons : "<<comps<<endl;
    cout<<"Number of Swaps : "<<swaps<<endl;
}

int main(){

int uid = 20;
int best[10],worst[10];

for(int i=0;i<10;i++){
    best[i] = (uid+((uid+1)*i));
    worst[i] = (uid+((uid+1)*(9-i)));
}

int random[10] = {2,12,76,5,29,7,88,102,35,234};

cout<<"Enter choice:"<<endl<<"1. Insertion Sort"<<endl<<"2. Selection Sort"<<endl;
int choice;
cin>>choice;

if(choice==1){
    cout<<"Best Case: ";
    for(int i=0;i<10;i++){
        cout<<best[i]<<" ";
    }
    insertionsort(best,10);
    cout<<endl<<endl;

    cout<<"Worst Case: ";
    for(int i=0;i<10;i++){
        cout<<worst[i]<<" ";
    }
    insertionsort(worst,10);
    cout<<endl<<endl;
```

```
    cout<<"Random Case: ";
    for(int i=0;i<10;i++){
        cout<<random[i]<<" ";
    }
    insertionsort(random,10);
    cout<<endl<<endl;
}

else if(choice==2){
    cout<<"Best Case: ";
    for(int i=0;i<10;i++){
        cout<<best[i]<<" ";
    }
    selectionsort(best,10);
    cout<<endl<<endl;

    cout<<"Worst Case: ";
    for(int i=0;i<10;i++){
        cout<<worst[i]<<" ";
    }
    selectionsort(worst,10);
    cout<<endl<<endl;

    cout<<"Random Case: ";
    for(int i=0;i<10;i++){
        cout<<random[i]<<" ";
    }
    selectionsort(random,10);
    cout<<endl<<endl;
}


return 0;
}
```

Output :

1. Insertion Sort

```
Enter choice:
1. Insertion Sort
2. Selection Sort
1
Best Case: 20 41 62 83 104 125 146 167 188 209
Sorted array : 20 41 62 83 104 125 146 167 188 209
Number of Comparisons : 9
Number of Swaps : 0


Worst Case: 209 188 167 146 125 104 83 62 41 20
Sorted array : 20 41 62 83 104 125 146 167 188 209
Number of Comparisons : 45
Number of Swaps : 45


Random Case: 2 12 76 5 29 7 88 102 35 234
Sorted array : 2 5 7 12 29 35 76 88 102 234
Number of Comparisons : 18
Number of Swaps : 9


PS C:\C++ Learning Course>
```

2. Selection Sort

```
Enter choice:
1. Insertion Sort
2. Selection Sort
2
Best Case: 20 41 62 83 104 125 146 167 188 209
Sorted array : 20 41 62 83 104 125 146 167 188 209
Number of Comparisons : 45
Number of Swaps : 0


Worst Case: 209 188 167 146 125 104 83 62 41 20
Sorted array : 20 41 62 83 104 125 146 167 188 209
Number of Comparisons : 45
Number of Swaps : 5


Random Case: 2 12 76 5 29 7 88 102 35 234
Sorted array : 2 5 7 12 29 35 76 88 102 234
Number of Comparisons : 45
Number of Swaps : 5


PS C:\C++ Learning Course>
```

Conclusion :

1. In this experiment, we learnt how to perform insertion sort and selection sort algorithms and analyze them using their time complexities.
2. From our observations, we come to a conclusion that insertion sort is more efficient than selection sort as selection sort always has a time complexity **$O(n^2)$** while insertion sort can achieve a time complexity of **$O(n)$** at its best.