DESIGN AND ANALYSIS OF ALGORITHMS

EXPERIMENT 1B

ANUSHKA ACHARYA
CSE DS D1 BATCH
UID: 2021700001

AIM: TO FIND RUNNING TIME OF AN ALGORITHM

THEORY:
The goal of the analysis of algorithms is to compare algorithms (or solutions) mainly in terms of running time but also in terms of other factors (e.g., memory, developer effort, etc.) The running time of an algorithm on a particular input is the number of primitive operations or "steps" executed.
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.
Characteristics of Insertion Sort:

- This algorithm is one of the simplest algorithm with simple implementation
- Basically, Insertion sort is efficient for small data values
- Insertion sort is adaptive in nature, i.e. it is appropriate for data sets which are already partially sorted.

Insertion Sort - GeeksforGeeks
Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted portion. This process is repeated for the remaining unsorted portion of the list until the entire list is sorted. One variation of selection sort is called "Bidirectional selection sort" that goes through the list of elements by alternating between the smallest and largest element, this way the algorithm can be faster in some cases.
The algorithm maintains two subarrays in a given array.

- The subarray which already sorted.
- The remaining subarray was unsorted.

Selection Sort Algorithm - GeeksforGeeks

ALGORITHMS:

(A) FOR GENERATING RANDOM NUMBERS:
STEP_1: Start. Include the standard c library in the program.
STEP_2: Define an integer x
STEP_3: Run a for loop from 0 to 100000 to generate 100000 numbers
STEP_4: Access the random function rand() inside the loop to generate 100000 random numbers.
STEP_5: Print the numbers and save the text file. Stop

(B) INSERTION SORT:

STEP_1: Start. Initialise size of the array as 100 to consider first block of 100 elements.

STEP_2: Open the file where the random numbers are stored to read the numbers.

STEP_3: Start a while loop for size<=100000 so the program executes for all the blocks. In the while loop:

(1) Initialise time taken as 0. Declare begin time using clock() function.

(2) Declare an array of the size given and start reading the random numbers from the file using a for loop.

(3) Consider first two elements, compare them. If second element is greater than the first, swap them and put the first element in the sorted subarray.

(4) Consider second and third element, compare them and swap if needed. Now if the third element is less than both second and first elements, put it as the first element of sorted subarray by swapping.

(5) Repeat steps 3 and 4 until the last element is reached.

(6) Print the sorted subarray, increment the size and bring the file pointer back to beginning of the file.

(7) Declare end time using clock() function and calculate runtime using begin and end times.

STEP_4: Stop

(C) SELECTION SORT:

STEP_1: Start. Declare a function which swaps two elements using a temporary variable.

STEP_2: Declare a function to perform the selection sort process. In this function:

(1) Use two for loops to find the minimum element of the array by comparing two consecutive elements.

(2) Once you find the minimum element, bring it to the front by swapping it with the first element of the array.

(3) Return the sorted array to the main function.

STEP_5: In the main function, initialise size of array as 100. Initialise time taken as 0. Declare begin time using clock() function.

STEP_6: Open the file where the random numbers are stored to read the numbers.

STEP_7: Start a while loop for size<=100000 so the program executes for all the blocks. In the while loop:

(1) Declare an array of the size given and start reading the random numbers from the file using a for loop.

(2) Call the sort function to get the sorted array

(3) Print the array using for loop, increment size by 100 and bring the file pointer back to beginning of the file.

(4) Declare end time using clock() function and calculate runtime using begin and end times.

STEP_8: Stop

CODE:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
```

```
int main(){
   int x;
   for(int i=0;i<=100000;i++){
   x=rand()%100000;
   printf("%d ",x);
   }
   return 0;
}
```

INSERTION SORT:

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

void main(){
int pos,swap,size=100;
FILE *fp;
fp = fopen("anu.txt","r");
while(size<=100000){
double time_taken=0.0;
clock_t begin=clock();
int a[size];
for(int j=0;j<size;j++){
fscanf(fp,"%d ",&a[j]);
}
for(int k = 0; k < (size) - 1; k++){
pos=k;
for(int p = k + 1; p < size; p++)
{
if(a[pos] > a[p])
pos=p;
}
if(pos != k)
{
int temp=a[k];
a[k]=a[pos];
a[pos]=temp;
}
}
int i1, element, j1;
   for (i1 = 1; i1 < size; i1++) {
   element = a[i1];
   j1 = i1 - 1;
   while (j1>= 0 && a[j1] > element) {
         a[j1 + 1] = a[j1];
         j1 = j1 - 1;
      }
      a[j1 + 1] = element;
   }
for(int test1=1;test1<size;test1++){
printf("%d\t",a[test1]);
}
   printf("\n\n");
   size=size+100;
```

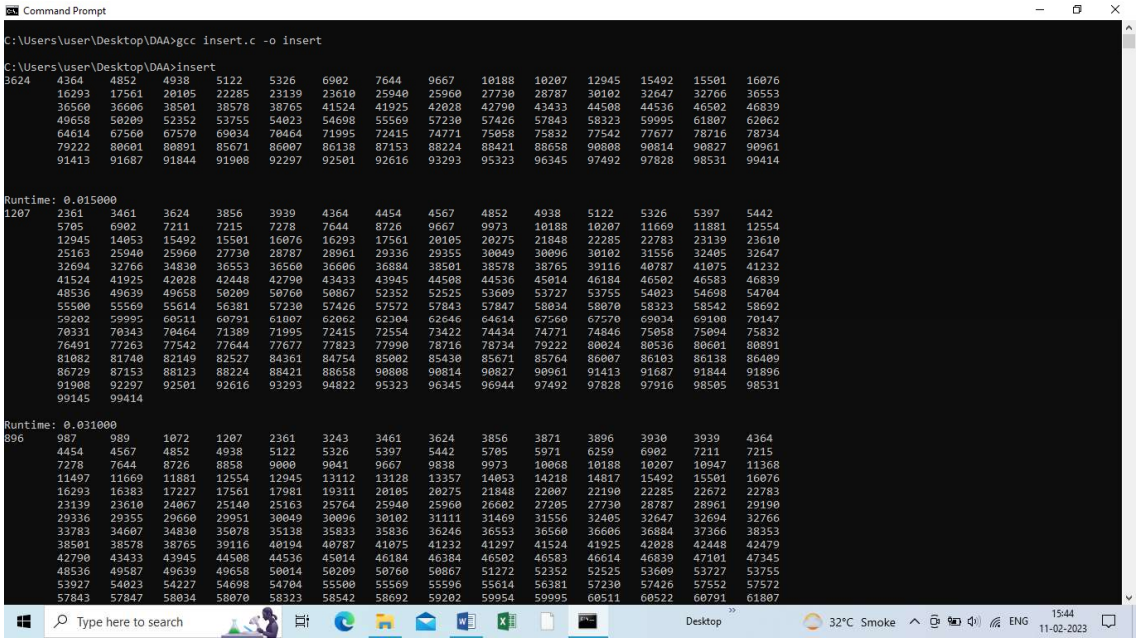
```
   fseek(fp,0,SEEK_SET);
clock_t end=clock();
        time_taken = (double)(end-begin)/CLOCKS_PER_SEC;
        printf("Runtime: %f\n",time_taken);
        }
}
```

SELECTION SORT:

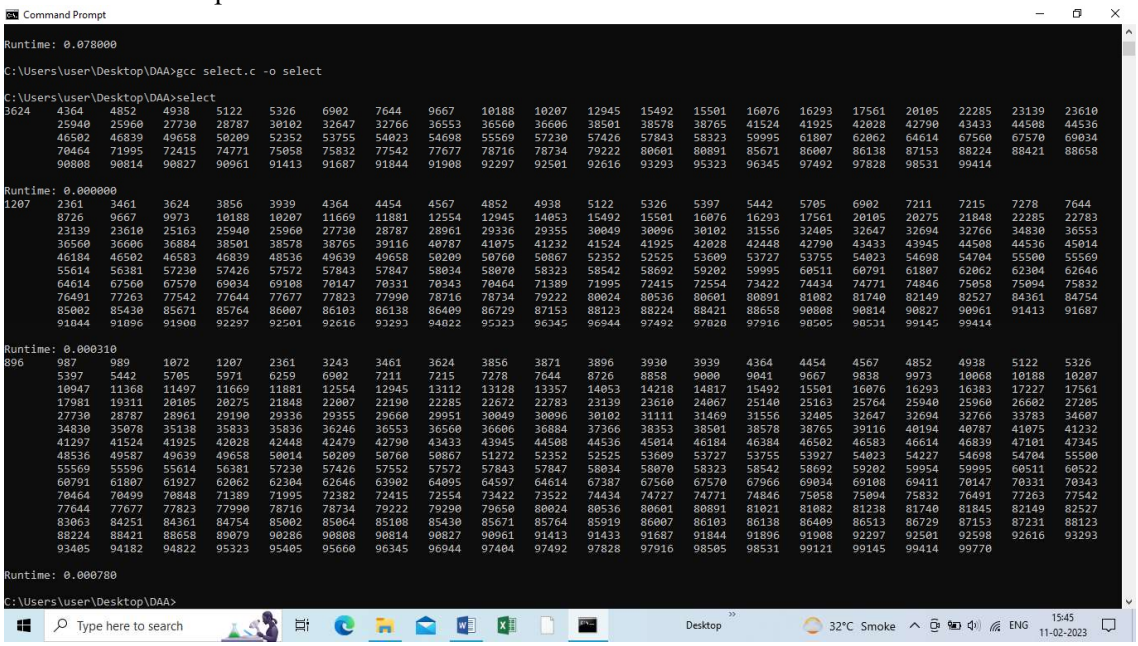
```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

void swap(int*a , int*b){
        int temp = *a;
        *a = *b;
        *b = temp;
}

void selSort(int* arr , int size){
        for(int i=0;i<size-1;i++){
                int minId = i;
                for(int j=i+1;j<size;j++){
                        if(arr[j]<arr[minId]){
                                minId = j;
                        }
                }
                if(i!=minId){
                        swap(&arr[i],&arr[minId]);
                }
        }
}

int main(){
int size=100;
FILE *fp;
fp = fopen("anu.txt","r");
double time_taken=0.0;
clock_t begin=clock();
while(size<=100000){
int a[size];
for(int j=0;j<size;j++){
fscanf(fp,"%d ",&a[j]);
}
selSort(a,size);
for(int test1=1;test1<size;test1++){
printf("%d\t",a[test1]);
}
   printf("\n\n");
   size=size+100;
   fseek(fp,0,SEEK_SET);
clock_t end=clock();
        time_taken = (double)(end-begin)/CLOCKS_PER_SEC;
        printf("Runtime: %f\n",time_taken);
        }
}
```

OUTPUT:

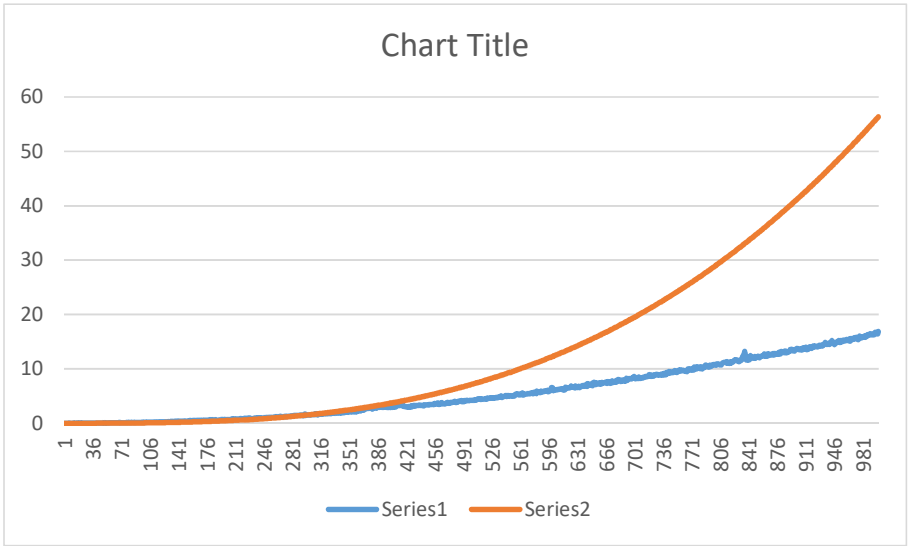
Insertion sort output for first three blocks:



Selection sort output for first three blocks:



GRAPH:



OBSERVATION:

In this graph, the blue line represents selection sort whereas the red curve represents insertion sort algorithm. It can be seen clearly then selection sort has a higher runtime than the insertion sort algorithm. This is because in selection sort the array is scanned repeatedly to find the minimum element and then the minimum element is brought to the front by swapping. So the number of comparison needed is more which gives this algorithm a time complexity as O(n^2) in all cases. Opposed to this, in insertion sort, consecutive elements are compared and swapping occurs as soon as the order is disrupted. This leads to less comparisons and time complexity of O(n) in best case scenario. This is why, insertion sort is more efficient than selection sort algorithm.

CONCLUSION:

From this experiment I learnt how to perform selection and insertion sort algorithms. I used the sorting techniques on 100000 random numbers and found out that runtime for insertion sort is less than select sort which helped me analyse that insertion sort is the algorithm more efficient between the two.