Marco Zuniga
COMP 3270 (Algorithms)
Programming Assignment
July 9, 2018

Introduction:

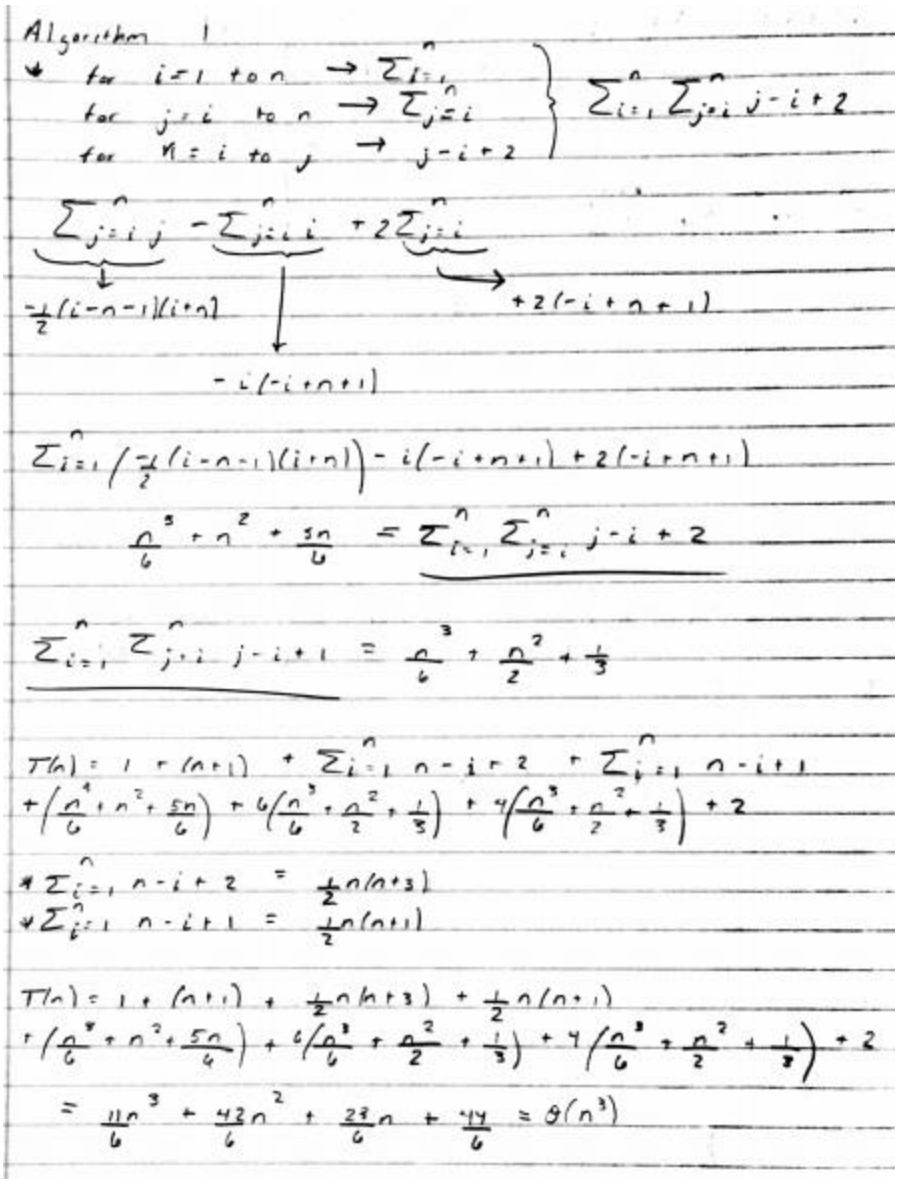
The student was required to use the knowledge based off past homework assignments to analyze the complexities of multiple algorithms both theoretically and empirically. All algorithms return the biggest sum in a subsequence of numbers.

Theoretical/Empirical Analysis:

```
Algorithm-1(X : array[P..Q] of integer)
1       maxSoFar = 0
2       for L = P to Q
3           for U = L to Q
4               sum =0
5               for I = L to U
6                   sum = sum + X[I]
                    /* sum now contains the sum of X[L..U]  */
7               maxSoFar = max (maxSoFar, sum)
8       return maxSoFar
```

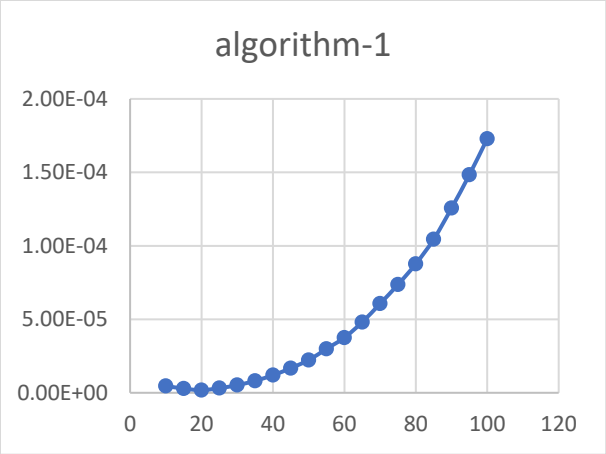
The first algorithm can be found above. The student calculated the cost for executing each step as well as the number of times each step will be executed. These calculations can be found below.

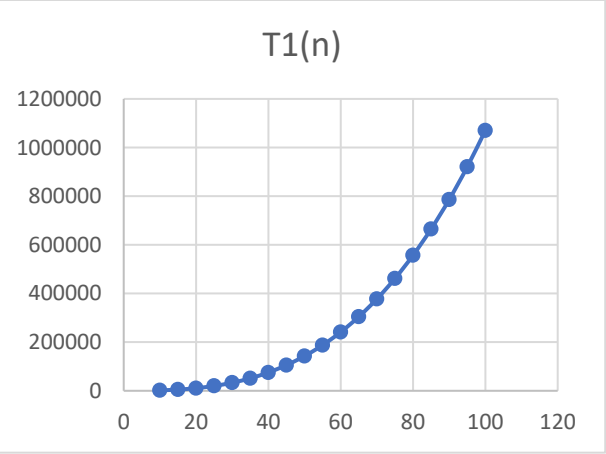
Step	Cost of each execution	Total # of times executed
1	1	1
2	1	n+1
3	1	$\sum_{i=1}^{n} n-i+2$
4	1	$\sum_{i=1}^{n} n-i+1$
5	1	$\sum_{i=1}^{n} \sum_{j=i}^{n} j-i+2$
6	6	$\sum_{i=1}^{n} \sum_{j=i}^{n} j-i+1$
7	4	$\sum_{i=1}^{n} \sum_{j=i}^{n} j-i+1$
8	2	1



$T_1(n) = \frac{11}{6}n^3 + \frac{42}{6}n^2 + \frac{23}{6}n + \frac{44}{6} = \Theta(n^3)$

The student concluded that it should be theta because the algorithm will perform all steps when executed. All values will be processed as opposed to an algorithm that is trying to find a specific value and terminating. After calculating the theoretical complexity of algorithm 1, the student ran the algorithm 1000 times and plotted the average times it took the algorithm to find the biggest sum. The algorithm began with an Array of size 10 and incremented by 5 until it ran an Array of size 100. This experiment was performed on all algorithms. One can find a scatter plot for both the empirical and theoretical analysis.



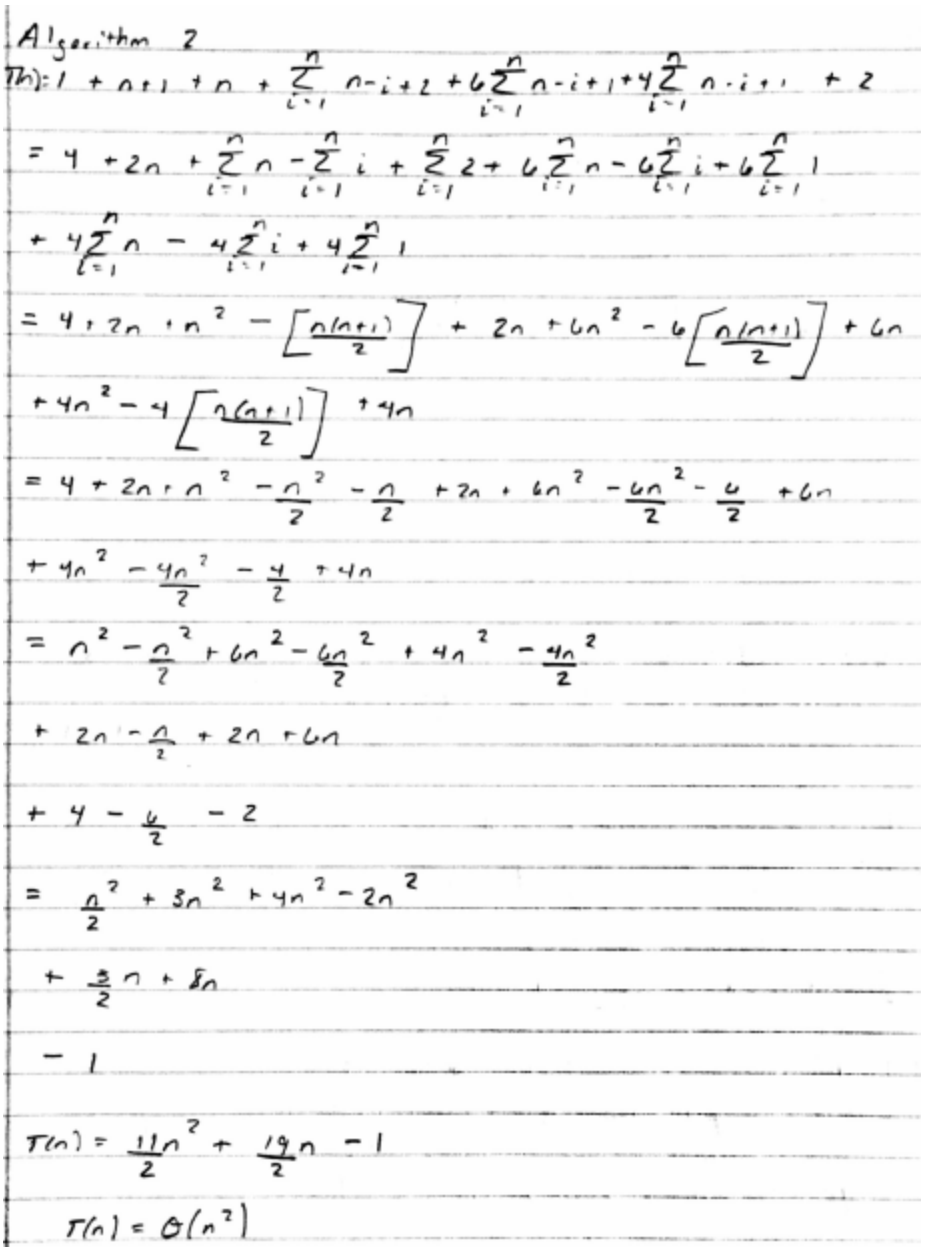


By observing the two graphs, one will find that both graphs grow at the same rate. This confirms that algorithm 1's complexity order is $\Theta(n^3)$.

```
Algorithm-2(X : array[P..Q] of integer)
1       maxSoFar = 0
2       for L = P to Q
3           sum =0
4           for U = L to Q
5               sum = sum + X[U]
                /* sum now contains the sum of X[L..U] */
6               maxSoFar = max(maxSoFar,sum)
7       return maxSoFar
```

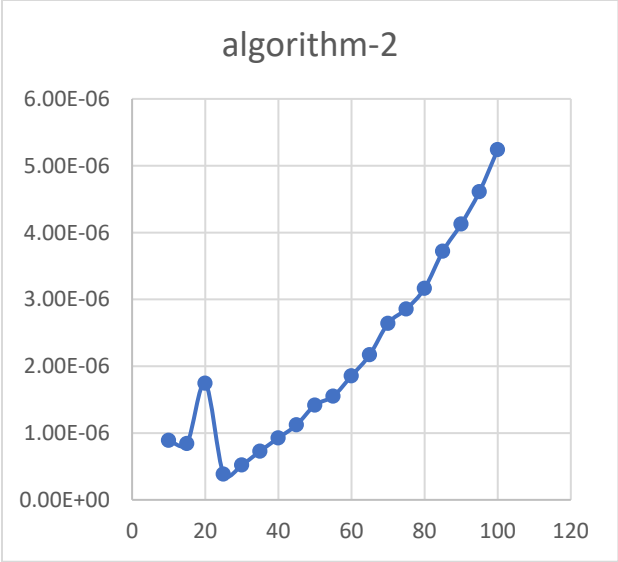
The second algorithm can be found above. All calculations/graphs can be found below.

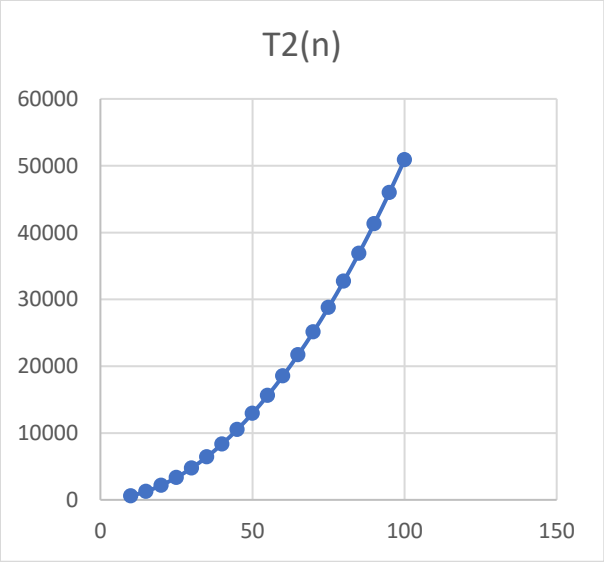
Step	Cost of each execution	Total # of times executed
1	1	1
2	1	n+1
3	1	n
4	1	$\sum_{i=1}^{n} n-i+2$
5	6	$\sum_{i=1}^{n} n-i+1$
6	4	$\sum_{i=1}^{n} n-i+1$
7	2	1



$T_2(n) = \frac{11}{2}n^2 + \frac{19}{2}n - 1 = \Theta(n^2)$

Theta-notation was used because all steps will be executed.





The two graphs show that they grow at the time rate meaning algorithm 2's order complexity is indeed $\Theta(n^2)$. It can also be seen that the rate of growth for algorithm 2 grows similar but slower than algorithm 1 (i.e. $n^3 > n^2$).

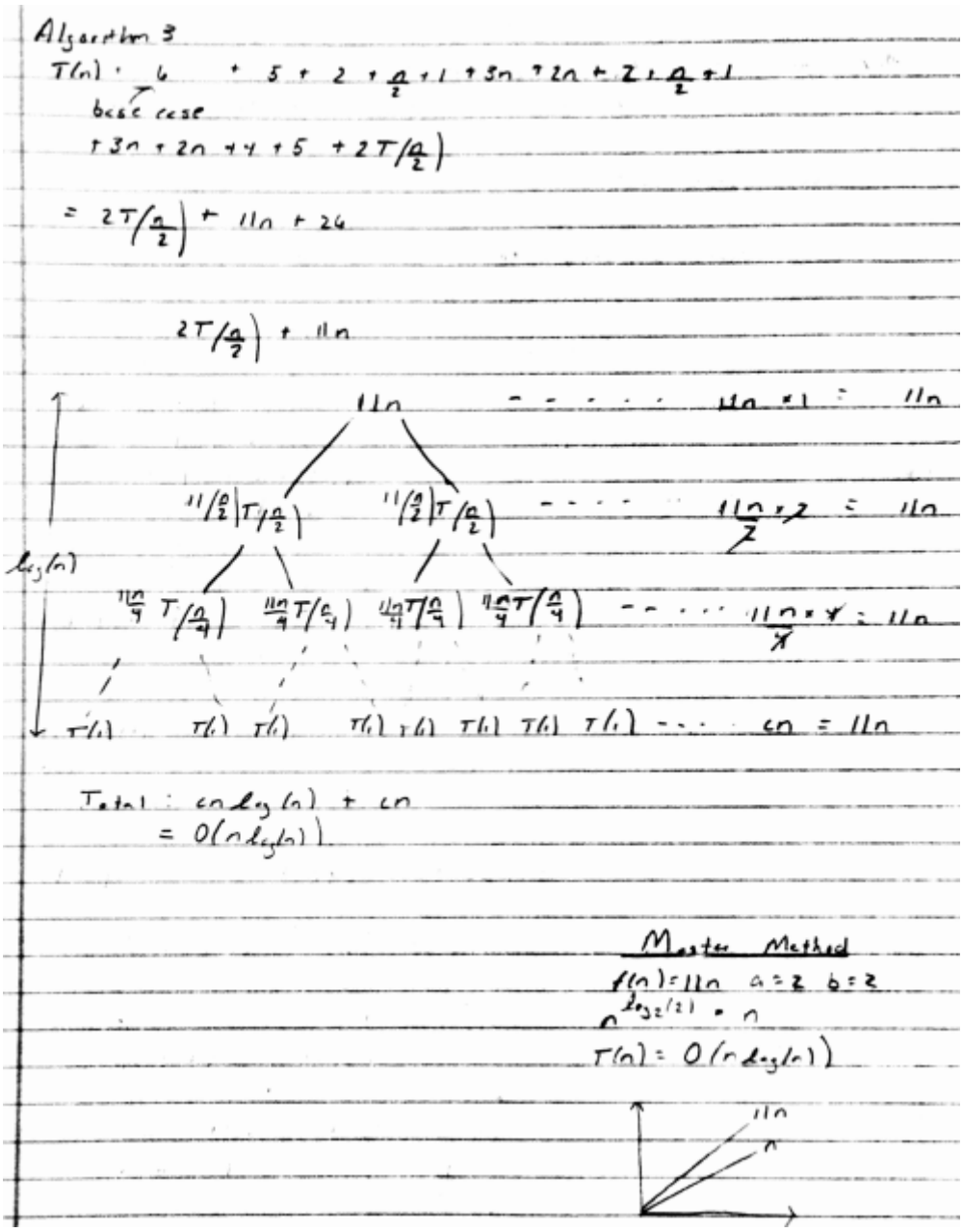
Algorithm-3

```
recursive function MaxSum(X[L..U]: array of integer, L, U: integer)
1       if L > U then
            return 0   /* zero- element vector */
2       if L=U then
            return max(0,X[L])  /* one-element vector */
3       M = (L+U)/2  /* A is X[L..M], B is X[M+1..U] */
        /* Find max crossing to left */
4               sum = 0; maxToLeft =0
5               for I = M downto L do
6                   sum =sum+X[I]
7                   maxToLeft = max(maxToLeft,sum)
        /* Find max crossing to right */
8               sum=0; maxToRight=0
9               for I = M+1 to U
10                  sum=sum+X[I]
11                  maxToRight = max(maxToRight, sum)
12      maxCrossing = maxToLeft + maxToRight

13      maxInA = maxSum(X, L, M)
14      maxInB = maxSum(X,M+1,U)
15      return max(maxCrossing, maxInA, maxInB)
```

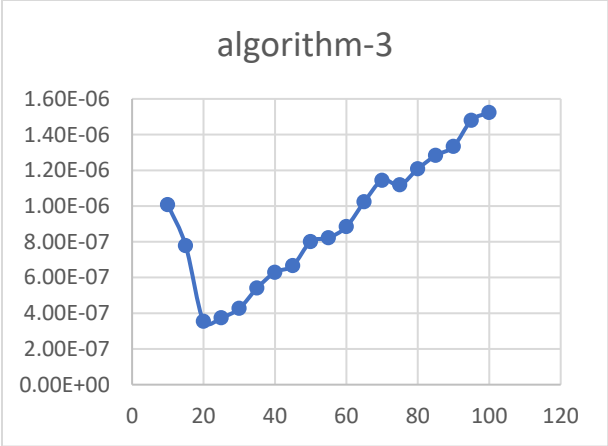
The third algorithm can be found above and calculations/graphs can be found below.

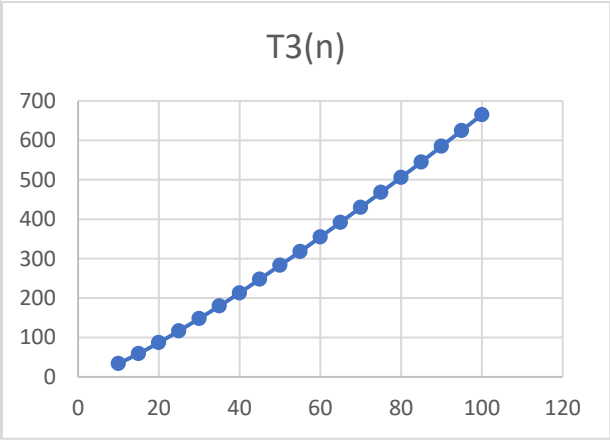
Step	Cost of each execution	Total # of times executed in any single recursive call
1	4	1
2	8	1
Steps executed when input is a base case: 1 & 2		
First recurrence relation: T(n=1 or n=0) = 8 + 4 = 12		
3	5	1
4	2	1
5	1	$\frac{n}{2} + 1$
6	6	$\frac{n}{2}$
7	4	$\frac{n}{2}$
8	2	1
9	1	$\frac{n}{2} + 1$
10	6	$\frac{n}{2}$
11	4	$\frac{n}{2}$
12	4	1
13	1	-
14	1	-
15	5	1
Steps executed when input is NOT a base case: All		
Second recurrence relation: $T(n>1) = 2T(\frac{n}{2}) + 11n + 26$		
Simplified second recurrence relation (ignore the constant term): $T(n>1) = 2T(\frac{n}{2}) + 11n$		



$T_3(n) = 2T(\frac{n}{2}) + 11n + 26 = O(nlog(n))$

The student drew a recursion tree to narrow down what the growth rate of algorithm 3 is. Next, the student used the Master Method to prove that the algorithm will be upper-bounded by some constant.





Here we see that both graphs grow at the same rate. Therefore, the theoretical complexity must have been calculated correctly. Also note that the graphs grow at a slower rate than the graphs in algorithm 2. (i.e. $n^2 > n\log(n)$).

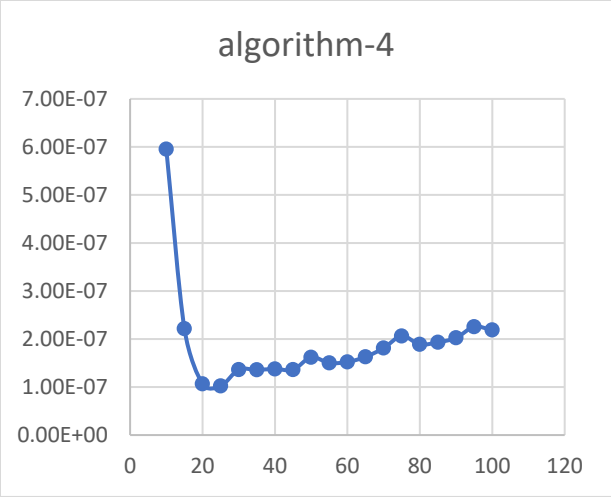
```
Algorithm-4(X : array[P..Q] of integer)
1       maxSoFar = 0
2       maxEndingHere= 0
3       for I = P to Q
4           maxEndingHere = max(0, maxEndingHere + X[I])
5           maxSoFar = max(maxSoFar, maxEndingHere)
6       return maxSoFar
```

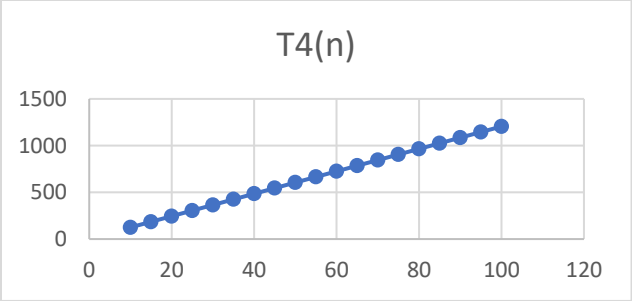
The last algorithm given can be found above and calculations/graphs can be found below.

Step	Cost of each execution	Total # of times executed
1	1	1
2	1	1
3	1	n+1
4	7	n
5	4	n
6	2	1

$T_4(n) = 1 + 1 + n + 1 + 7n + 4n + 2 = 12n + 5 = \Theta(n)$

The student concluded theta-notation because all steps will be executed.





Here we can see that both graphs grow linearly. The only issue the student had was trying to figure out why the theoretical graph grew faster than the one for algorithm 3. The student then realized that because nlog(n) was used to graph the theoretical complexity instead of the actual recurrence found, $\log_2(100)$ will come out to $6.64n < 12n + 5$. The time graphs on the other hand prove that algorithm 3 grows at a faster rate than algorithm 4 (i.e. $n\log(n) > n$).