

ANALYSIS of ALGORITHMS – 1

BLG – 335E

ASSIGNMENT 1 REPORT

Name: Hakan SARAÇ

ID: 150140061

a)

	Statement	Steps/ execut ion	Frequency		Total Steps	
			If-true	If-false	If-true	If_false
	Algorithm Merge(a,low,mid,high)					
1.	{	0			0	0
2.	k:=low; i:=low; j:=mid+1;	3	1	1	1	1
3.	while((k<=mid) and (j<=high)) do	1	n/2 +1	n/2 +1	n/2 +1	n/2 +1
4.	{					
5.	if(a[k]<=a[j]) then	1	n/2	n/2	n/2	n/2
6.	{ b[i]:=a[k]; k:=k+1;}	2	n/2	0	n	0
7.	else					
8.	{b[i]:=a[j]; j:=j+1;}	2	0	n/2	0	0
9.	i:=i+1;	1	n/2	n/2	n/2	n/2
10.	}					
11.	if(k>mid) then		1	1	1	1
12.	for h:=j to high do	1	n/2 +1	0	n/2 +1	0
13.	{b[i]:=a[h]; i:=i+1; }	2	n/2	0	n	0
14.	else					
15.	for h:=k to mid do	1	0	n/2 +1	0	n/2 +1
16.	{b[i]:=a[h]; i:=i+1;}	2	0	n/2	0	n
17.	for h:=low to high do	1	n+1	n+1	n+1	n+1
18.	a[h]:=b[h];	1	n	n	n	n
19.	}					
Total					O(n)	$\Omega(n)$

Table 1. Merge Algorithm Time Complexity.

The pseudo code of merge algorithm is given in the Table 1. The upper asymptotic bound is  $O(n)$  and lower asymptotic bound is  $\Omega(n)$  as given in the 'Total' line of Table 1. This pseudo code was given in the lecture notes.

	Statement	Steps / execution	Frequency	Total steps
	Algorithm Merge-Sort(A,p,r)	0	-	0
1	if p<r then	1	1	1
2	q = Floor[(p+r)/2]	1	1	1
3	Merge-Sort(A,p,q)	x	1	x
4	Merge-Sort(A,q+1,r)	x	1	x
5	Merge(A,p,q,r)	O(n)	1	O(n)
Total				2x + O(n)+2

Table 2. Merge Sort Algorithm Time Complexity.

Pseudo code of merge sort algorithm is also was in the lecture notes. According to Table 2 (  $O(n) + 2n + 2$  ) the worst and best time complexity of merge sort is  $O(n \log n)$ . In other words, the upper asymptotic bound is  $O(n \log n)$  and the lower asymptotic bound is  $\Omega(n \log n)$  for merge sort algorithm.

	Statement	Steps/e xecutio n	Frequency		Total Steps	
			If-true	If-false	If-true	If-false
1.	i:=length[A]	1	1	1	1	1
2.	sorted:=false	1	1	1	1	1
3.	while((i>1) and (sorted==false)) do	2	n	2	2n	4
4.	{					
5.	sorted:=true;	1	n-1	1	n-1	1
6.	for j:=1 to i-1 do	1	n.(n-1)	n	n.(n-1)	n
7.	if(A[j]<A[j-1]) then	1	(n-1)(n-1)	n-1	(n-1)(n-1)	1
8.	{temp:=A[j-1]; A[j-1]:=A[j]; A[j]:=temp; sorted:=false;}	4	(n-1)(n-1)	0	4.(n-1)(n-1)	0
9.	i:=i-1;	1	n-1	1	n-1	1
10.	}					
Total					$O(n^2)$	$\Omega(n)$

Table 3. Bubble Sort Algorithm Time Complexity.

Pseudo code of bubble sort is given in the assignment. Table 3 is created according to this code. The upper asymptotic bound is  $O(n^2)$  and the lower asymptotic bound is  $\Omega(n)$ . The lower time complexity is exist when the array is sorted before algorithm.

b)

```

hakan@ubuntu: ~/algo1/hw1
hakan@ubuntu:~/algo1/hw1$ g++ assignment1.cpp -o assignment1
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 1000 sorted.txt
Time: 1.2e-05
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 1000 sorted.txt
Time: 1.4e-05
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 1000 sorted.txt
Time: 1.4e-05
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 1000 unsorted.txt
Time: 0.004619
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 1000 unsorted.txt
Time: 0.004672
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 1000 unsorted.txt
Time: 0.004807

```

Figure 1.

Average time of bubble sort on sorted.txt for n=1000 elements: 1.3e-05 s

Average time of bubble sort on unsorted.txt for n=1000 elements: 0.0047 s

```
hakan@ubuntu: ~/algo1/hw1
hakan@ubuntu:~/algo1/hw1$ g++ assignment1.cpp -o assignment1
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 10000 unsorted.txt
Time: 0.488726
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 10000 unsorted.txt
Time: 0.487144
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 10000 unsorted.txt
Time: 0.486787
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 10000 sorted.txt
Time: 4.6e-05
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 10000 sorted.txt
Time: 4.6e-05
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 10000 sorted.txt
Time: 4.6e-05
```

Figure 2.

Average time of bubble sort on sorted.txt for n=10,000 elements: 4.6e-05 s

Average time of bubble sort on unsorted.txt for n=10,000 elements: 0.487 s

```
hakan@ubuntu: ~/algo1/hw1
hakan@ubuntu:~/algo1/hw1$ g++ assignment1.cpp -o assignment1
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 100000 sorted.txt
Time: 0.000456
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 100000 sorted.txt
Time: 0.000442
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 100000 sorted.txt
Time: 0.000442
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 100000 unsorted.txt
Time: 57.9618
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 100000 unsorted.txt
Time: 58.0105
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 100000 unsorted.txt
Time: 57.9739
```

Figure 3.

Average time of bubble sort on sorted.txt for n=100,000 elements: 0.0004 s

Average time of bubble sort on unsorted.txt for n=100,000 elements: 58.0 s

```
hakan@ubuntu: ~/algo1/hw1
hakan@ubuntu:~/algo1/hw1$ g++ assignment1.cpp -o assignment1
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 1000000 sorted.txt
Time: 0.004428
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 1000000 sorted.txt
Time: 0.004444
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 1000000 sorted.txt
Time: 0.004448
```

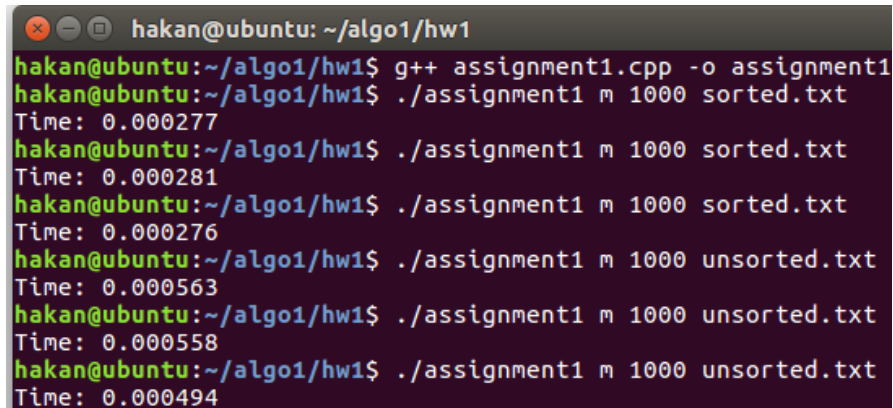
Figure 4.

Average time of bubble sort on sorted.txt for n=1,000,000 elements: 0.004 s

```
hakan@ubuntu:~/algo1/hw1$ g++ assignment1.cpp -o assignment1
hakan@ubuntu:~/algo1/hw1$ ./assignment1 b 1000000 unsorted.txt
Time: 3748.7
```

Figure 5.

Average time of bubble sort on unsorted.txt for n=1,000,000 elements: 3748.7 s

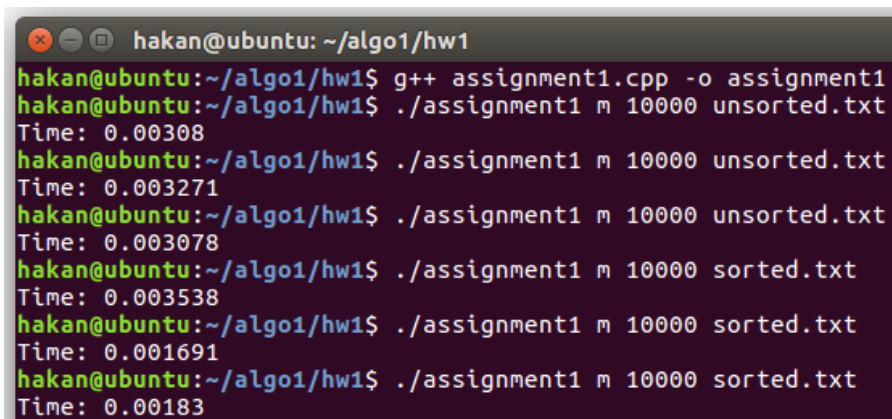


```
hakan@ubuntu: ~/algo1/hw1
hakan@ubuntu:~/algo1/hw1$ g++ assignment1.cpp -o assignment1
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000 sorted.txt
Time: 0.000277
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000 sorted.txt
Time: 0.000281
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000 sorted.txt
Time: 0.000276
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000 unsorted.txt
Time: 0.000563
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000 unsorted.txt
Time: 0.000558
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000 unsorted.txt
Time: 0.000494
```

Figure 6.

Average time of merge sort on sorted.txt for n=1000 elements: 0.00027 s

Average time of merge sort on unsorted.txt for n=1000 elements: 0.0005 s



```
hakan@ubuntu: ~/algo1/hw1
hakan@ubuntu:~/algo1/hw1$ g++ assignment1.cpp -o assignment1
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 10000 unsorted.txt
Time: 0.00308
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 10000 unsorted.txt
Time: 0.003271
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 10000 unsorted.txt
Time: 0.003078
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 10000 sorted.txt
Time: 0.003538
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 10000 sorted.txt
Time: 0.001691
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 10000 sorted.txt
Time: 0.00183
```

Figure 7.

Average time of merge sort on sorted.txt for n=10,000 elements: 0.002 s

Average time of merge sort on unsorted.txt for n=10,000 elements: 0.003 s

```
hakan@ubuntu: ~/algo1/hw1
hakan@ubuntu:~/algo1/hw1$ g++ assignment1.cpp -o assignment1
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 100000 unsorted.txt
Time: 0.037421
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 100000 unsorted.txt
Time: 0.037736
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 100000 unsorted.txt
Time: 0.03758
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 100000 sorted.txt
Time: 0.020144
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 100000 sorted.txt
Time: 0.020109
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 100000 sorted.txt
Time: 0.020127
```

Figure 8.

Average time of merge sort on sorted.txt for n=100,000 elements: 0.02 s

Average time of merge sort on unsorted.txt for n=100,000 elements: 0.037 s

```
hakan@ubuntu: ~/algo1/hw1
hakan@ubuntu:~/algo1/hw1$ g++ assignment1.cpp -o assignment1
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000000 unsorted.txt
Time: 0.441473
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000000 unsorted.txt
Time: 0.441075
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000000 unsorted.txt
Time: 0.441219
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000000 sorted.txt
Time: 0.23041
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000000 sorted.txt
Time: 0.232513
hakan@ubuntu:~/algo1/hw1$ ./assignment1 m 1000000 sorted.txt
Time: 0.231082
```

Figure 9.

Average time of merge sort on sorted.txt for n=1,000,000 elements: 0.23 s

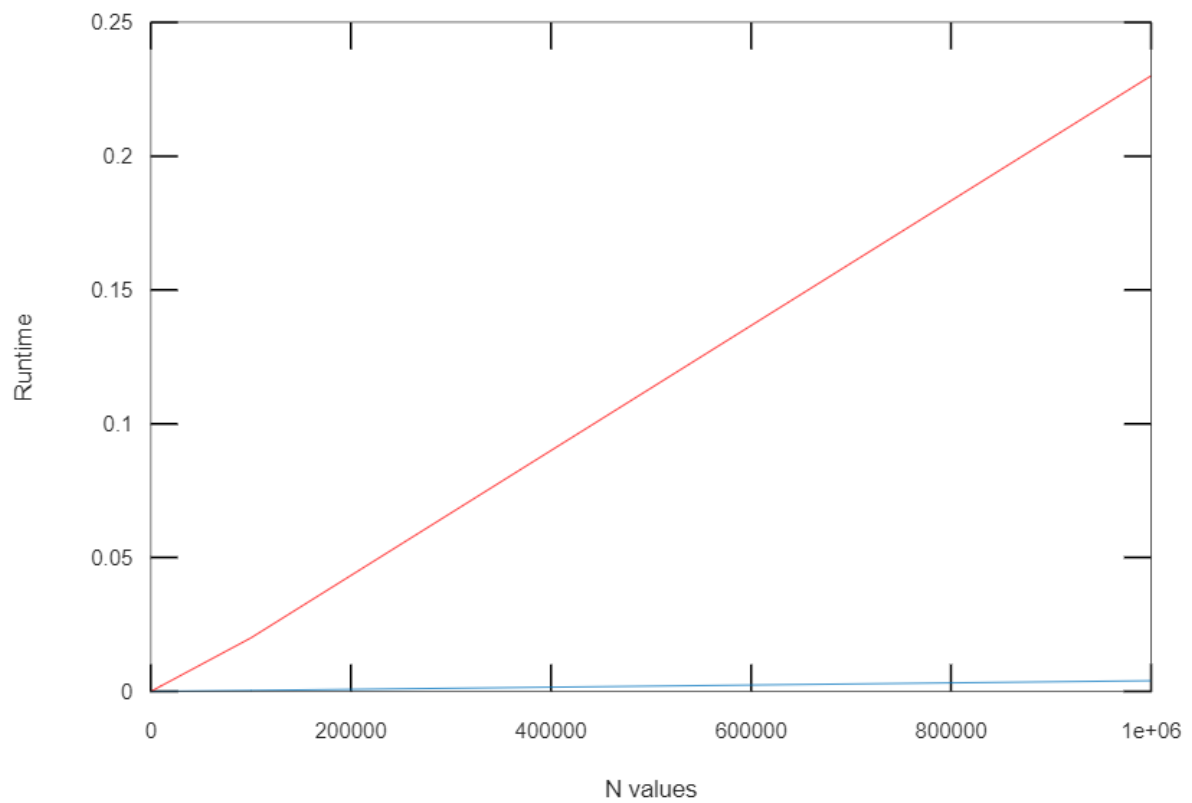
Average time of merge sort on unsorted.txt for n=1,000,000 elements: 0.44 s

c) In this part, we compare bubble sort and merge sort algorithms for unsorted and sorted and arrays that have different number of elements.

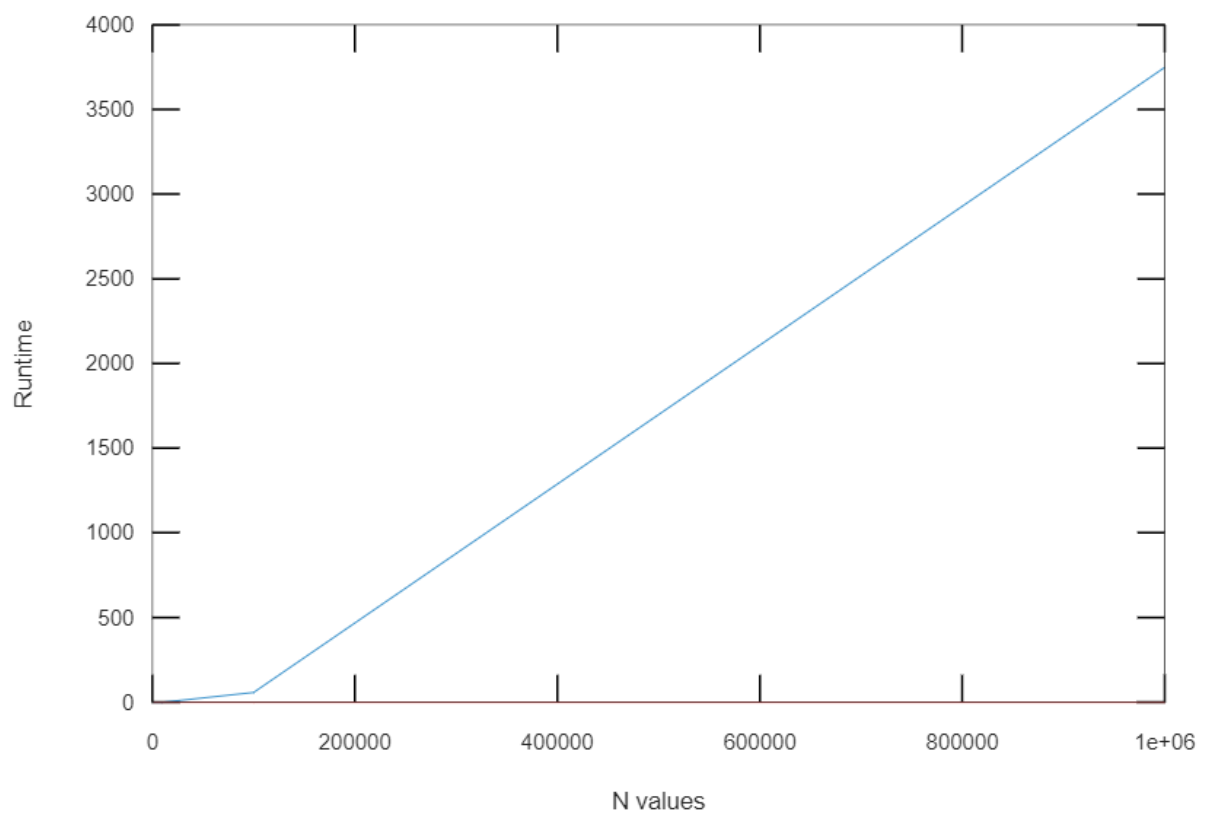
Bubble sort is more efficient than merge sort for already sorted arrays. Bubble sort have  $\Omega(n)$  time complexity for best case(already sorted arrays). On the other hand, Merge sort have  $O(n\log n)$  for every cases. The graphics of these cases is given as Figure 10.

For the unsorted arrays, merge sort is better algorithm than bubble sort. Worst case of bubble sort for time complexity is  $O(n^2)$ . Merge sort also have  $O(n\log n)$  time complexity for worst cases. The graphics of these cases for different value of arrays is given as Figure 11.

In conclusion, the better solution is bubble sort for already sorted arrays that means best cases. However, using merge sort is better idea for worst cases.



*Figure 10. Bubble(b) and Merge(r) sort for sorted arrays*



*Figure 11. Bubble(b) and Merge(r) sort for unsorted arrays*

**d)** The given mystery algorithm returns r value which conclusion of  $n*(n-1)*(n+1) / 3$ . The time complexity of mystery algorithm is given in the Table 4. The upper and the lower bound of the algorithm are equal each other and  $O(n^3)$ .

Statement		Step s/ex ecuti on	Frequency		Total Steps	
			If-true	If-false	If-true	If-false
	Algorithm Mystery(n)					
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	n*n	n*n	n*n
4	for k:=1 to j do	1	$n*(n+1)/2*$ $n*(n+1)/2$	$n*(n+1)/2*$ $n*(n+1)/2$	$n*(n+1)/2*$ $n*(n+1)/2$	$n*(n+1)/2*$ $n*(n+1)/2$
5	r:=r+1	1	$n*(n-1)*(n+1)$	$n*(n-1)*(n+1)$	$n*(n-1)*(n+1)$	$n*(n-1)*(n+1)$
6	return r	1	1	1	1	1
Total					$O(n^3)$	$O(n^3)$

Table 4. Mystery Algorithm Time Complexity.