

# DESIGN AND ANALYSIS OF ALGORITHMS

## EXPERIMENT 1B

**Name:** Janhavi R Deshmukh

**Branch:** CSE(DS) D1

**UID:** 2021700017

**Aim: – To find the running time of an algorithm.**

### Theory:

Algorithm analysis is an important part of computational complexity theory, which provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

Time Complexity: Time Complexity is defined as the number of times a particular instruction set is executed rather than the total time taken. It is because the total time took also depends on some external factors like the compiler used, processor's speed, etc.

Space Complexity: Space Complexity is the total memory space required by the program for its execution.

### 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.

### 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.

### Selection sort:

It first finds the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. In this algorithm, the array is divided into two parts, first is sorted part, and another one is the unsorted part. Initially, the sorted part of the array is empty, and unsorted part is the given array. Sorted part is placed at the left, while the unsorted part is placed at the right. In selection sort, the first smallest element is selected from the unsorted array and placed at the first position. After that second smallest element is selected and placed in the second position. The process continues until the array is entirely sorted.

### Algorithm:

To create a text file containing 100000 random numbers

Step 1: Start. Include stdlib.h library to use the rand function.

Step 2: Initialise an array of size 100000 number[100000]

Step 3:Run a for loop from i=0 to 100000

Step 4: Call the rand() function and store the value in number[i] array .

Step 5:Create a input.txt file and save the 100000 randomly generated values which is in the range from 0 to 100000 .Stop.

### Selection Sort:

Step 1: Start. Write a function to swap two numbers using a temporary variable.

Step 2: Declare a function selectionSort(int arr[],n) which takes the array and given size of the array as argument.

Step 3:Run for loop which runs from i=0 to n-1 and take the minimum element at first position of array by initialising minIndex=i

Step 4:Run another for loop from j=i+1 to n which will keep checking the next two consecutive numbers in array.

Step 5: Check if arr[j]<arr[minIndex] and find the smallest element

Step 6: If smallest element is found update minIndex=j and call the swap function by swapping it with first element of array.

Step 7:Now in main ,access the text file input.txt where the random numbers are stored using fopen

Step 8:Initialise block =1 and size of data array as 100 i.e data[size]

Start 9:Run a while loop while(block<=1000) to access 1000 blocks , in this loop

1. Run a for loop from i=0 to i=size and start reading the random numbers in data[i] array.
2. Declare clock\_t t to get starting time of program
3. Call selectionsort(data,size) to get the sorted array of first 100 numbers
4. Get the time elapsed by t=clock()-t
5. Calculate the time take for block by t/clocks\_per\_sec and print the runtime for first block
6. Update size=size+100 and block++
7. Update the pointer by fseek to move the cursor to the start of the file.

Step 10: Stop

### Insertion Sort :

Step 1: Start. Declare insertionSort(int arr[],int n) function

Step 2: Iterate from arr[1] to arr[n] over the array.

Step 3:Compare the current element (key) to its predecessor. If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

Step 4: Repeat the steps 7,8,9 as above in main function and call insertionSort(data,size) in main

Step 5:Stop

### **Program:**

#### 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 selectionSort(int arr[],int n)
{
    for(int i=0;i<n-1;i++)
    {
        int minIndex=i;
        for(int j=i+1;j<n;j++)
        {
            if(arr[j]<arr[minIndex])
            {
                minIndex=j;
            }
        }

        swap(&arr[minIndex],&arr[i]);
    }
}

int main()
{
    //to generate 1000000 random numbers
    /*int numbers[1000000];
    for(int i=0;i<1000000;i++)
    {

        numbers[i]=rand()% 100000;
        printf("%d \n",numbers[i]);
    }
}
```

```

}*/

FILE* ptr;

// file in reading mode
ptr = fopen("input.txt", "r");

if (NULL == ptr)
{
    printf("file can't be opened \n");
}

int block=1;
int size=100;
while(block<=1000)
{
    int data[size];
    for(int i=0;i<size;i++)
    {
        fscanf(ptr,"%d",&data[i]);
        //printf("%d ",data[i]);
    }

    clock_t t;
    t = clock();
    selectionSort(data,size);
    if(block<3) //this prints sorted first 2 blocks only
    {
        printf("\nafter sorting block %d:\n",block);
        for(int i=0;i<size;i++)
        {
            printf("%d \n",data[i]);
        }
    }

    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC;
    printf("\nRuntime for block number %d : %f\n",block,time_taken);

    size=size+100;
    block++;
    fseek(ptr,0,SEEK_SET); //moving cursor again to start pointer of txt file

}

fclose(ptr);

```

```
}
```

### Insertion sort:

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

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

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main()
{
    FILE* ptr;

    // file in reading mode
    ptr = fopen("input.txt", "r");

    if (NULL == ptr)
    {
        printf("file can't be opened \n");
    }

    int block=1;
    int size=100;
    while(block<=1000)
    {
        int data[size];
        for(int i=0;i<size;i++)
        {
            fscanf(ptr,"%d",&data[i]);
            //printf("%d",data[i]);
        }
    }
}
```

```

clock_t t;
t = clock();
insertionSort(data,size);
if(block<3) //this prints sorted first 2 blocks only
{
    printf("\nafter sorting block %d:\n",block);
    for(int i=0;i<size;i++)
    {
        printf("%d \n",data[i]);
    }
}

t = clock() - t;
double time_taken = ((double)t)/CLOCKS_PER_SEC;
printf("\nRuntime for block number %d : %f\n",block,time_taken);

size=size+100;
block++;
fseek(ptr,0,SEEK_SET); //moving cursor again to start pointer of txt file

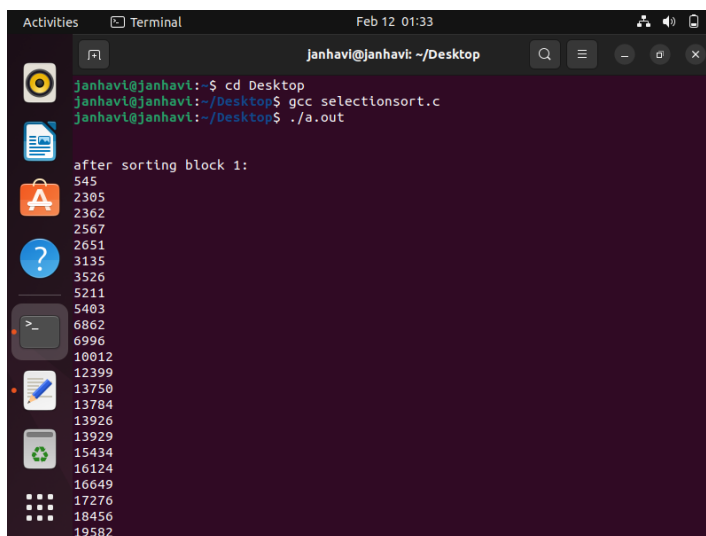
}

fclose(ptr);
}

```

## Output:

Selection sort: (first 2 blocks)



```

janhavi@janhavi: ~/Desktop
janhavi@janhavi: ~/Desktop$ cd Desktop
janhavi@janhavi: ~/Desktop$ gcc selectionsort.c
janhavi@janhavi: ~/Desktop$ ./a.out

after sorting block 1:
545
2305
2362
2567
2651
3135
3526
5211
5403
6862
6996
10012
12399
13750
13784
13926
13929
15434
16124
16649
17276
18456
19582

```

```
Activities Terminal Feb 12 01:34
janhavi@janhavi: ~/Desktop
99932
1 0.000453
after sorting block 2:
545
723
1474
2245
2305
2362
2567
2651
3135
3526
4567
5211
5403
5732
5771
6219
6862
6996
8117
8235
9441
9503
9859
10012
```

Insertion sort: (first 2 blocks)

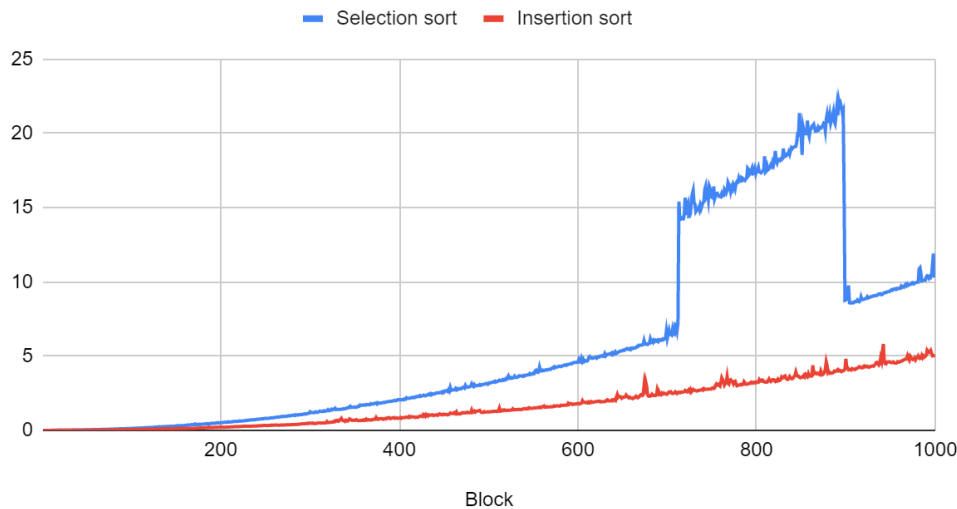
```
Activities Terminal Feb 12 03:48
janhavi@janhavi: ~/Desktop
after sorting block 1:
545
2305
2362
2567
2651
3135
3526
5211
5403
6862
6996
10012
12399
13750
13784
13926
13929
15434
16124
16649
17276
18456
19582
20059
20925
21530
```

```
Activities Terminal Feb 12 03:49
janhavi@janhavi: ~/Desktop
99932
0.000103
after sorting block 2:
545
723
1474
2245
2305
2362
2567
2651
3135
3526
4567
5211
5403
5732
5771
6219
6862
6996
8117
8235
9441
9503
9859
10012
```

- runtime for all 1000 blocks

## Graph:

Selection sort and Insertion sort runtime vs block number



## Observation:

From above graph, it is clearly visible that the running time for selection sort algorithm representing blue line is more than the running time for insertion sort algorithm.

The insertion sort inserts the values in a pre sorted file to sort a set of values. On the other hand, the selection sort finds the minimum number from the list and sort it in some order which requires multiple scanning of the array and therefore more comparisons. As a result the best case time complexity of selection sort is  $O(n^2)$ . On the other hand in insertion sort the number of times an element is moved or swapped is greater than the comparisons made. Due to this time complexity in best case for insertion sort is  $O(n)$ .

Among both of the sorting algorithm, the insertion sort is fast, efficient, stable while selection sort only works efficiently when the small set of elements is involved or the list is partially previously sorted.

## Conclusion:

From this experiment I learnt and understood the implementation of selection and insertion sort by performing it on 100000 randomly generated numbers by using `rand()` function. Also learnt how to use the `clock()` function for calculating the runtime of a program. Runtime for insertion sort is less than that of selection sort and therefore insertion sort algorithm is more efficient than selection sort.