DESIGN AND ANALYSIS OF ALGORITHM

PART B

Sujal Chordia
D1
2021700015

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

CODE:

1) Random numbers :

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



int main(){


    FILE *f;
    f = fopen("new.txt","w");
    if (f == NULL)
        {
            printf("ERROR Creating File!");
            exit(1);
        }
    for(int i=0; i<100000 ; i++){
        int num = rand()%100000;
        fprintf(f,"%d\n",num);
    }
    printf("DONE!!");
    fclose(f);
    return 0;
}
```

2) Selection and inertion sort code :

```
#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]);
      }
   }
}
```

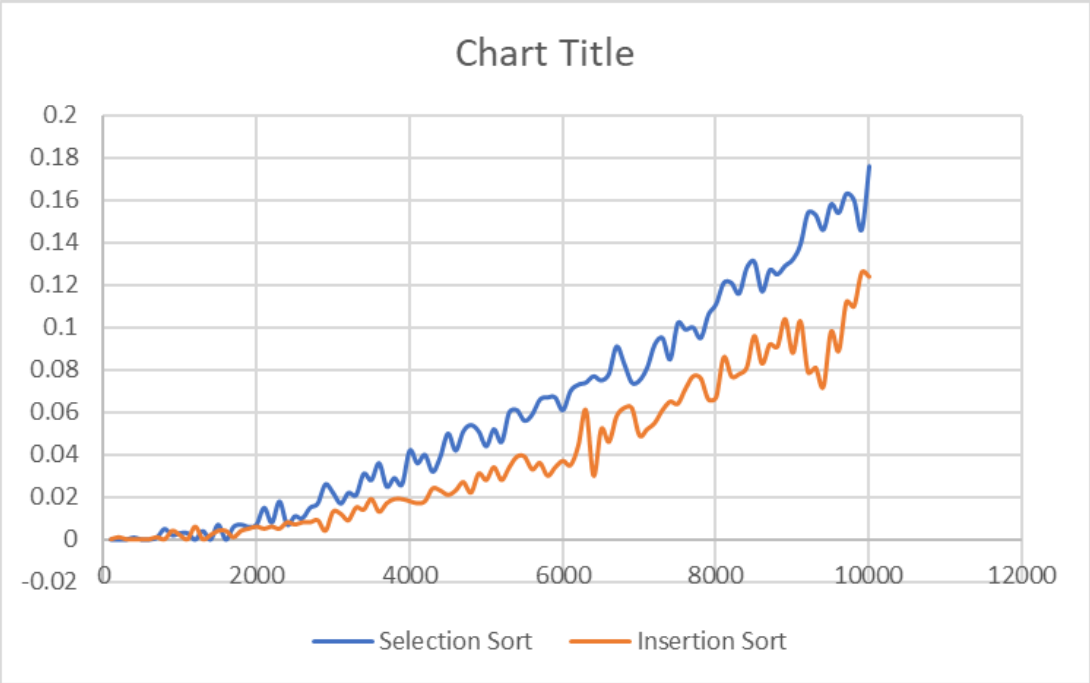
```
void insertSort(int *arr, int n){
   int i, key, j;
   for (i = 1; i < n; i++)
   {
      key = arr[i];
      j = i - 1;

      /* Move elements of arr[0..i-1],
         that are greater than key,
         to one position ahead of
         their current position */
      while (j >= 0 && arr[j] > key)
      {
          arr[j + 1] = arr[j];
          j = j - 1;
      }
      arr[j + 1] = key;
   }
}


int main(){

   /*int  i=0;
   int arr[100];
   while(i<=100){
      arr[i] = rand();
      i++;
      printf("%d\n",arr[i]);
   }*/
   for(int i=1;i<=100;i++){
      int j=0;
      int numberArray[100000];
      FILE *f;
      f  =  fopen("new.txt","r");
      for (j = 0; j < 100000; j++){
         fscanf(f, "%d,", &numberArray[j] );
         }
      fclose(f);
      clock_t t;
      t = clock();
      insertSort(numberArray,i*100);
      t = clock() - t;
      double time_taken = ((double)t)/CLOCKS_PER_SEC;
      printf("%f\n",time_taken);

   }
   /*for(int j=0;j<100000;j++){
      printf("%d\n",numberArray[j]);
   }
   fclose(f);
   i=0;
   f  =  fopen("new.txt","r");
   for (i = 0; i < 100000; i++){
       fscanf(f, "%d,", &numberArray[i] );
```

```
        }
        for(int j=0;j<100000;j++){
        printf("%d\n",numberArray[j]);
    }
    /*int  i=0;
    int arr[100];
    while(i<=100){
        arr[i] = rand();
        i++;
        printf("%d\n",arr[i]);
    }
    insertSort(numberArray,100000);
    printf("The sorted array is");
    for(int j=0;j<100000;j++){
        printf("%d\n",numberArray[j]);
    }
    fclose(f);

    fun();
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds

    printf("fun() took %f seconds to execute \n", time_taken);*/

}
```

Output :

```
PS C:\Users\smsha\Desktop\Rupali> & .\"insAndSel.exe"
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000
0.000000
0.001000
0.001000
0.001000
0.001000
0.001000
0.002000
0.002000
0.002000
0.001000
0.002000
0.002000
0.003000
0.003000
0.003000
0.004000
0.004000
0.003000
0.004000
0.006000
0.006000
0.006000
0.006000
0.006000
0.006000
0.006000
0.007000
0.007000
0.008000
```

GRAPH :



The graph above has blue curves / lines which is selection sort and also has orange curves/lines which is insertion sort . Looking at this graph we clearly interpret that the selection sort and insertion sorting in starting were having same run time complexity but later the run time complexity of selection sort kept on increasing . The total running time for selection sort has three parts: The running time for all the calls to index Of Minimum . The running time for all the calls to swap . The running time for the rest of the loop in the selection Sort function. Whereas Insertion sort has an average and worst-case running time of O (n 2) **O**(n^2) O(n2**)**, so in most cases, a faster algorithm is more desirable.

Conclusion : By performing this experiment I learned about selection and insertion sort also I worked on 100000 numbers and ploted them on graph through which learned about the run time complexity of insertion and selection sort . While working it was observed the run time complexity of insertion sort is less than selection sort .