AOA EXPERIMENT 1

60004210118
HEMANT SINGH
B1

Aim:

The aim of this experiment is to compare and analyze the time complexity of two sorting algorithms - selection sort and insertion sort. The main objective is to determine which algorithm performs better in terms of time complexity when sorting a given input data set.

Theory:

Selection sort involves finding the smallest element in the unsorted portion of the array and swapping it with the element at the beginning of the unsorted portion. This process is repeated until the entire array is sorted. The time complexity of selection sort is O(n^2), where n is the size of the input array.

Insertion sort involves iteratively inserting each element of the unsorted portion of the array into its proper place in the sorted portion of the array. The time complexity of insertion sort is also O(n^2), but it has been observed that insertion sort performs better than selection sort for small input sizes.

When analyzing the time complexity of sorting algorithms, it is essential to consider the best case, average case, and worst-case scenarios. The best-case scenario occurs when the input data set is already sorted. The worst-case scenario occurs when the input data set is in reverse order, and the average case occurs when the input data set is randomly ordered.

To analyze the best case scenario, we can simply check if the input data set is already sorted before applying the sorting algorithm. If the data set is already sorted, then the algorithm can simply return the sorted data set without any additional processing. For selection sort and insertion sort, the best case scenario has a time complexity of O(n), where n is the size of the input array. This is because the algorithm only needs to iterate through the array once to determine that it is already sorted.

The worst-case scenario for selection sort and insertion sort occurs when the input data set is in reverse order. In this scenario, the time complexity of both algorithms is

O(n^2), where n is the size of the input array. This is because the algorithm needs to perform a large number of comparisons and swaps to sort the data set.

The average-case scenario occurs when the input data set is randomly ordered. The time complexity of sorting algorithms in the average case can be more challenging to determine, as it depends on the specific input data set. However, in general, the time complexity of selection sort and insertion sort in the average case is also O(n^2).

Code:

Insertion Sort

```
#include <stdio.h>
#include <time.h>

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

void main() {
   clock_t start_t, end_t;
   double total_t;
   int size=15000;
   int arr[size];

   //array for best case
```

```
    // for(int i=0;i<size;i++){
    //     arr[i]=i;
    //     // printf("%d\n",arr[i]);
    // }

    //array for avg case
    for(int i=size;i>0;i--){
        arr[i]=rand();
        // printf("%d\n",arr[i]);
    }

    //array for worst case
    // for(int i=size;i>0;i--){
    //     arr[size-i]=i;
    //     // printf("%d\n",arr[i]);
    // }

    start_t = clock();
    insertionSort(arr,size);
    end_t = clock();
    total_t = (double)(end_t - start_t) ;
    printf("Total time taken by CPU for avg : %f\n", total_t  );
    printf("Exiting of the program...\n");
}
```

FOR 15000

```
/tmp/M0jDWI3x4j.o
Total time taken by CPU for best : 102.000000
Exiting of the program...
```

```
/tmp/M0jDWI3x4j.o
Total time taken by CPU for avg : 219429.000000
Exiting of the program...
```

```
/tmp/M0jDWI3x4j.o
Total time taken by CPU for worst : 411657.000000
Exiting of the program...
```

FOR 30000

```
/tmp/CHqMRyKHhO.o
Total time taken by CPU for best : 158.000000
Exiting of the program...
```

```
/tmp/CHqMRyKHhO.o
Total time taken by CPU for avg : 816069.000000
Exiting of the program...
```

```
/tmp/CHqMRyKHhO.o
Total time taken by CPU for worst : 1616842.000000
Exiting of the program...
```

Selection Sort

```
#include <stdio.h>
#include <time.h>

void selectionSort(int *arr,int size){
   int min,temp;
   for(int i=0;i<size;i++){
      min = i;
      temp = arr[i];
      for(int j=i+1;j<size;j++){
         if(arr[min]>arr[j]){
            min = j;
         }
      }
      arr[i]=arr[min];
      arr[min]=temp;
   }
}

void main() {
   clock_t start_t, end_t;
   double total_t;
   int size=15000;
   int arr[size];

   //array for best case
   // for(int i=0;i<size;i++){
   //    arr[i]=i;
   //    // printf("%d\n",arr[i]);
   // }
```

```
    //array for avg case
    for(int i=size;i>0;i--){
       arr[i]=rand();
       // printf("%d\n",arr[i]);
    }

    //array for worst case
    // for(int i=size;i>0;i--){
    //     arr[size-i]=i;
    //     // printf("%d\n",arr[i]);
    // }

    start_t = clock();
    selectionSort(arr,size);
    end_t = clock();
    total_t = (double)(end_t - start_t) ;
    printf("Total time taken by CPU for avg : %f\n", total_t  );
    printf("Exiting of the program...\n");
}
```

FOR 15000

```
/tmp/CHqMRyKHhO.o
Total time taken by CPU for best : 304214.000000
Exiting of the program...
```

```
/tmp/CHqMRyKHhO.o
Total time taken by CPU for avg : 337036.000000
Exiting of the program...
```

```
/tmp/CHqMRyKHhO.o
Total time taken by CPU for worst : 341823.000000
Exiting of the program...
```

FOR 30000

```
/tmp/CHqMRyKHhO.o
Total time taken by CPU for best : 1216956.000000
Exiting of the program...
```

```
/tmp/CHqMRyKHhO.o
Total time taken by CPU for avg: 1345398.000000
Exiting of the program...
```

```
/tmp/CHqMRyKHhO.o
Total time taken by CPU for worst: 1391455.000000
Exiting of the program...
```

Output:

For 15,000 Elements:

Sorting Algorithm	Best Case	Average Case	Worst Case
Insertion Sort	102	219429	411657
Selection Sort	304214	337036	341823

For 30,000 Elements:

Sorting Algorithm	Best Case	Average Case	Worst Case
Insertion Sort	158	816069	1616842
Selection Sort	1216956	1345398	1391455

Conclusion:

In this experiment, we have analysed the time complexity of selection sort and insertion sort for input data sets of size 15000 and 30000. We have measured the time taken by each algorithm to sort the data set and comparing their performance. By analysing the results for the best case, average case, and worst-case scenarios, we gained a better understanding of the time complexity of selection sort and insertion sort and determine which algorithm is more efficient for sorting large data sets.