Tanay Cansin Dogan
150150119

BLG 335E – Homework 1

Introduction:

In this assignment we have implemented the **bubble sort** and **merge sort** algorithms. Then we analyze those implementations by using techniques which are covered in lectures called asymptotic lower bound and asymptotic upper bound. In order to apply more easily, execution tables are constructed as given:

Bubble Sort						
	Statement	Execution	Frequency		Total Steps	
			while-true	while-false	while-true	while-false
1.	int temp;	1	1	1	1	1
2.	for(int i = 1; i < N; i++){	1	N+1	N+1	N+1	N+1
3.	int j = i;	1	N	N	N	N
4.	while(j > 0 && list[j] < list[j-1]){	3	N*(N)	N	3*N*N	3*N
5.	temp = list[j];	1	N*(N-1)	0	N*(N-1)	0
6.	list[j] = list[j-1];	1	N*(N-1)	0	N*(N-1)	0
7.	list[j-1] = temp;	1	N*(N-1)	0	N*(N-1)	0
8.	j--;	1	N*(N-1)	0	N*(N-1)	0
9.	}}	-	-	-	-	-
Total:					$7N^2$-2N+2	5N+2

Asymptotic Upper Bound: $O(n^2)$
Asymptotic Lower Bound: $\Omega(n)$

Merge						
	Statement	Execution	Frequency		Total Steps	
			If-true	If-false	If-true	If-false
1.	int i = low; int j = mid;	2	1	1	2	2
2.	vector<int> temp;	1	1	1	1	1
3.	while(i < mid && j < high) {	2	N/2 + 1	N/2 + 1	N + 2	N + 2
4.	…	3	N/2	N/2	3*N/2	3*N/2
5.	}	-	-	-	-	-
	if(i >= mid){	1	1	1	1	1
.	…	3	N/2 + 1	0	3*N/2 + 1	0
.	} else{	-	-	-	-	-
.	…	3	0	N/2 + 1	0	3*N/2 + 1
10.	}	-	-	-	-	-
11.	for(int k=0; k < high - low; k++){	3	N+1	N+1	3*N + 3	3*N + 3
12.	list[low + k] = temp[k];	2	N	N	2*N	2*N
13.	}	-	-	-	-	-
Total:					9N+10	9N+10

Asymptotic Upper Bound: O(n)
Asymptotic Lower Bound: Ω(n)

Merge Sort				
	Statement	Execution	Frequency	Total Steps
1.	if(high - low > 1){	2	1	2
2.	int mid = (low + high) / 2;	2	1	2
3.	MergeSort(list, low, mid);	T(n/2)	1	T(n/2)
4.	MergeSort(list, mid, high);	T(n/2)	1	T(n/2)
5.	Merge(list, low, mid, high);	O(n)	1	O(n)
6.	}	-	-	-
Total:				2*T(n/2)+O(n)

According to the Master Theorem Case2:

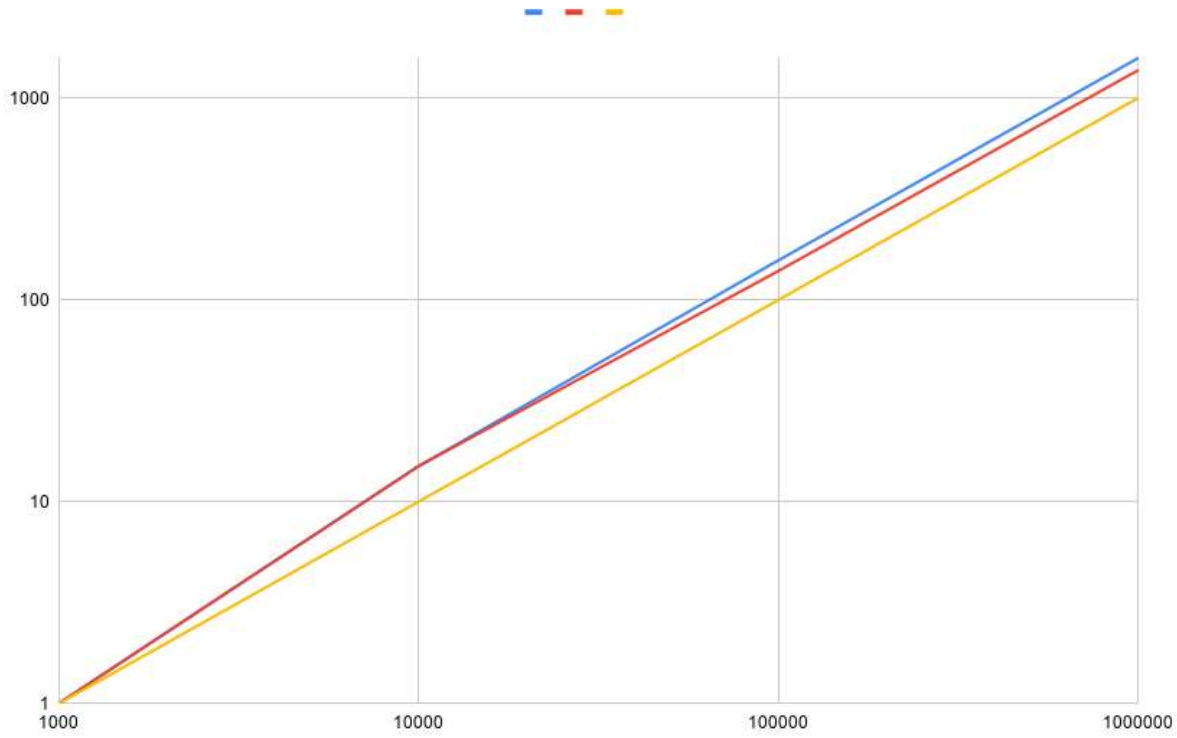
$\log_a b$ where a = 2, b = 2 , Thus:

Asymptotic Upper Bound: $O(n*\log_2 n)$

Asymptotic Lower Bound: $\Omega(n*\log_2 n)$

Clock values for each run:

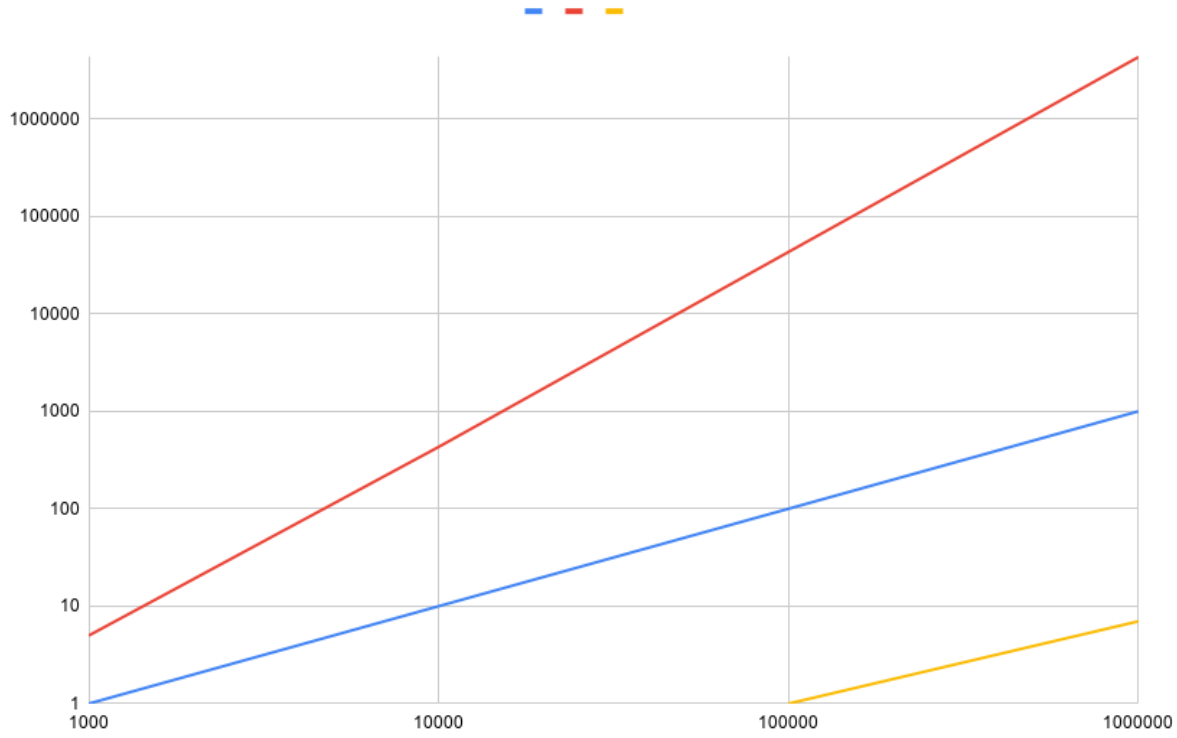
	Merge		Bubble	
	unsorted	sorted	unsorted	sorted
1000	1	1	5	0
10000	15	15	431	0
100000	157	139	43166	1
1000000	1577	1373	4316731	7



Orange line: O(n) : To show O(n) with 1 – 10 – 100 – 1000 made up sequence

Red line: sorted.txt merge sort O(n*logn)

Blue line: unsorted.txt merge sort O(n*logn)



Blue line: O(n) : To show O(n) with 1 – 10 – 100 – 1000 made up sequence
Red line: unsorted.txt Bubble sort $O(n^2)$
Yellow line: sorted.txt Bubble sort O(n)

I would prefer bubble sort for nearly sorted data. Because bubble sort converges to O(n) when data is nearly sorted. Otherwise I would prefer merge sort which is computationally less complex than bubble sort algorithm.

	Statement	Execution	Frequency	Total Steps
1.	r <- 0	1	1	1
2.	for i <- 1 to n do	1	N + 1	N + 1
3.	for j <- i+1 to n do	1	N*((N-1)/2 + 1)	N*((N-1)/2 + 1)
4.	for k <- 1 to j do	1	N*((N-1)/2)*((N-1)/2 + 1)	N*((N-1)/2)*((N-1)/2 + 1)
5.	r <- r+1;	2	N*(N-1)*(N-1)/4	N*(N-1)*(N-1)/4
6.	return r	1	1	1
Total:				$O(n^3)$

As can be seen in row 6, function increments r, N*(N-1)*(N-1)/4 times so it returns that value.