Department of Computer Science

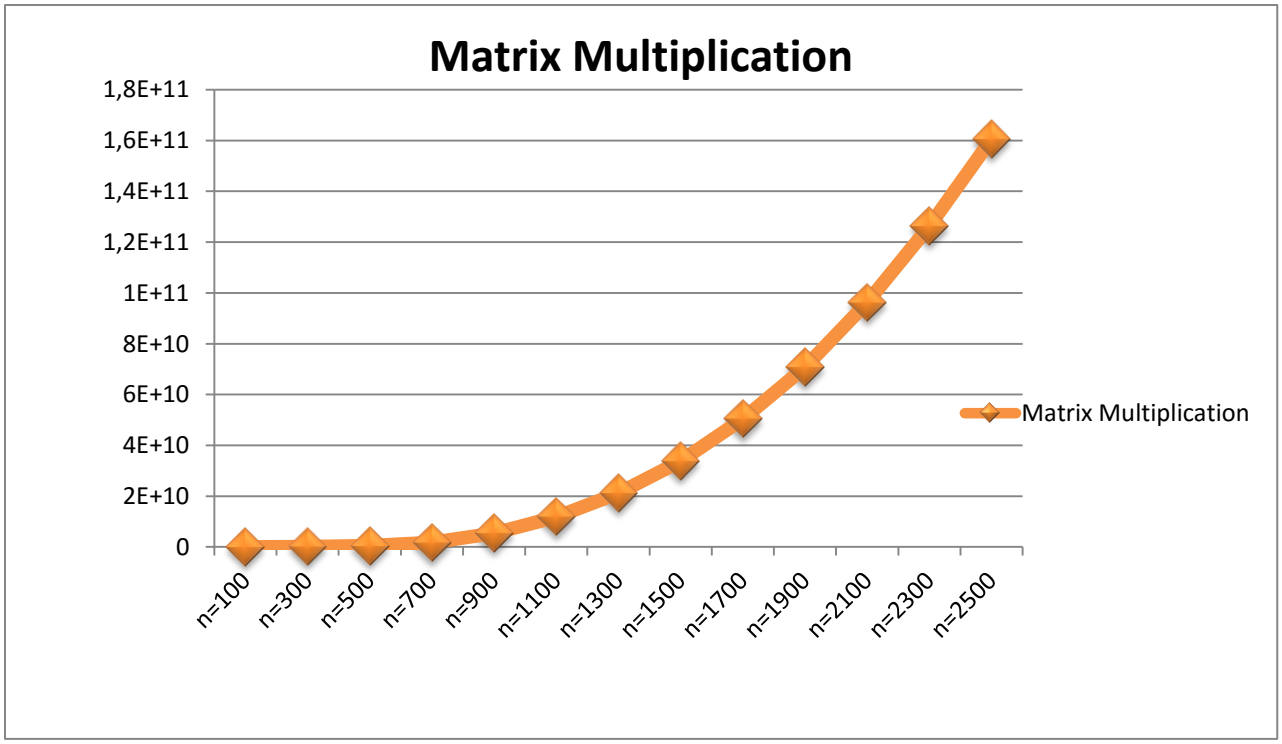
- **Name Surname : Orhan YILMAZ**
- **Number : 21328595**
- **Course : BBM204**
- **Experiment : Analysis of Algorihtms**

Introduction

In assigment-1 , we have 5 different problems. They are, matrix multiplication, bubble sort, finding maximum element, merge sort and binary search. Their algorithms are already written with natural language. Our purpose is converting to programming language, analyze their working time and showing in graph and table.

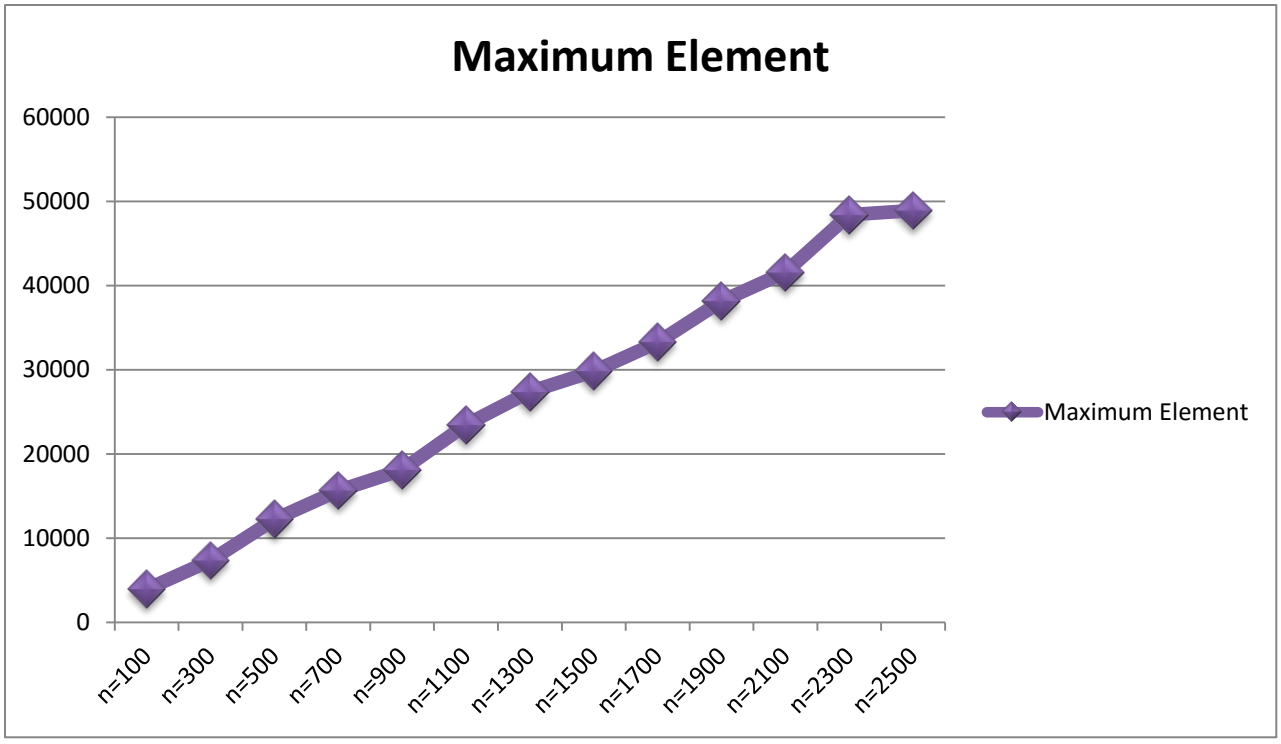
Algorithms/n	*100*	*300*	*500*	*700*	*900*	*1100*	*1300*	*1500*	*1700*	1900	*2100*	*2300*	*2500*
Matrix Multiplication	18M	23M	56M	1.6B	5.3B	11.5B	19.9B	32.5B	49.2B	71.1B	94.1B	122B	156B
Bubble Sort	378K	4M	3.6M	2.1M	3.4M	3.9M	3.8M	4.6M	6.1M	6.2M	7.9M	9.7M	11M
Finding Maximum Element	3.9K	7.3K	11.2K	16.6K	19K	26K	29.3K	30.7K	31.6K	39K	41.9K	47.3K	49.8K
Merge Sort	147K	466K	790K	1M	1M	1.3M	1.4M	1.5M	1.7M	1.7M	1.8M	1.9M	2M
Binary Search	638	673	905	992	974	1.1K	1.1K	1.3K	1.3K	1.4K	1.6K	1.8K	1.8K

(values are type of nanosecond, K for thousand, M for million, B for billion)



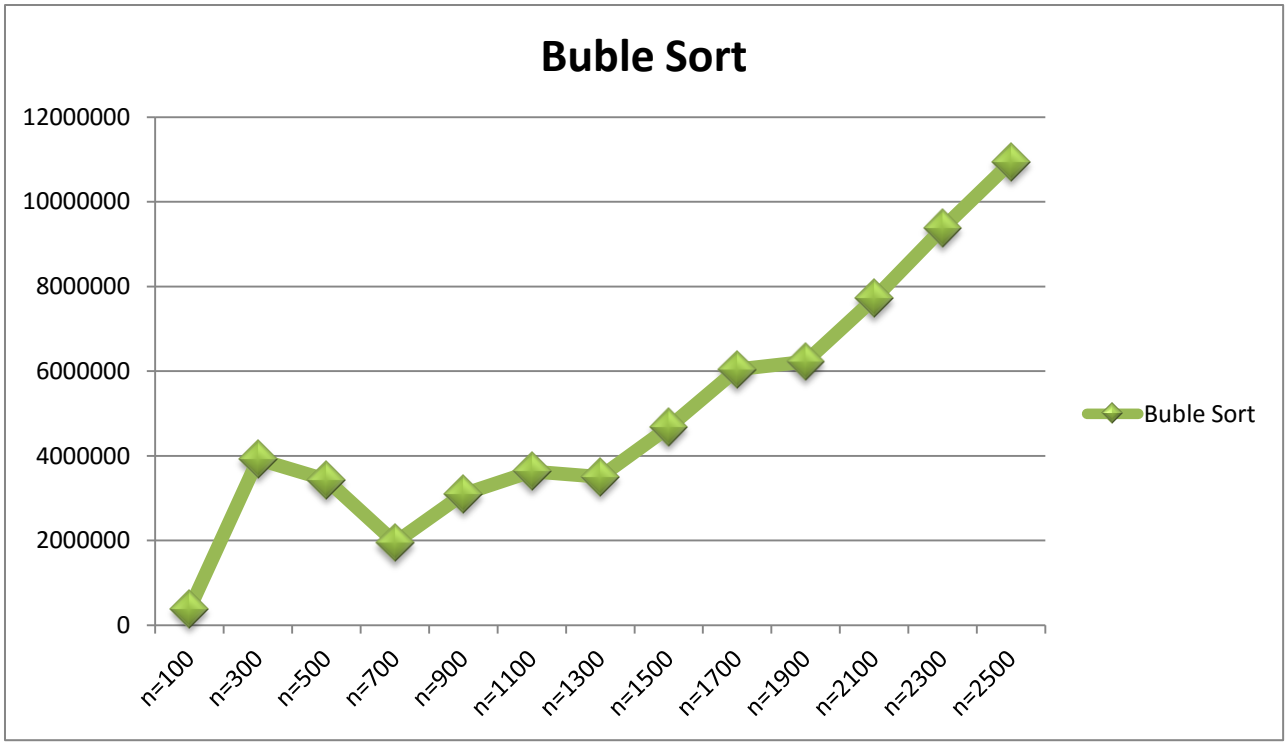
Code	Unit Cost	Times
`int[ ] [ ] matrix3 = new int[ ] [ ];`	c1	1
`for(int i=1 ; i <= n ; i++){`	c2	n
`for(int j=1 ; j <= n ; j++){`	c3	n^2
`matrix3[i][j] = 0;`	c4	n^2
`for(int k=1 ; k <=n ; k++){`	c5	n^3
`matrix3[i][j] = matrix3[i][j] + matrix1[i][k] * matrix2[k][j];`	c6	n^3
`}`		
`}`		
`}`		

Total Cost = c1 + c2 * n + c3 * n + c4 * n^2 + c5 * n^3 + c6 * n^3The time required for this algorithm is proportional to n^3 which is determined as growth rate and it is usually denoted as **O(n^3)**.



Code	Unit Cost	Times
`int[ ] array = new int[ ];`	c1	1
`int maximum_element = 0;`	c2	1
`maximum_element = array[0];`	c3	1
`for(int j=1 ; j <= n ; j++){`	c4	n
`if(array[j] > maximum_element){`	c5	n
`maximum_element = array[j];`	c6	1 -- n
`}`		
`}`		

Total Cost = c1 + c2 + c3 + c4 * n + c5 * n + c6 * (1 -- n)The time required for this algorithm is proportional to n which is determined as growth rate and it is usually denoted as **O(n)**.



Code	Unit Cost	Times
`int[ ] array = new int[ ];`	c1	1
`int swaps = 0;`	c2	1
`int temp = 0;`	c3	1
`for(int j=1 ; j <= n ; j++){`	c4	n
`swaps = 0 ;`	c5	n
`for(int k = 0 ; k <= n – j ; k++){`	c6	(n^2-n)/2
`if(array[k] > array[k+1]){`	c7	(n^2-n)/2
`temp = array[k];`	c8	1 -- (n^2-n)/2
`array[k] = array[k+1];`	c9	1 -- (n^2-n)/2
`array[k+1] = temp;`	c10	1 -- (n^2-n)/2
`swaps = swaps+1;`	c11	1 -- (n^2-n)/2

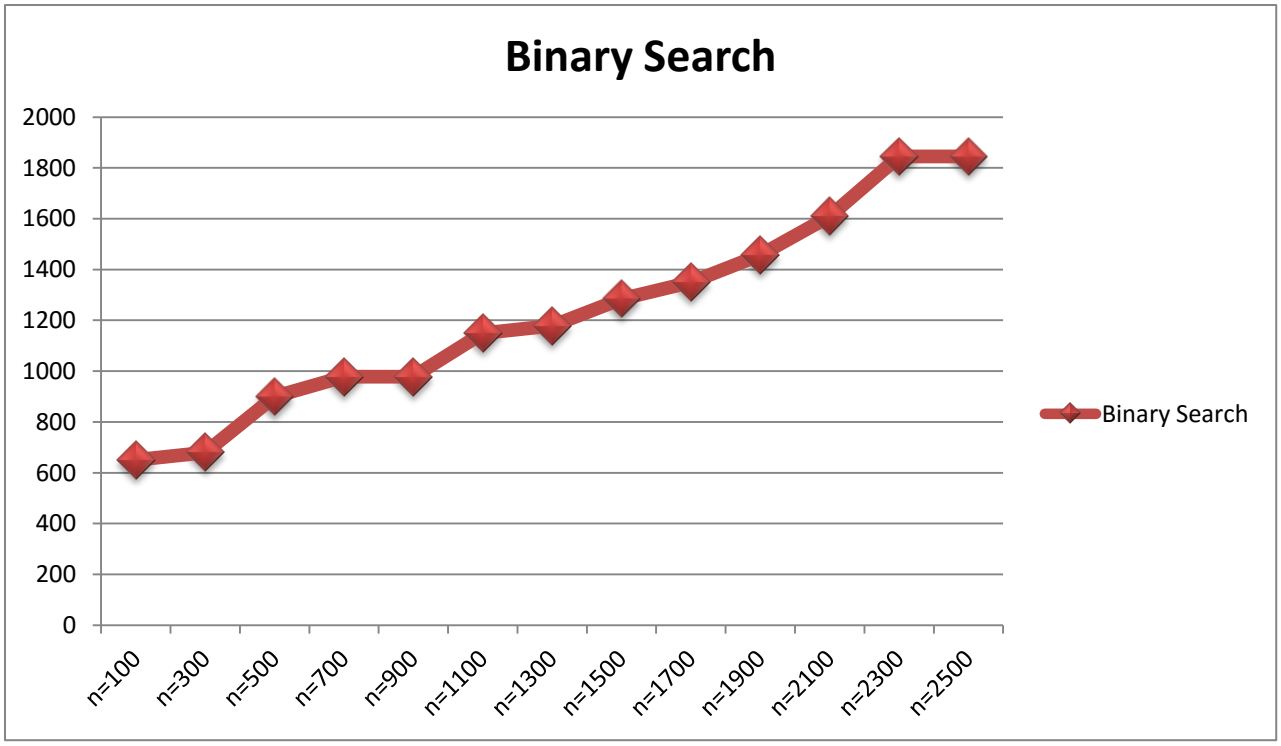
```
            }
        }
        if (swaps==0)                                   c12    1 -- (n^2-n)/2
            break;                                      c13          1
}
```

Total Cost = c1 + c2 + c3 + c4 * n + c5 * n + c6 * ((n^2-n)/2) + c7 * ((n^2-n)/2) +

c8 * (1 -- (n^2-n)/2) + c9 * (1 -- (n^2-n)/2) + c10 * (1 -- (n^2-n)/2) +

c11 * (1 -- (n^2-n)/2) + c12 * (1 -- (n^2-n)/2) + c13

The time required for this algorithm is proportional to n^2 which is determined as growth rate and it is usually denoted as **O(n^2)**.



Code	Unit Cost	Times
`int[ ] array = new int[ ];`	c1	1
`int left = 0;`	c2	1
`int right = 0;`	c3	1
`int mid = 0;`	c4	1
`left = array[0];`	c5	1
`right = array[n];`	c6	1
`while(left <= right){`	c7	1 --log n
`mid = (int)((right -left) /2)+left;`	c8	1 -- logn
`if( array[mid] == value){`	c9	1 -- logn
`return mid;`	c10	1
`}`		
`if(value < array[mid]){`	c11	1 --log n
`right = mid - 1;`	c12	1 – log n

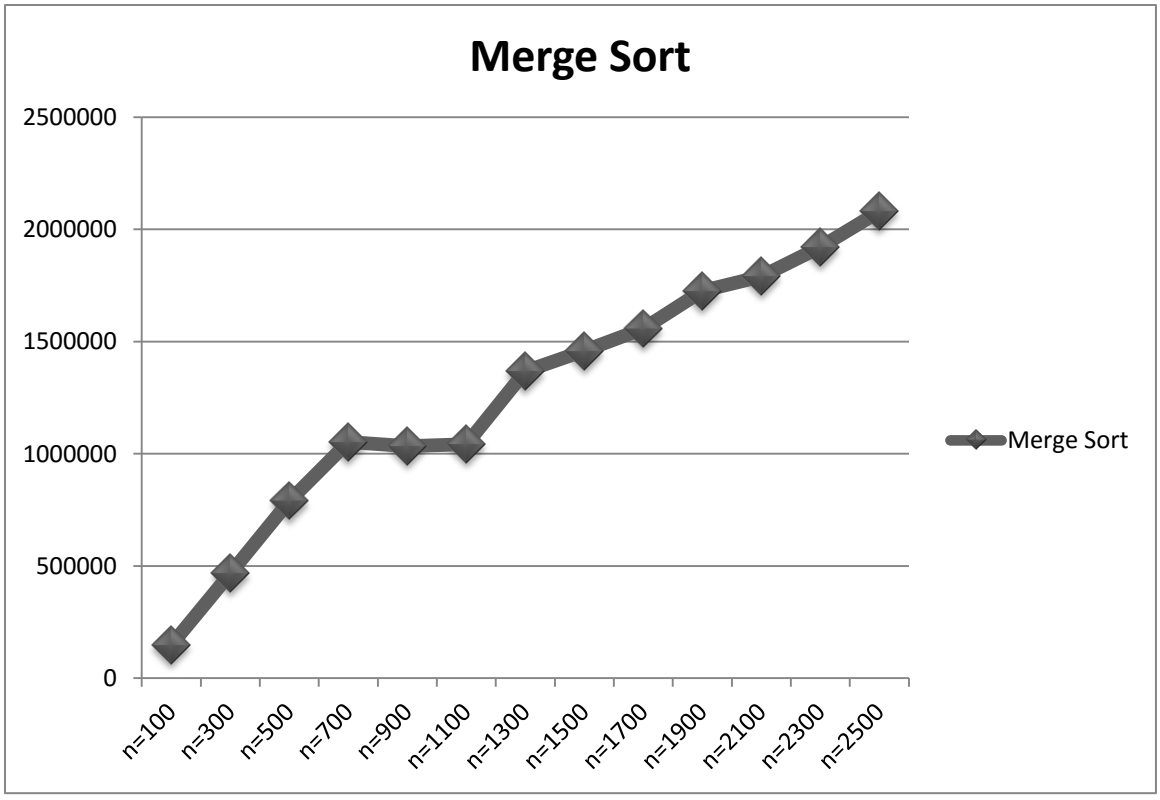
```
        }
        else{                                   c13     1 --log n
            left = mid + 1;                     c14     1 --log n
        }
}
```

Total Cost = c1 + c2 + c3 + c4 + c5 + c6 + c7 * (1 --log n) + c8 * (1 --log n) +

c9 * (1 – log n)+ c10 + c11 * (1 – log n) + c12 * (1 --log n) +

c13 * (1 --log n) + c14 * (1 --log n)

The time required for this algorithm is proportional to log n which is determined as growth rate and it is usually denoted as **O(log n)**.



Code **Unit Cost** **Times**

Comments

In assignment-1 , we worked on some algorithm's performance and we worked on running time. And maybe we learned how they can much faster.

For anaylze, we seperate them line by line and we wrote line's costs.We took growth rate among the all lines and we say this is worst case, this is running time. We learned calculation running time in course before this assignment but i was not sure everything is clear for me ,now i am sure of it.

We don't compare algorithm's running time except merge and buble sort. Because these three algorithms' problems are diffrent from each other and there isn't better or faster function in there . This compare , this sentence is meanless.

For the merge and bubble sort compare, i can say merge sort is faster than bubble sort. Because merge sort running time is nlogn , bubble sort is n^2. But i couldn't show that in my report , because i had some issues with merge sort algorithm and couldn't write runnig time in my report. But i know how it works and it's running time.

I learned that about algorithms , every unnecessary lines or code pieces are effects running time. To be a good programmer , i must choose too effective algorithm .