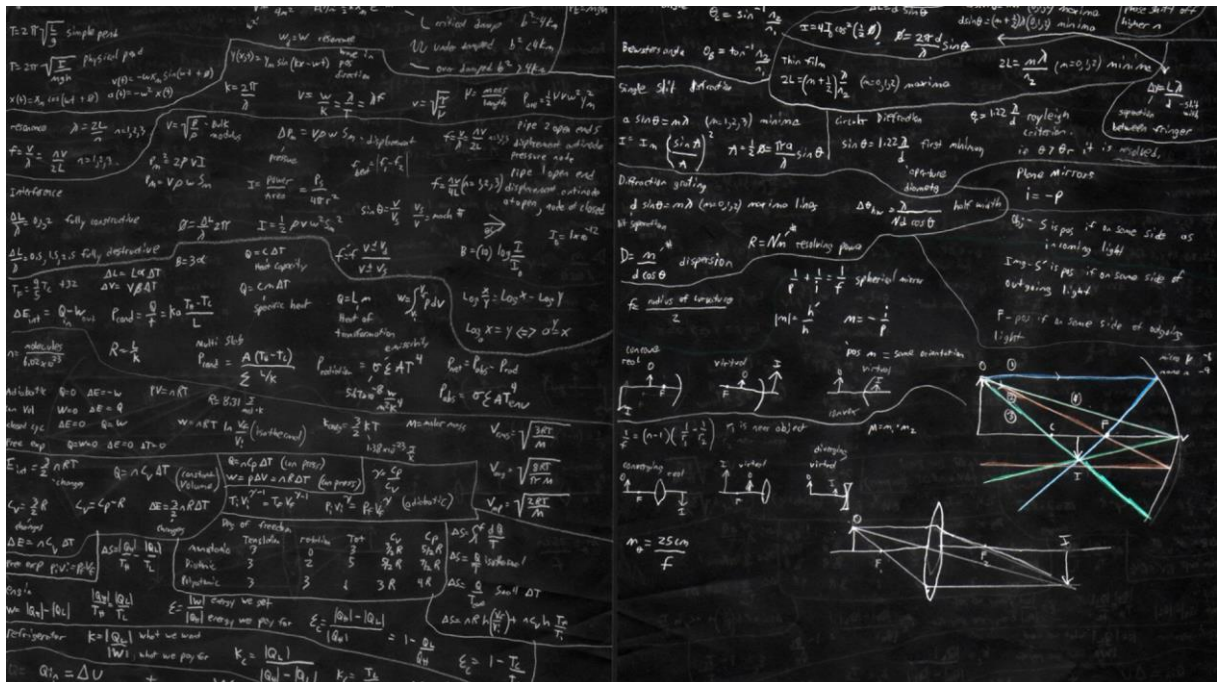




ANALYSIS of ALGORITHMS

CSE2046

ALGORITHM PROJECT REPORT




















UFUK GÜRBÜZ – 150113058

PROCEDURES

STEP 1

Designing the Experiment:

I decided to create different inputs which include every probability. I wrote a little code for creating inputs. The inputs are numbers increasing, decreasing and random. I decided the size of inputs as 1, 5, 10, 30, 50, 100 thousands, Because, this project is a performance analysis and comparison sorting algorithm project. So, i should define inputs which have different distribution orders and sizes for seeing correct results. You can find my inputs in “inputFiles” folder.

	2ser_1000Sayi_Duz.txt	20.4.2016 16:56	Metin Belgesi	6 KB
	2ser_1000Sayi_Ters.txt	21.4.2016 10:11	Metin Belgesi	6 KB
	3er_5000Sayi_Duz.txt	20.4.2016 16:56	Metin Belgesi	31 KB
	3er_5000Sayi_Ters.txt	20.4.2016 17:08	Metin Belgesi	33 KB
	5er_10000Sayi_Duz.txt	20.4.2016 16:56	Metin Belgesi	67 KB
	5er_10000Sayi_Ters.txt	21.4.2016 10:13	Metin Belgesi	67 KB
	25er_30000Sayi_Duz.txt	20.4.2016 16:57	Metin Belgesi	231 KB
	25er_30000Sayi_Ters.txt	20.4.2016 17:12	Metin Belgesi	231 KB
	50ser_50000Sayi_Duz.txt	20.4.2016 16:57	Metin Belgesi	418 KB
	50ser_50000Sayi_Ters.txt	21.4.2016 10:14	Metin Belgesi	418 KB
	100er_100000Sayi_Duz.txt	20.4.2016 16:57	Metin Belgesi	869 KB
	100er_100000Sayi_Ters.txt	21.4.2016 10:42	Metin Belgesi	869 KB
	random1000Sayi.txt	20.4.2016 16:56	Metin Belgesi	5 KB
	random5000Sayi.txt	20.4.2016 16:57	Metin Belgesi	29 KB
	random10000Sayi.txt	20.4.2016 16:58	Metin Belgesi	58 KB
	random30000Sayi.txt	20.4.2016 16:58	Metin Belgesi	192 KB
	random100000Sayi.txt	20.4.2016 16:58	Metin Belgesi	651 KB

STEP 2

Coding and Running:

I implemented all sorting algorithm (Insertion, Merge, Quick, Tree) in C language. I took input numbers from file respectively and put into a singly linked list. Because, the using linked list is faster than array. You change only linker pointers. Then, i tried all sorting algorithms using datas that in linked list step by step. The everything is okay, the algorithms are working correctly. Then, i run all algorithms for all input files and i saw these results:

```
          <<< 2'SER - 1000 SAYI - DUZ >>>
Insertion Sort Time:  0.000018      Comparison: 999
Merge Sort Time:     0.000209      Comparison: 5044
Quick Sort Time:     0.016576      Comparison: 499500
Tree Sort Time:      0.011618      Comparison: 499500
```

<<< 2'SER - 1000 SAYI - TERS >>>

Insertion Sort Time:	0.005746	Comparison:	499500
Merge Sort Time:	0.000205	Comparison:	4932
Quick Sort Time:	0.008377	Comparison:	499500
Tree Sort Time:	0.011187	Comparison:	499500

<<< 3'ER - 5000 SAYI - DUZ >>>

Insertion Sort Time:	0.000093	Comparison:	4999
Merge Sort Time:	0.001404	Comparison:	32004
Quick Sort Time:	0.713322	Comparison:	12497500
Tree Sort Time:	0.321805	Comparison:	12497500

<<< 3'ER - 5000 SAYI - TERS >>>

Insertion Sort Time:	0.191072	Comparison:	12497500
Merge Sort Time:	0.001297	Comparison:	29804
Quick Sort Time:	0.245391	Comparison:	12497500
Tree Sort Time:	0.265419	Comparison:	12497500

<<< 5'ER - 10000 SAYI - DUZ >>>

Insertion Sort Time:	0.000156	Comparison:	9999
Merge Sort Time:	0.002511	Comparison:	69008
Quick Sort Time:	1.670965	Comparison:	49995000
Tree Sort Time:	0.721082	Comparison:	49995000

<<< 5'ER - 10000 SAYI - TERS >>>

Insertion Sort Time:	0.392340	Comparison:	49995000
Merge Sort Time:	0.001314	Comparison:	64608
Quick Sort Time:	0.507087	Comparison:	49995000
Tree Sort Time:	1.434330	Comparison:	49995000

<<< 25'ER - 30000 SAYI - DUZ >>>

Insertion Sort Time:	0.000540	Comparison:	29999
Merge Sort Time:	0.008890	Comparison:	227728
Quick Sort Time:	11.981096	Comparison:	449985000
Tree Sort Time:	5.325361	Comparison:	449985000

<<< 25'ER - 30000 SAYI - TERS >>>

Insertion Sort Time:	4.661879	Comparison:	449985000
Merge Sort Time:	0.004315	Comparison:	219504
Quick Sort Time:	4.407753	Comparison:	449985000
Tree Sort Time:	6.439731	Comparison:	449985000

<<< 50'SER - 50000 SAYI - DUZ >>>

Insertion Sort Time:	0.000462	Comparison:	49999
Merge Sort Time:	0.007989	Comparison:	401952
Quick Sort Time:	29.085880	Comparison:	1249975000
Tree Sort Time:	14.787731	Comparison:	1249975000

```

<<< 50'SER - 50000 SAYI - TERS >>>
Insertion Sort Time: 9.330057      Comparison: 1249975000
Merge Sort Time:    0.007650      Comparison: 382512
Quick Sort Time:    12.356730     Comparison: 1249975000
Tree Sort Time:     15.156277     Comparison: 1249975000

```

```

<<< 100'ER - 100000 SAYI - DUZ >>>
Insertion Sort Time: 0.001191      Comparison: 99999
Merge Sort Time:    0.017900      Comparison: 853904
Quick Sort Time:    116.637609    Comparison: 704982704
Tree Sort Time:     65.870738     Comparison: 704982704

```

```

<<< 100'ER - 100000 SAYI - TERS >>>
Insertion Sort Time: 39.305529     Comparison: 704982704
Merge Sort Time:    0.017632      Comparison: 815024
Quick Sort Time:    52.216142     Comparison: 704982704
Tree Sort Time:     72.609355     Comparison: 704982704

```

```

<<< RANDOM 1000 SAYI >>>
Insertion Sort Time: 0.001126      Comparison: 248519
Merge Sort Time:    0.000200      Comparison: 8724
Quick Sort Time:    0.000196      Comparison: 10749
Tree Sort Time:     0.000162      Comparison: 11057

```

```

<<< RANDOM 5000 SAYI >>>
Insertion Sort Time: 0.044246      Comparison: 6213111
Merge Sort Time:    0.001297      Comparison: 55201
Quick Sort Time:    0.001314      Comparison: 67654
Tree Sort Time:     0.001050      Comparison: 66266

```

```

<<< RANDOM 10000 SAYI >>>
Insertion Sort Time: 0.246604      Comparison: 25173648
Merge Sort Time:    0.005843      Comparison: 120519
Quick Sort Time:    0.006638      Comparison: 150286
Tree Sort Time:     0.005094      Comparison: 153292

```

```

<<< RANDOM 30000 SAYI >>>
Insertion Sort Time: 4.470460      Comparison: 225147052
Merge Sort Time:    0.010399      Comparison: 408683
Quick Sort Time:    0.013328      Comparison: 516032
Tree Sort Time:     0.009221      Comparison: 500626

```

```

<<< RANDOM 100000 SAYI >>>
Insertion Sort Time: 50.652383     Comparison: 2495647852
Merge Sort Time:    0.043517      Comparison: 1536465
Quick Sort Time:    0.055878      Comparison: 1990578
Tree Sort Time:     0.034555      Comparison: 1769750

```

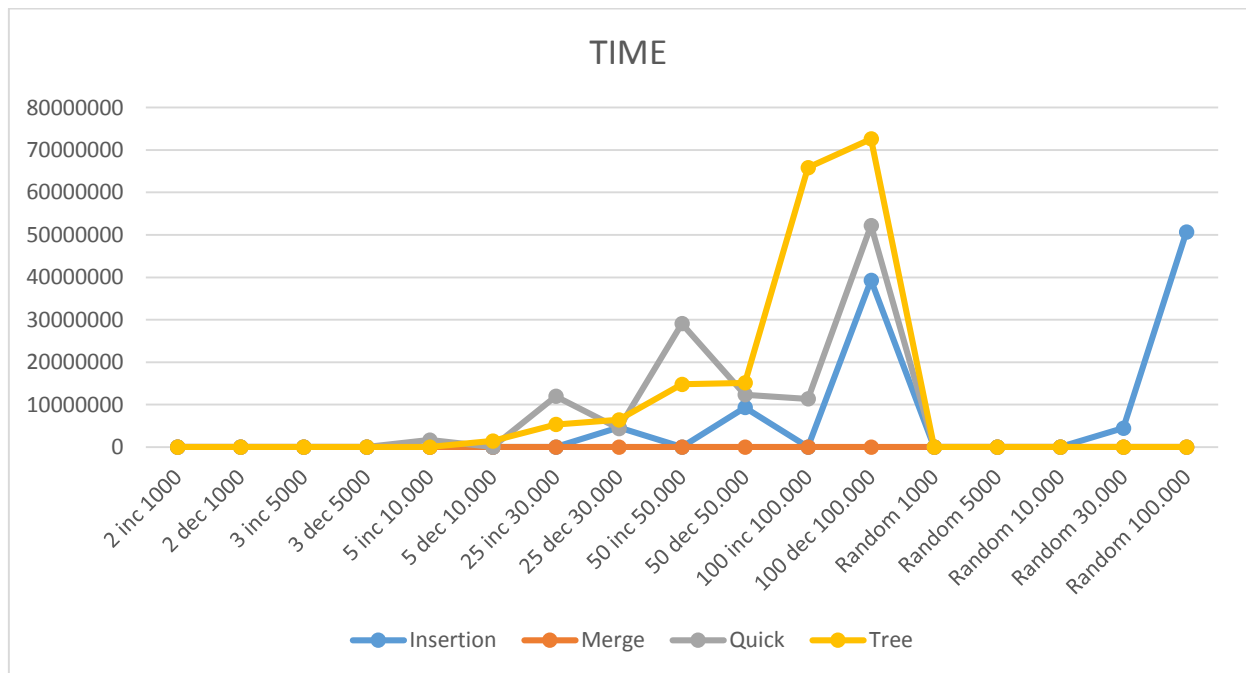
I examined these results thoroughly and calculated Big O complexity according to input size. The results are correct exactly.

STEP 3

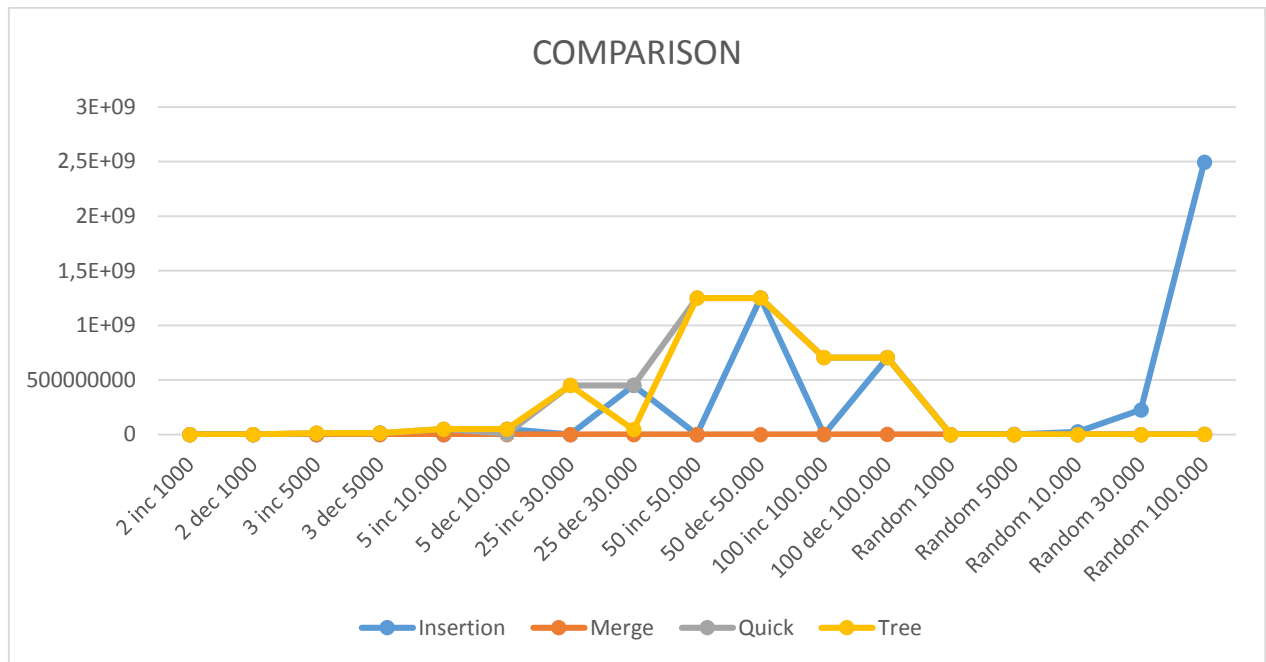
Illustrating and Analyzing Results:

a)

	TIME				COMPARISON			
SIZE	Insertion	Merge	Quick	Tree	Insertion	Merge	Quick	Tree
2 inc 1000	0.000018	0.000209	0.016576	0.011618	999	5044	499500	499500
2 dec 1000	0.005746	0.000205	0.008377	0.011187	499500	4932	499500	499500
3 inc 5000	0.000093	0.001404	0.713322	0.321805	4999	32004	12497500	12497500
3 dec 5000	0.191072	0.001297	0.245391	0.265419	12497500	29804	12497500	12497500
5 inc 10.000	0.000156	0.002511	1.670965	0.721082	9999	69008	49995000	49995000
5 dec 10.000	0.392340	0.001314	0.507087	1.434330	49995000	64608	49995000	49995000
25 inc 30.000	0.000540	0.008890	11.981096	5.325361	29999	227728	449985000	449985000
25 dec 30.000	4.661879	0.004315	4.407753	6.439731	449985000	219504	449985000	449985000
50 inc 50.000	0.000462	0.007989	29.085880	14.787731	49999	401952	1249975000	1249975000
50 dec 50.000	9.330057	0.007650	12.356730	15.156277	1249975000	382512	1249975000	1249975000
100 inc 100.000	0.001191	0.017900	113.63761	65.870738	99999	853904	704982704	704982704
100 dec 100.000	39.305529	0.017632	52.216242	72.609355	704982704	815024	704982704	704982704
Random 1000	0.001126	0.000200	0.000196	0.000162	248519	8724	10749	11057
Random 5000	0.044246	0.001297	0.001314	0.001050	623111	55201	67654	66266
Random 10.000	0.246604	0.005843	0.006638	0.00509	25173648	120519	150286	153292
Random 30.000	4.470460	0.010399	0.013328	0.009221	225147052	408683	516032	500626
Random 100.000	50.652383	0.043517	0.055878	0.034555	2495647852	1536465	1990578	1769750



I plotted time usage of algorithms using values in the table. I saw that every sorting algorithm use different time. They affect input type which has different ordering and size.



I plotted numbers of comparison of algorithms using values in the table. I saw that every sorting algorithm make different times comparison. They affect input type which has different ordering and size. So, the numbers of comparison is changing.

b) The Comparison of Sorting Algorithms

I compared all algorithm according to time usage and numbers of comparison.

The Insertion Sort: If the list is in increasing order (best case), the numbers of comparison are “ n ” (input size – $O(n)$). If the list is in decreasing order (worst case), the numbers of comparison are “ $n*(n+1)/2$ ” ($O(n^2)$) and are same with quick and tree sorting. But; if the list is in random order (average case), the numbers of comparison are between ‘best case’ and ‘worst case’ ($O(n^2)$).

The usage time of insertion sorting is **less** than (fast) other sorting when the list was in increasing order. But; when the list was in decreasing order, the insertion sort is slow according to other sorting.

The Merge Sort: The numbers of comparison are “ $n * \log n$ ” ($O(n * \log n)$) in every situation (best, worst, average cases).

The usage time of merge sorting is **the least** (the fastest) algorithm excluding in increasing order. The insertion is the fastest only in this situation.

The Quick Sort: If the list is in increasing order (best case), the numbers of comparison are “ $n * \log n$ ” (input size – $O(n * \log n)$). If the list is in decreasing order (worst case), the numbers of comparison are “ n^2 ” ($O(n^2)$) and are same tree sorting. But; if the list is in

random order (average case), the numbers of comparison are between 'best case' and 'worst case' ($O(n * \log n)$).

The usage time of quick sorting is **more** than (slow) in increasing order and in decreasing order according to insertion sorting. But, the usage time is **less** than (fast) random order according to insertion sorting. If the list is in decreasing order (worst case), it is **faster** than tree sort.

The Tree Sort: If the list is in increasing order (best case), the numbers of comparison are “ $n * \log n$ ” (input size – $O(n * \log n)$). If the list is in decreasing order (worst case), the numbers of comparison are “ n^2 ” ($O(n^2)$) and are same quick sorting. But; if the list is in random order (average case), the numbers of comparison are between ‘best case’ and ‘worst case’ ($O(n * \log n)$).

The usage time of tree sorting is **more** than (slow) in increasing order and in decreasing order according to insertion sorting. But, the usage time is **less** than (fast) random order according to insertion sorting. If the list is in increasing order (best case), it is **faster** than quicksort.

- The **Merge Sorting** is the fastest sorting algorithm excluding in increasing order situation. At that time, the insertion sort is the fastest sorting algorithm in increasing order.

c) The Comparison of The Empirical Results with Theoretical Results

I found empirical results as above table. I calculated “ Big O ” complexity of them and saw that my empirical results are correct. You can see theoretical and empirical numbers of comparison of all algorithms for input file that has 1000 numbers.

THEORETICAL				EMPIRICAL		
Sorting	Best	Worst	Average	Best	Worst	Average
Insertion	1000	1.000.000	1.000.000	999	499500	248519
	O(n)	O(n ²)	O(n ²)			
Merge	3000	3000	3000	5044	4932	8724
	O(n * log n)	O(n * log n)	O(n * log n)			
Quick	3000	1.000.000	3000	4995	499500	10749
	O(n * log n)	O(n ²)	O(n * log n)			
Tree	3000	1.000.000	3000	4932	499500	11057
	O(n * log n)	O(n ²)	O(n * log n)			
Comparisons of 1000 numbers						

Also you can see theoretical and empirical numbers of comparison of all algorithms for input file that has 30.000 numbers.

Sorting	THEORETICAL			EMPIRICAL		
	Best	Worst	Average	Best	Worst	Average
Insertion	30.000 $O(n)$	900.000.000 $O(n^2)$	900.000.000 $O(n^2)$	29.999	449.985.000	225.147.052
Merge	134.314 $O(n * \log n)$	134.314 $O(n * \log n)$	134.314 $O(n * \log n)$	227.728	4932	408683
Quick	134.314 $O(n * \log n)$	900.000.000 $O(n^2)$	134.314 $O(n * \log n)$	342.185	449.985.000	516032
Tree	134.314 $O(n * \log n)$	900.000.000 $O(n^2)$	134.314 $O(n * \log n)$	349.815	449.985.000	500626
Comparisons of 30.000 numbers						

I saw some deflections above tables. The reason of this deflection is my input scenario. I can't choose the best and the worst case situations. But, i approached these situations.

The theoretical results can't be absolute right. It is a only mathematical expression. Namely, it shows the numbers of comparison according Big O complexity. But, the empirical results are different from theoretical results. Because, the features of systems are different in every computer. The results can affect some cases such as microprocessor, main memory, operating system and programs that running in the background. So, we should decrease these effects to minimize.

REFERENCES

- Introduction to Design and Analysis of Algorithms, Anany Levitin, 3rd Edition
- https://en.wikipedia.org/wiki/Sorting_algorithm
- <http://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html>
- <https://www.cs.cmu.edu/~adamchik/15-121/lectures/Sorting%20Algorithms/sorting.html>
- <http://www.sorting-algorithms.com/>
- <https://www.youtube.com/watch?v=ZZuD6iUe3Pc>
- Experience of Assistant Prof. Ömer Korçak ☺