# ANALYSIS OF ALGORITHMS I
# PROJECT 3
# REPORT

## Part A. Sorting in Linear Time (50 pts)

**1-)** Comparison based sorting algorithms such as insertion sort, quicksort or merge sort's running time (worst case) can be O(n*logn), not more faster. We prove that by using decision tree. On the other hand, counting sort algorithm is running in linear time. It include just four times (two times n sized, t times k sized, n=elements of array, k= max element's range) loop so that it is running O(n+k). While we are implementing radix sort be, we should use counting sort which time is just is both fast and stable. In sum up, sorting digit by digit with radix sort whose running time is O(d*n)(b is number of bits and for 0 to $n^{d-1}$, b=d*logn).

| RADIXSORT | | | | | | |
|---|---|---|---|---|---|---|
| **N=** | **10** | **100** | **1000** | **10000** | **100000** | **1000000** |
| **Running time(s)** | **0** | **0.001** | **0.01** | **0.131** | **0.844** | **4.064** |

**2-)** Radix sort's cornerstone is whether intermediate sorting algorithm is stable or not. Stable means that preserve relative order of same values, for instance, if there are two elements A and B, and their keys are equals. If in unsorted array A is before B, then in sorted array A must be before B. Counting sort is a good example for stable sorting. Gnome sort is similar to insertion sort and when we analysis it on basic example array (such as [3, 5, 3, 2, 5], I use it as an example) include equal keys, it is clearly seen that Gnome sort is stable because that when values are compared with if control statement, we can say that only if values are different they swap, not values are less than equal (<=). In other words, if they are same key, their place is not changed. . On the other hand, manipulated freezing sort algorithm is not stable because those values are exchanged from right or left side so that there is a possibility to change same values' relative order. It is seen on debugging.
Iuse(http://pythontutor.com/visualize.html#mode=edit) also in order to see how elements are change.

## Part B. Heap Sort (50 pts)

**1-)** Max-Heapify function's running time is O(lgn).In Build-Max-Heap function, Max-heapify is called n time so that it costs O(n*lgn). However, tighter bound of Build-Max-Heap function is O(n). In HeapSort function, n-1 times Max-Heapify and one times Build-Max-Heap is called. Totally, Heapsort's running time is O(n*lgn).

| HEAPSORT | | | | | | |
|---|---|---|---|---|---|---|
| **N=** | **10** | **100** | **1000** | **10000** | **100000** | **1000000** |
| **Running time(s)** | **0** | **0.001** | **0.007** | **0.09** | **1.152** | **14.558** |

**2-) I try to complete this part, yet it is not works correctly. I write it as a comment line**

**By OKAN YILDIRIM**
**150160537**