Burak Bozdag
150170110

BLG335E Homework 1 Report

(a) Give the asymptotic lower bound AND upper bound on the running time for Bubble Sort and Merge Sort with the methods you covered in the lectures using the table above and show that your implementation of these algorithms fit these values.

	Statement	Steps/execution	frequency		Total Steps	
			If-true	If-false	If-true	If-false
1.	bool flag;	1	1	1	1	1
2.	while(true) {	1	N	0	N	0
3.	flag = false;	1	1	1	1	1
4.	for(int i = 0; i < N - 1; i++) {	1	N(N+1)	N+1	N(N+1)	N+1
5.	if(A[i] > A[i + 1]) {	1	N*N	0	N*N	0
6.	int temp = A[i];	1	N*N	0	N*N	0
7.	A[i] = A[i + 1];	1	N*N	0	N*N	0
8.	A[i + 1] = temp;	1	N*N	0	N*N	0
9.	flag = true;	1	N*N	0	N*N	0
10.	}	0	0	0	0	0
11.	}	0	0	0	0	0
12.	if(flag == false)	1	1	1	1	1
13.	break;	1	1	1	1	1
14.	}	0	0	0	0	0
Total	//	11	6N^2+2N+4	N + 5	6N^2+2N+4	N + 5

Figure 1 Bubble sort analysis.

	Statement	Steps/execution	frequency		Total Steps	
			If-true	If-false	If-true	If-false
1.	if(L < R) {	1	1	0	1	0
2.	int M = (L + R) / 2;	1	1	0	1	0
3.	merge(A, L, M);	1	$\log_2 N$	0	$\log_2 N$	0
4.	merge(A, M + 1, R);	1	$\log_2 N$	0	$\log_2 N$	0
5.	int arrL[M - L + 1], arrR[R - M];	2	$\log_2 N$	0	$2\log_2 N$	0
6.	for(int i = 0; i < M - L + 1; i++)	1	N/2	0	N/2	0
7.	arrL[i] = A[i + L];	1	N/2	0	N/2	0
8.	for(int i = 0; i < R - M; i++)	1	N/2	0	N/2	0
9.	arrR[i] = A[i + M + 1];	1	N/2	0	N/2	0
10.	int i = 0, j = 0, k = L;	3	3	0	9	0
11.	while(i < M - L + 1 && j < R - M) {	1	N	0	N	0
12.	if(arrL[i] <= arrR[j])	1	N	0	N	0
13.	A[k] = arrL[i++];	2	N	0	2N	0
14.	else	1	N	0	N	0
15.	A[k] = arrR[j++];	2	N	0	2N	0
16.	++k;	1	N	0	N	0
17.	}	0	0	0	0	0
18.	while(i < M - L + 1)	1	N/2	0	N/2	0
19.	A[k++] = arrL[i++];	3	N/2	0	3N/2	0
20.	while(k < R - M)	1	N/2	0	N/2	0
21.	A[k++] = arrR[j++];	3	N/2	0	3N/2	0
22.	}	0	0	0	0	0
Total	//	29	$3\log_2 N + 10N + 5$	0	$4\log_2 N + 14N + 11$	0

Figure 2 Merge sort analysis.

Bubble sort: $O(N^2)$, Merge sort: $log_2 N$.

(b) 10 points Run each search methods for each different value of N as 1000, 10000, 100000,1000000 with both of the given input files "sorted.txt" and "unsorted.txt". Calculate the average time of execution for each value of N for each file.

Bubble sort and merge sort are compared for N values 1000, 10000, 100000 and 1000000:

```
burak@Burak:/mnt/c/Users/Burak/Desktop/Analysis of Algorithms/Homework 1$ ./program b 1000 unsorted.txt
Sorting...
0.015625 seconds

Writing to the output file...
Done.
```

Figure 3 Bubble sort, N = 1000, unsorted.

```
burak@Burak:/mnt/c/Users/Burak/Desktop/Analysis of Algorithms/Homework 1$ ./program m 1000 unsorted.txt
Sorting...
0.015625 seconds

Writing to the output file...
Done.
```

Figure 4 Merge sort, N = 1000, unsorted.

```
burak@Burak:/mnt/c/Users/Burak/Desktop/Analysis of Algorithms/Homework 1$ ./program b 1000 sorted.txt
Sorting...
0 seconds

Writing to the output file...
Done.
```

Figure 5 Bubble sort, N = 1000, sorted.

```
burak@Burak:/mnt/c/Users/Burak/Desktop/Analysis of Algorithms/Homework 1$ ./program m 1000 sorted.txt
Sorting...
0 seconds

Writing to the output file...
Done.
```

Figure 6 Merge sort, N = 1000, sorted.

```
burak@Burak:/mnt/c/Users/Burak/Desktop/Analysis of Algorithms/Homework 1$ ./program b 10000 unsorted.txt
Sorting...
0.359375 seconds

Writing to the output file...
Done.
```

Figure 7 Bubble sort, N = 10000, unsorted.

```
burak@Burak:/mnt/c/Users/Burak/Desktop/Analysis of Algorithms/Homework 1$ ./program m 10000 unsorted.txt
Sorting...
0.015625 seconds

Writing to the output file...
Done.
```

Figure 8 Merge sort, N = 10000, unsorted.

Figure 9 Bubble sort, N = 10000, sorted.

Figure 10 Merge sort, N = 10000, sorted.

Figure 11 Bubble sort, N = 100000, unsorted.

Figure 12 Merge sort, N = 100000, unsorted.

Figure 13 Bubble sort, N = 100000, sorted.

Figure 14 Merge sort, N = 100000, sorted.

Figure 15 Bubble sort, N = 1000000, unsorted (it took so much time).

Figure 16 Merge sort, N = 1000000, unsorted.

```
burak@Burak: /mnt/c/Users/Burak/Desktop/Analysis of Algorithms/Homework 1$ ./program b 1000000 sorted.txt
Sorting...
0 seconds

Writing to the output file...
Done.
```

Figure 17 Bubble sort, N = 1000000, sorted.

```
burak@Burak: /mnt/c/Users/Burak/Desktop/Analysis of Algorithms/Homework 1$ ./program m 1000000 sorted.txt
Sorting...
0.109375 seconds

Writing to the output file...
Done.
```

Figure 18 Merge sort, N = 1000000, sorted.

(c) After calculating execution times for both of the files, you will prepare two-line plots (in Excel or MATLAB) for each file, in order to visualize the runtime complexity of Bubble Sort and Merge Sort for different values of N. Then you are expected to interpret the results with respect to the asymptotic bounds you have given in a. Indicate in which cases you would choose which algorithm. Why?

After executing the program as shown in the previous part of the report, I prepared two graphs for both sorted and unsorted inputs using Microsoft Excel in order to analyze and interpret two algorithms: bubble sort and merge sort. The following graphs are the results of the execution sessions:

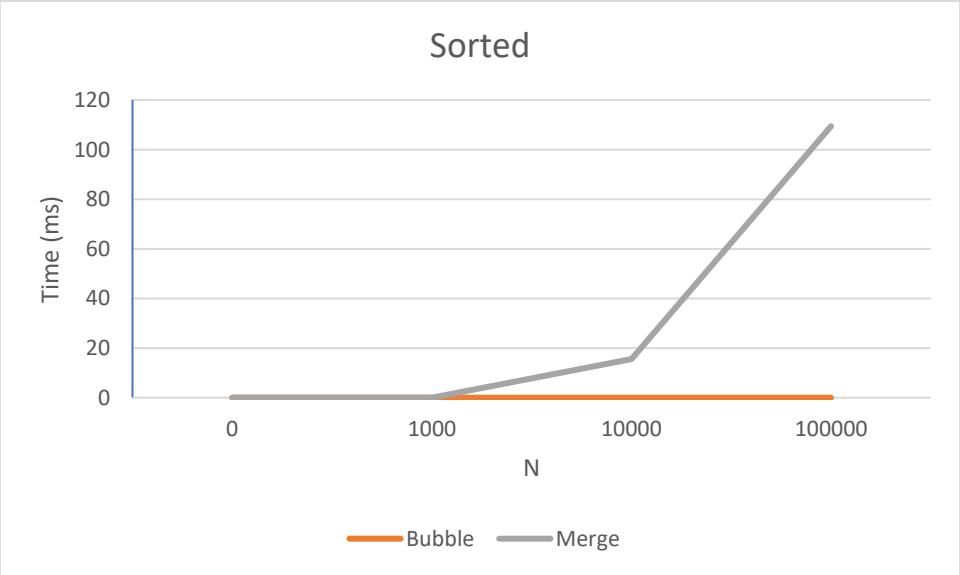

Figure 19 Sorted input plot.

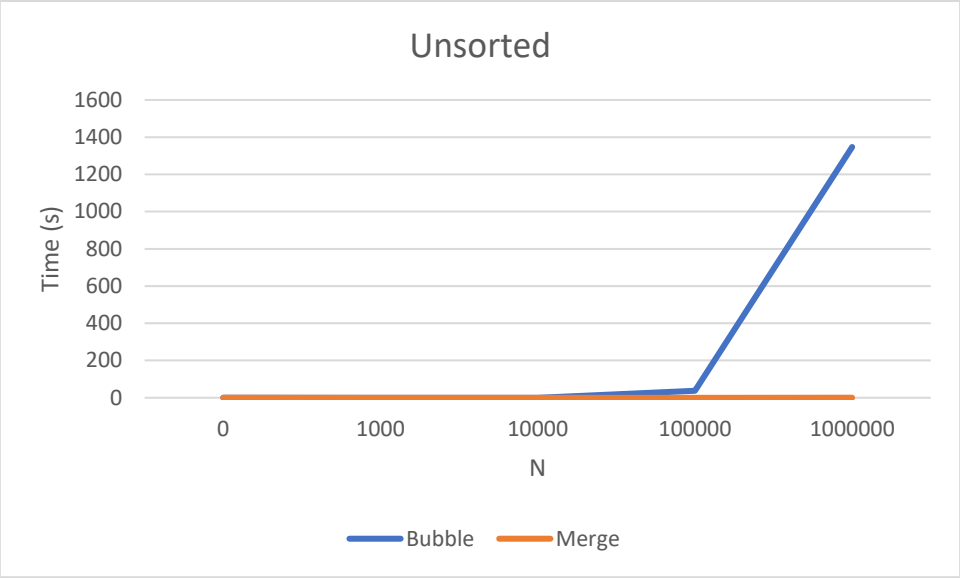

Figure 20 Unsorted input plot.

If we look at the sorted graph, we can see that merge sort time increases while bubble sort remains very low. But when the data is unsorted, there is a huge increase in bubble sort time. The reason of this is bubble algorithm works best when the input is nearly sorted, while merge sort always divides the data despite the situation of the data.

When there is a data that is nearly sorted, we can use bubble sort. But when there is a data that is so complex and unsorted, we can use divide & conquer algorithms such as merge sort and quick sort.

(d) Examine the following **Mystery** function:

1. Algorithm Mystery(n)

2. r <- 0

3. for i <- 1 to n do

4. for j <- i+1 to n do

5. for k <- 1 to j do

6. r <- r+1;

7. return r

What value is returned by the algorithm? Express your answer as a function of n. Compute its time complexity with the same method you used in a.

Burak Bozdag
150170110

The value that is returned by the function *Mystery(n)* in terms of *n* is:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} j$$

The time complexity is calculated below:

	Statement	Steps/execution	frequency		Total Steps	
			If-true	If-false	If-true	If-false
1.	r <- 0	1	1	1	1	1
2.	for i <- 1 to n do	1	n+1	n+1	n+1	n+1
3.	for j <- i+1 to n do	1	n*(n+1)/2	n*(n+1)/2	n*(n+1)/2	n*(n+1)/2
4.	for k <- 1 to j do	1	n*(n+1)*n/2	n*(n+1)*n/2	n*(n+1)*n/2	n*(n+1)*n/2
5.	r <- r+1;	1	n*(n+1)*n/2	n*(n+1)*n/2	n*(n+1)*n/2	n*(n+1)*n/2
6.	return r	1	1	1	1	1
7.	//	6	(2(n^2+3)+3n(n+1))/2	(2(n^2+3)+3n(n+1))/2	(2(n^2+3)+3n(n+1))/2	(2(n^2+3)+3n(n+1))/2

$$n^3 + \frac{3n^2}{2} + \frac{3n}{2} + 3 => O(n^3)$$