# Istanbul Technical University

## Department of Computer Engineering

BLG 335E Analysis of Algorithms
Assignment 3 Report

Ece Naz Sefercioğlu
150130140

## Part A. Sorting in Linear Time

### 1)

| N | Execution Time (seconds) |
|---|---|
| 5 | 0.004154 |
| 10 | 0.000434 |
| 50 | 0.001484 |
| 100 | 0.001834 |
| 500 | 0.004762 |
| 1000 | 0.007112 |
| 5000 | 0.028411 |
| 10000 | 0.048813 |
| 50000 | 0.192414 |
| 100000 | 0.364979 |
| 500000 | 1.78944 |
| 1000000 | 3.59434 |

Time complexity of Radix Sort is $\Theta(nk)$, it has a linear complexity. When we inspect the results, we see ratio between N values are smiliar to ratio between execution times. It shows a linear increase. On smaller N values, this increment shows less linearity.

**2)**

For a sorting algorithm to be used in the intermediate step of Radix Sort, the algorithm must be stable, meaning that it should preserve the input order among equal elements.

To decide whether we can use Gnome Sort or the manipulated version of Freezing Sort, we should inspect if they are stable sorting algorithms or not.

For example let's say we have 3 students and their sucesfull submits on the assignments of the course Analysis of Algorithms 1.

{ 3:Alice, 3: Daisy, 5:Melinda }

A bug causes the order of them get mixed.

It becomes { 3:Daisy, 3: Alice, 5:Melinda } and we want to achive a descending order, so we use the manipulated version of Freezing Sort.

The outcome of the sort is { 5:Melinda, 3: Alice, 3:Daisy }.

We see the students' with the same amount of succesful submits, Alice and Daisy, input order is not preserved. With the aid of this counter example, we can say the manipulated version of **Freezing Sort** is not a stable algorithm and **can not be used** in the intermediate step of Radix Sort.

Another bug causes the order to get mixed again.

It becomes { 3:Alice, 5:Melinda, 3:Daisy } and we want to achive an ascending order, so we use Gnome Sort.

The outcome of the sort is { 3: Alice, 3:Daisy, 5:Melinda }

We see the students' with the same amount of succesful submits, Alice and Daisy, input order is preserved. With the aid of this example, we may say **Gnome Sort** is a stable algorithm and **may be used** in the intermediate step of Radix Sort.

To prove this assumption, we should inspect the code of the Gnome Sort. It starts to sort from the start of the array and compares the consecutive elements of the array. If they are equal or the left one is smaller then right one no exchange is made. If there can not be any exchange between the equal elements, their input order is preserved, so **Gnome Sort** is a stable algorithm and **can be used** in the intermediate step of Radix Sort.

# C++ Code for Gnome Sort

```
int i=1;
int j=2;
int temp;
while(i<size){
  if(arr[i-1]<=arr[i]){  // no swap when equal or smaller
    i=j;
    j++;
  }
  else{
    temp=a[i-1];
     a[i-1]=a[i];
     a[i]=temp;
     i--;
     if(i==0)
      {
           i=j;
           i=j+1;
           }
    }
}
```

## Part B. Heap Sort

### 1)

| N | Execution Time (seconds) |
|---|---|
| 5 | 0.003514 |
| 10 | 0.000379 |
| 50 | 0.000989 |
| 100 | 0.001783 |
| 500 | 0.004851 |
| 1000 | 0.007331 |
| 5000 | 0.058753 |
| 10000 | 0.073077 |
| 50000 | 0.328065 |
| 100000 | 0.670332 |
| 500000 | 3.78563 |
| 1000000 | 8.02339 |

Time complexity of Heap Sort is Θ(n log(n)). When we inspect the results, we see ratio between N values are not very smiliar to ratio between execution times. It shows an increase smiliar to nlog(n) ratio . On smaller N values, this increment shows less stability. With respect to complexity

**2)** Output of the execution:

```
B Wins
At the end of the execution:
Clan A has a total of 50460159 CPs.
Clan B has a total of 50528505 CPs.
Difference between amount of the CPs = 68346
```