Department of Computer Engineering

BLG 335E
Analysis of Algorithm 1
Homework 1
Report

ID	*Name*	*Surname*
150140126	*Emre*	*Reyhanlıoğlu*

PART A – LOWER AND UPPER BOUNDS OF ALGORITHMS

Bubles ort has lower bound with O(n)which is linear bound and upper bound with O(n^2) which is quadratic bound.

	Statement	Steps/Execution	Frequency		Total Steps	
			If-true	If-false	If-true	If-false
1	bublesort(a, n){					
2	i <- length[A]	1	1	1	1	1
3	sorted <- False	1	1	1	1	1
4	while i is greater than 1 AND sorted is False {	1	n-1	n-1	n-1	n-1
5	sorted <- True	1	n-2	0	n-2	0
6	do for j=1 to i – 1 {	1	n*(n-2)	0	$n^2 -2n$	0
7	do if A[j] < A[j – 1] {	1	(n-1)*(n-2)	0	$n^2 -3n+2$	0
8	temp <- A[j]; A[j] <- A[j – 1];	2	(n-1)*(n-2)	0	$2n^2 -6n+4$	0
9	A[j – 1]<-temp; sorted <- False	2	(n-1)*(n-2)	0	$2n^2 -6n+4$	0
10	}					
11	}					
12	}					
Total					$6n^2 -15n+9$	n+1

Merge sort's lower bound and upper bound are same and it is O(nlogn) which is superlinear bound.

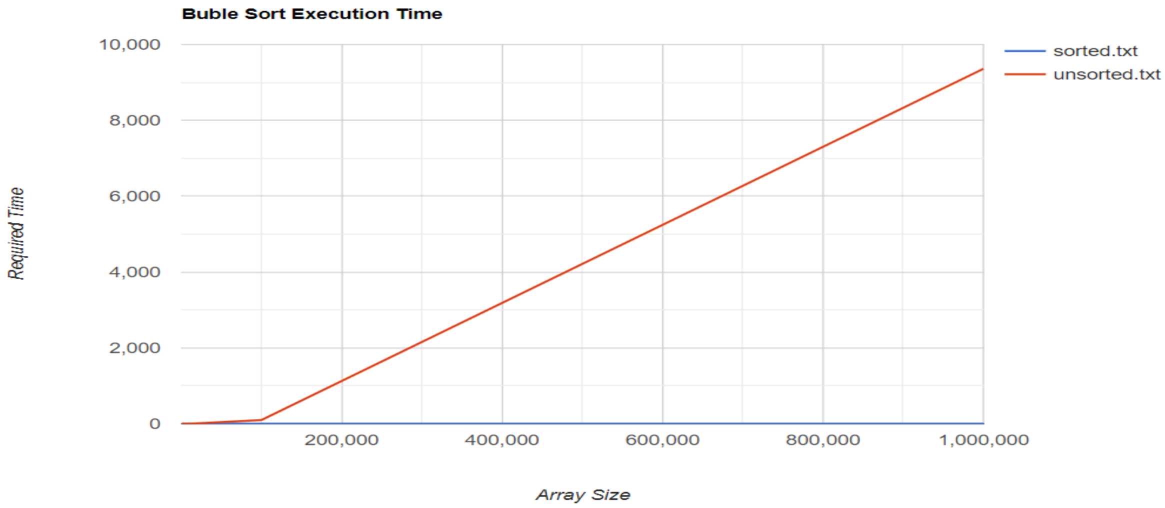
	Statement	Steps/Execution	Frequency		Total Steps	
			If-true	If-false	If-true	If-false
1	mergeSort (a, first, last)					
2	if (first < last) {	1	1	1	1	1
3	mid = (first+last)/2	1	1	0	1	0
4	mergeSort(a,first, mid);	T(n/2)	1	0	T(n/2)	0
5	mergeSort(a,mid+1, last);	T(n/2)	1	0	T(n/2)	0
6	merge(a, first, mid, last);	O(n)	1	0	O(n)	0
7	}					
Total					2*T(n/2) +O(n)	1
1	merge(a, first, mid, last) {					
2	k=first; i = first; j=mid+1	3	1	1	1	1
3	while((k<=mid) and (j<=last)) {	1	n/2 +1	n/2 +1	n/2 +1	n/2 +1
4	if(a[k]<=a[j] {	1	n/2	n/2	n/2	n/2
5	b[i] = a[k]; k++;	2	n/2	0	n	0
6	} else{					
7	b[i] = a[k]; j++;	2	0	n/2	0	n
8	} i++;	1	n/2	n/2	n/2	n/2
9	}					
10	if(k>mid) {	1	1	1	1	1
11	for(h=j to last) {	1	n/2 +1	0	n/2 +1	0
12	b[i] = a[h]; i++; }	2	n/2	0	n	0
13	}else{	1				
14	for(h=k to mid) {	1	0	n/2 +1	0	n/2 +1
15	b[i] = a[h]; i++; }	2	0	n/2	0	n
16	for(h=first to last) {	1	n+1	n+1	n+1	n+1
17	a[h] = b[h]; }	1	n	n	n	n
18	}					
Total					4n+4	4n+5

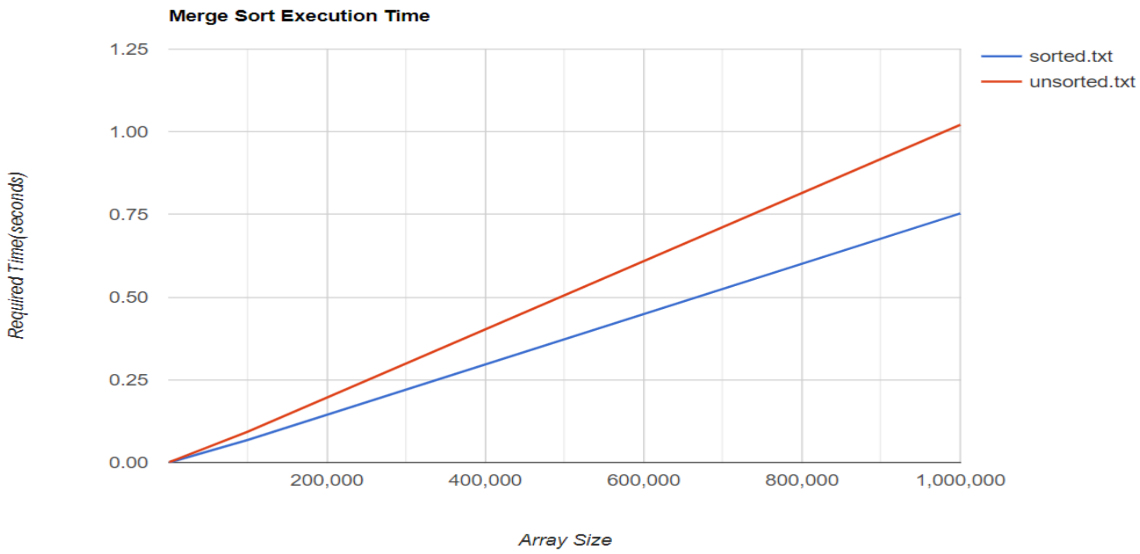
PART B – AVERAGE TIME OF EXECUTION TABLE

I have run my program for different input types and different array sizes. You can see the results below for both algorithms.

Input Type	Array Size(N)	Time	
		Buble Sort	Merge Sort
sorted	1k	0.0001 sec	0.001 sec
unsorted	1k	0.009 sec	0.001 sec
sorted	10k	0.0002 sec	0.006 sec
unsorted	10k	0.803 sec	0.009 sec
sorted	100k	0.001 sec	0.068 sec
unsorted	100k	96.344 sec	0.093 sec
sorted	1M	0.007 sec	0.753 sec
unsorted	1M	156 min	1.021 sec

PART C – GRAPHICAL RESULTS AND INTERPRETATIONS





As we can see, sorting a sorted list takes less time than unsorted list for both of these algorithms. Moreover, there is a huge gap between lower bound and upper bound of bublesort algorithm as I expected, because upper bound is quadratic and it increases much faster than lower band(linear). On the other hand, merge sort does not change much when array size increases which I expected, because merge sort has a superlinear upper and lower bound. All in all, I would prefer merge sort instead of bublesort for any data size, also I can use threads in merge sort to apply parallel programming approch and make this algoritm faster and faster.

Part D

This function is calculating 1*2+2*3+3*4+...+(n-1)*n+ which is equal to ∑ n(n+1) for n=1 to n-1.

$\sum n^2 + \sum n = (n-1)(n)(2n-1)/6 + (n-1)n/2 = [n(n-1)/2] . [(2n-1)/3 + 1]$

$= [n(n-1)/2]. [(2n+2)/3]$

$= n(n-1) (n+1)/3$

Finally;

Mystery(n) = (n-1) * n * (n+1) /3

	Statement	Steps/Execution	Frequency		Total Steps	
			If-true	If-false	If-true	If-false
1	Algorithm Mystery(n){					
2	r <- 0	1	1	1	1	1
3	for i <- 1 to n do	n+1	n+1	n+1	n+1	n+1
4	for j <- i+1 to n do	n*n	n*n	n*n	n*n	n*n
5	for k <- 1 to j do	n*(n-1)*(n+1)	n*(n-1)*(n+1)	n*(n-1)*(n+1)	n*(n-1)*(n+1)	n*(n-1)*(n+1)
6	r <- r+1;	n*(n-1)*n	n*(n-1)*n	n*(n-1)*n	n*(n-1)*n	n*(n-1)*n
7	return r	1	1	1	1	1
8	}					
Total					$2n^3+2$	$2n^3+2$

Consequently, this algorithm has $O(n^3)$ time complexity and its upper and lower bounds are same which is cubic.