BLG 335E - Analysis of Algorithms 1

HOMEWORK 1

Student Name: Halit Uyanık

Number: 150140138

A) Asymptotic Upper Bounds And Code Analysis

Merge Sort

O(n * log(n))

My Code:

I have implemented the merge-sort and insertion-sort algorithms directly from the slides. Therefore their complexity is expected to be the similar of those and can be seen from explanations and graphs.

```
void merge_sort(long int p, long int r) {
		int q = 0;
		if (p < r) {
			q = (p + r) / 2;
			merge_sort(p, q);
			merge_sort(q + 1, r);
			merge(p, q, r);
		}
}
```

Algorithm branches through arrays elements. Since this branching process works for n elements and at each step the limits are divided into two parts. We get log(n) from the branching part, and n from the element count ,therefore the complexity is O(n*log(n))

Insertion Sort

O(n^2)

My Code:

```
	void insertion_sort(long int n) {
		for (long int i = 1; i < n; i++) {
			Nokta pss = arr[i];
			long int j = i - 1;
			while (j > -1 && arr[j].distance > pss.distance) {
				arr[j + 1] = arr[j];
				j--;
			}
			arr[j + 1] = pss;
```

```
            }
        }
```

For each element in N we are iterating through a point till the start. Which means in the worst case N element will be iterated for N element. Therefore we take the worst case N^2 as the upper limit. Which can be seen below at graphs that the increase in N affects the insertion sort algorithm most.

Linear Search

O(n * k)

My Code:

```
            for (long int i = 0; i < N; i++) {
                for (long int j = 0; j < K; j++) {
                    if (lin_arr[j].distance == -1) {
                        lin_arr[arr_ptr] = arr[i];
                        arr_ptr++;
                        break;
                    }
                    else if (lin_arr[j].distance > arr[i].distance) {
                        lin_arr[j] = arr[i];
                        break;
                    }
                }
                if (arr_ptr == 10) {
                    arr_ptr = 0;
                }

            }
```

Since for each element in N the code iterates through K elements of the linear search array, we are doing an N*K operation in overall.

B) Average Time Results

N = 10					Average
	k = 1	k = 2	k = 10	k = N \ 2	
MS	0,002	0,001	0,004	0,002	0,00225
LS	0,001	0,001	0,008	0,001	0,00275
IS	0,002	0,002	0,005	0,002	0,00275

N= 100					Average
	k = 1	k = 2	k = 10	k = N \ 2	
MS	0,015	0,017	0,008	0,006	0,0115
LS	0,008	0,012	0,015	0,015	0,0125
IS	0,016	0,013	0,018	0,017	0,016

N = 1000					Average
	k = 1	k = 2	k = 10	k = N \ 2	
MS	0,131	0,107	0,129	0,09	0,11425
LS	0,088	0,085	0,118	0,106	0,09925
IS	0,089	0,135	0,09	0,143	0,11425

N=1000000					Average
	k = 1	k = 2	k = 10	k = N \ 2	
MS	55,887	57,645	57,652	57,749	57,23325
LS	54,324	53,883	54,425	54,341	54,24325
IS	1866	2059	1873	2032	1957,5

I need to make an explanation here as to why linear search and merge-sort are so close to each other for all calculations, I have tried the code at g++ in ssh.itu.edu.tr and the code worked very fast there. Hence it was not possible to get a proper timing for the execution.

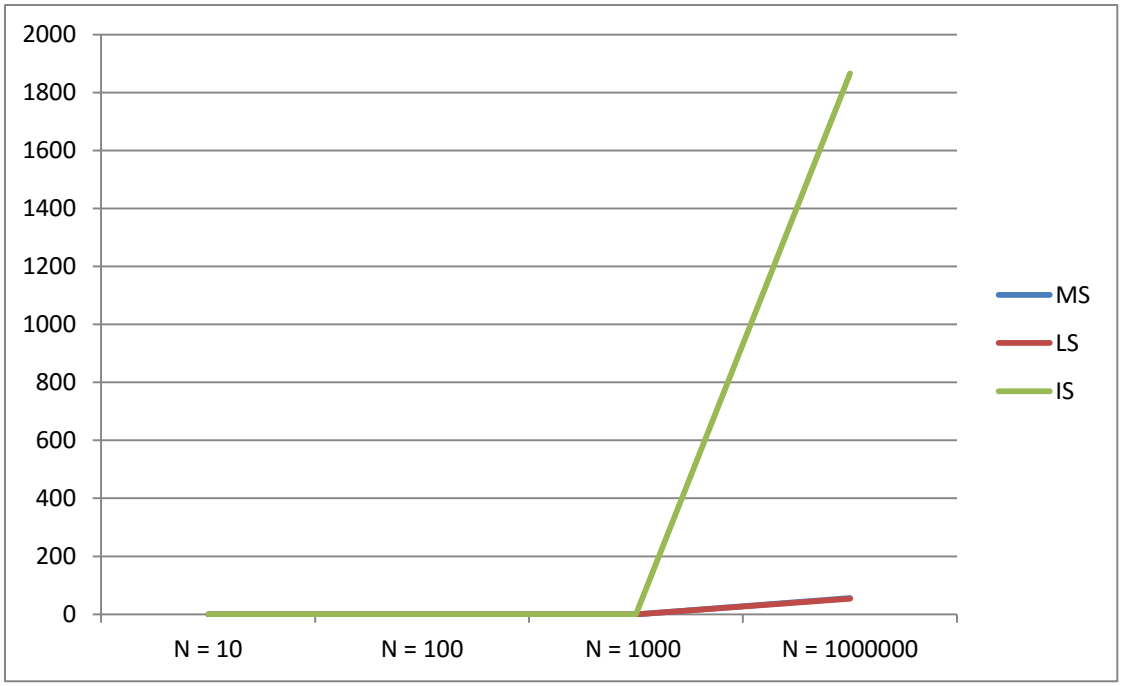
Then I decided to use my own computer, which is not very fast, and can give me some proper numbers. But the visual studio which i use loads the code a bit slow and when merge-sort dives into the lower branches, the pc got very slowed. I believe this affected the outcomes a lot.

C)Graphs

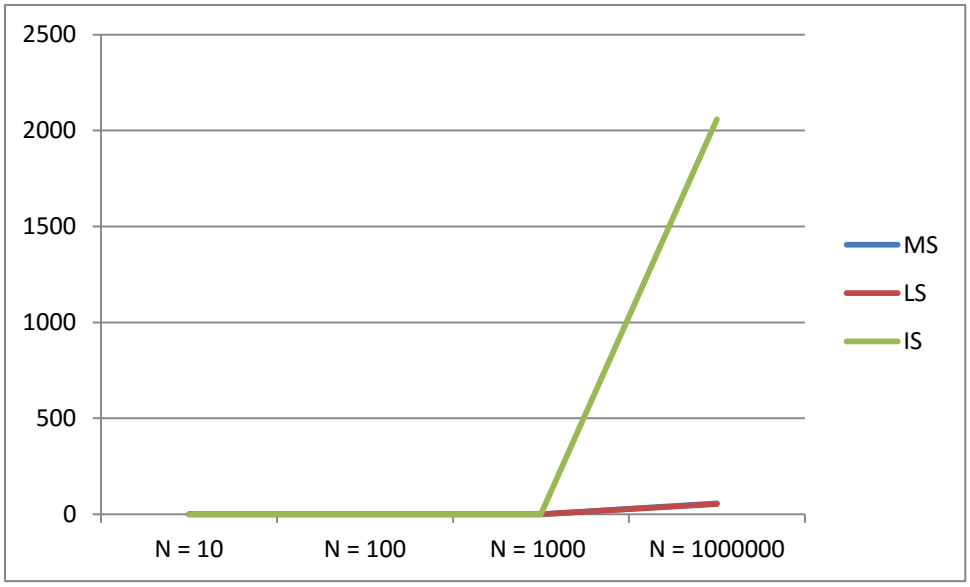
I wanted to give explanation for all data before giving graphs, because all of them are nearly equally same due to insertion sort taking a lot of time for 1000000 sorting data, and 10 100 and 1000 N values doesnt really make much difference between that.

From the graphs, linear search and merge-sort algorithms seems to be a better choice for large data amounts, and all three types seems to be giving nearly similar results for low amounts, and hence all can be used.

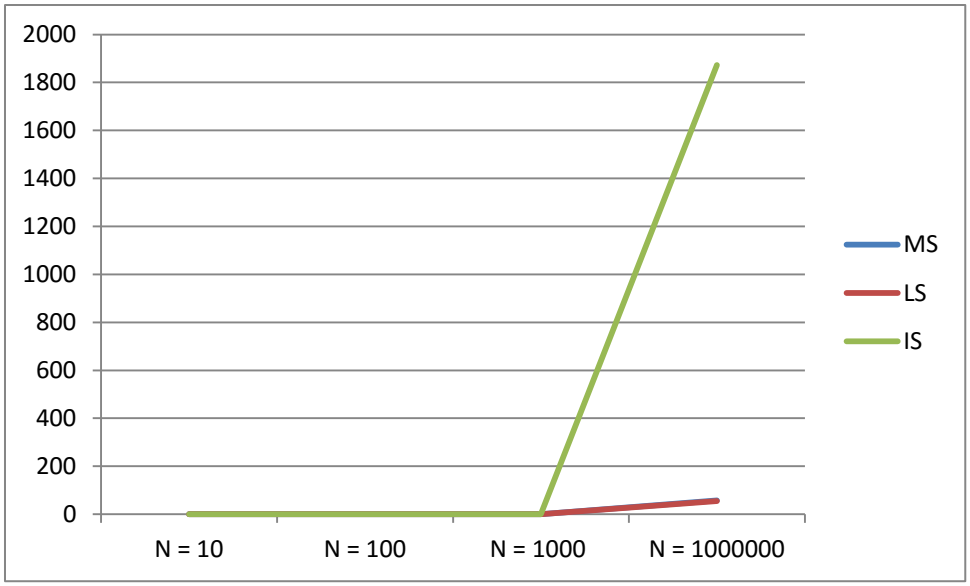
K = 1	N = 10	N = 100	N = 1000	N = 1000000
MS	0,002	0,015	0,131	55,887
LS	0,001	0,008	0,088	54,324
IS	0,002	0,016	0,089	1866



K = 2	N = 10	N = 100	N = 1000	N = 1000000
MS	0,001	0,017	0,107	57,645
LS	0,001	0,012	0,085	53,883
IS	0,002	0,013	0,135	2059



K = 10	N = 10	N = 100	N = 1000	N = 1000000
MS	0,004	0,008	0,129	57,652
LS	0,008	0,015	0,118	54,425
IS	0,005	0,018	0,09	1873



K = N / 2	N = 10	N = 100	N = 1000	N = 1000000
MS	0,002	0,006	0,09	57,749
LS	0,001	0,015	0,106	54,341
IS	0,002	0,017	0,143	2032