

BLG 335E HOMEWORK 3

REPORT

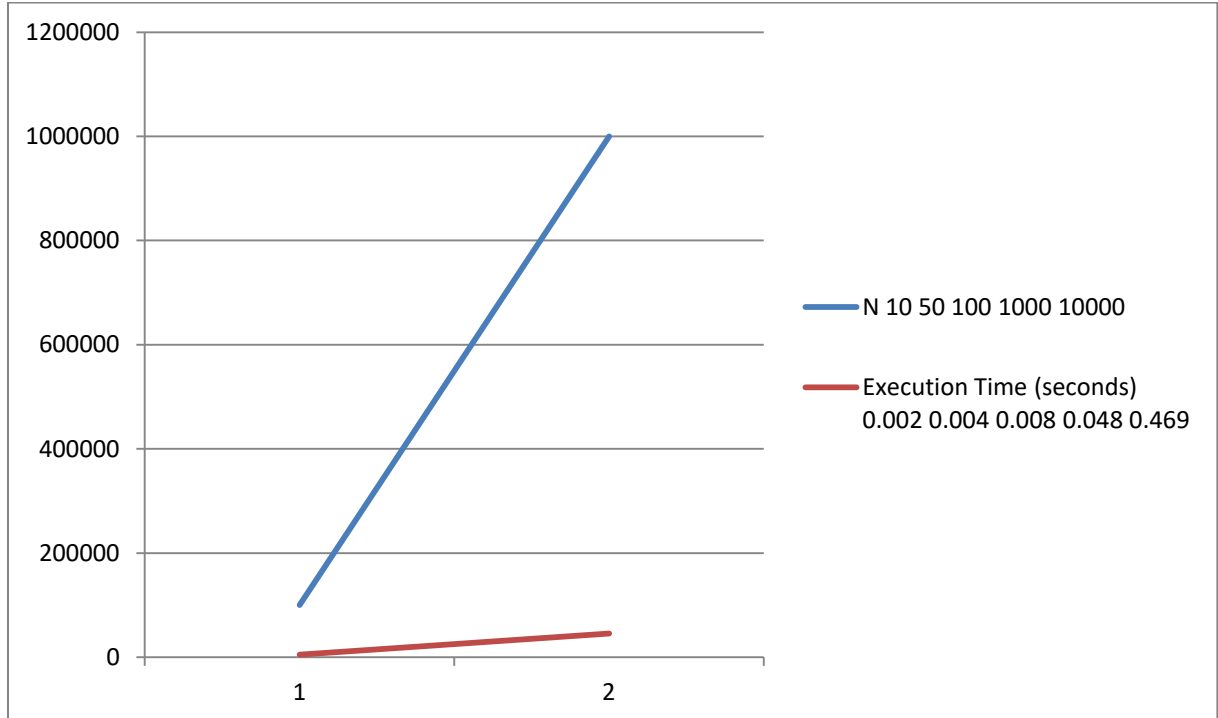
Student Name: Halit Uyanık

Student Number: 150140138

Part A. Sorting in Linear Time

1-)

N	Execution Time (seconds)
10	0.002
50	0.004
100	0.008
1000	0.048
10000	0.469
100000	4.759
1000000	45.377



Since counting sort is $O(n)$, I was also expecting a linear increase in execution time parallel to N . This can be more easily observed after 1000 values. The reason why $N < 1000$ is so near to each other is most probably fast execution of operations.

2-)

In order to use an algorithm in the intermediate step of radix sort, the algorithm needs to be stable. Which means it should not change the order of same values.

For an array of A = [7, 2, 4, 3, 4, 1, 6] lets check how both algorithms will be executed.

Gnome Sort:

i = 1, j = 2, size = 7

2 7 4 3 4 1 6

2 4 7 3 4 1 6

2 4 3 7 4 1 6

2 3 4 7 4 1 6

2 3 4 4 7 1 6

....

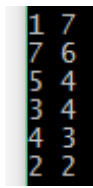
At this step it can be seen that the order of 4s are preserved. Hence it is a stable sorting algorithm that could be used in Radix sort.

Freezing Sort:

I have implemented the modified code and tested it on array: [7 , 2, 4 ,3, 4, 1, 6]

```
for (int frz = 0; frz < 7; frz++) {
    for (int i = frz; i < (7 + frz) / 2 + 1; i++) {
        if (i != n - i + frz && t[i].no < t[n - i + frz].no) {
            ts temp = t[n - i + frz];
            t[n - i + frz] = t[i];
            t[i] = temp;
        }
    }
    if (frz != 0) {
        for (int i = n - frz; i > (n - frz) / 2; i--) {
            if (i != n - i - frz && t[n - i - frz].no < t[i].no) {
                ts temp = t[n - i + frz];
                t[n - i + frz] = t[i];
                t[i] = temp;
            }
        }
    }
    cout << "After each phase:\n";
    for (int i = 0; i < 7; i++) {
        cout << t[i].id << " " << t[i].no << "\n";
    }
    cout << "-----\n";
}
for (int i = 0; i < 7; i++) {
    cout << t[i].id << " " << t[i].no << "\n";
}
```

The result was:

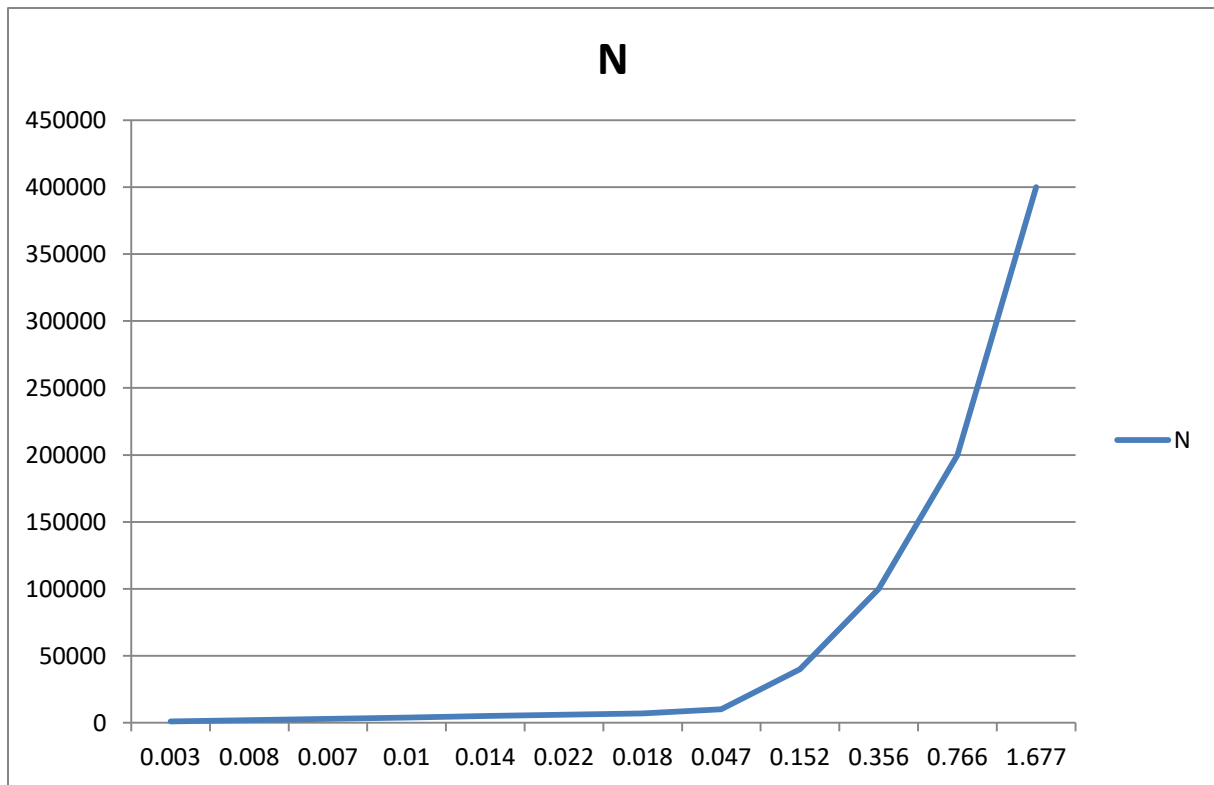


As can be seen the third 4 passed the 5th 4. Which means the algorithm is not stable and we can not use it in radix sort.

Part B. Heap Sort

1-)For different N values the results are like this:

N	Time (seconds)
1000	0.003
2000	0.008
3000	0.007
4000	0.01
5000	0.014
6000	0.022
7000	0.018
10000	0.047
40000	0.152
100000	0.356
200000	0.766
400000	1.677



As can be seen from the graph as N goes higher the execution time increases in a logarithmic way. The complexity of HeapSort is $O(n \cdot \lg n)$ in worst case time. Therefore the graph shows reasonable results.

2-)

The winner of the tournament is Team B. The final sums

Team A: 50460159

TeamB: 50528505

Player B wins.