Duygu Tümer 24842

CS 301 ASSIGNMENT-1

Problem-1

a) $T(n) = 2T(n/2) + n^3 \Rightarrow \Theta(n^3)$

Let's imply Master Theorem for this equation. So, in here a = 2 and b = 2 and f(n) = n^3 so asymptotically positive.

$n^{\log_b^a} = n^{\log_2^2} = n^1 = n$

We need to compare this with f(n).

$f(n) = n^3 = \Omega(n^{\log_b^a + \varepsilon}) = \Omega(n^{1+\varepsilon})$ case 3 may apply so we need to check.

$af(n/b) \leq cf(n)$

$2f(n/2) \leq cn^3$

$n^3/4 \leq cn^3$

$1/4 \leq c < 1$

So $T(n) = \Theta(f(n)) = \Theta(n^3)$

b) $T(n) = 7T(n/2) + n^2 \Rightarrow \Theta(n^{\log_2^7})$

Let's imply Master Theorem for this equation. So, in here a = 7 and b = 2 and f(n) = n^2 which is asymptotically positive.

$n^{\log_b^a} = n^{\log_2^7}$ we need to compare this with f(n).

$f(n) = n^2 = O(n^{\log_b^a - \varepsilon}) = O(n^{\log_2^7 - \varepsilon})$ case 1 applies.

So $T(n) = \Theta(n^{\log_2^7})$

c) $T(n) = 2T(n/4) + \sqrt{n} \Rightarrow \Theta(n^{1/2}logn)$

Let's imply Master Theorem for this equation. So, in here a = 2 and b = 4 and f(n) = $n^{1/2}$ which is asymptotically positive.

$n^{\log_b^a} = n^{\log_4^2} = n^{1/2}$ which is same as $f(n) = n^{1/2}$. That is why case 2 applies.

$T(n) = \Theta(n^{\log_4^2}logn) = \Theta(n^{1/2}logn)$

d) $T(n) = T(n-1) + n \Rightarrow \Theta(n^2)$

Let's find a general formula for this function.

$T(n) = T(n-1) + n$

$T(n-1) = T(n-2) + (n-1)$

$T(n-2) = T(n-3) + (n-2)$

So we can write T(n) as $T(n) = T(n-3) + (n-2) + (n-1) + n$

As we can see that there is a pattern and so we can generalize this function as:

$$T(n) = T(n-i) - \sum_{k=0}^{k=i-1} j + ni$$

$$T(n) = T(n-i) - \frac{(i-1)i}{2} + ni$$

Iteration terminates when $i = n$.

That is why, if we put n instead of i, $T(n) = \Theta(n^2)$.

Problem-2

a)

i) The best asymptotic worst-case running time of the naive recursive algorithm shown in Figure 1 is $\Theta(2^{n+m})$. Think as X and Y do not have any common characters and m = 2 and n = 3. We know that the base case of this algorithm is *if (i == 0 or j == 0): return 0.* Also we know that *else: return max(lcs(X,Y,i,j-1),lcs(X,Y,i-1,j))* evaluates two subproblems.

LCS(2,3)
LCS(1,3) LCS(2,2)
LCS(0,3) **LCS(1,2) LCS(1,2)** LCS(2,1)
…

As you can see, since this is a recursive function, there are the same subproblems which have already been solved. It continues exponentially. If it will continue to evaluate two subproblems till the end, this height of this tree will be $m+n$. That is why, the best asymptotic worst-case running time is $\Theta(2^{n+m})$.

ii) The best asymptotic worst-case running time of the recursive algorithm with memoization, shown in Figure 2 is $\Theta(mn)$. In the previous algorithm, we cannot store the subproblems. That is why, we have to solve the same subproblems and this takes time. Now, with this algorithm, we can hold the subproblems in a $m \times n$ matrix. So, we can remember the solved subproblems.

Let's say that X = "AB" so m = 2 and Y = "CAB" so n = 3. We will look at all these letters in X and Y, one by one. First we will compare 'C' and 'A' since they are the first letters, in X and Y. Since they are not the same we will put 0. Now, compare 'C' with 'B'. Then compare 'A' with 'A'. Since they are the same we will add +1 and we will remember this result. If there are again the same letters we will add +1 again.

	A	B
C	0	0
A	1	1
B	1	2

At the end, we can compute the result like above table. Since we are looking at all m and n words and comparing them one by one, the complexity will be $\Theta(mn)$.

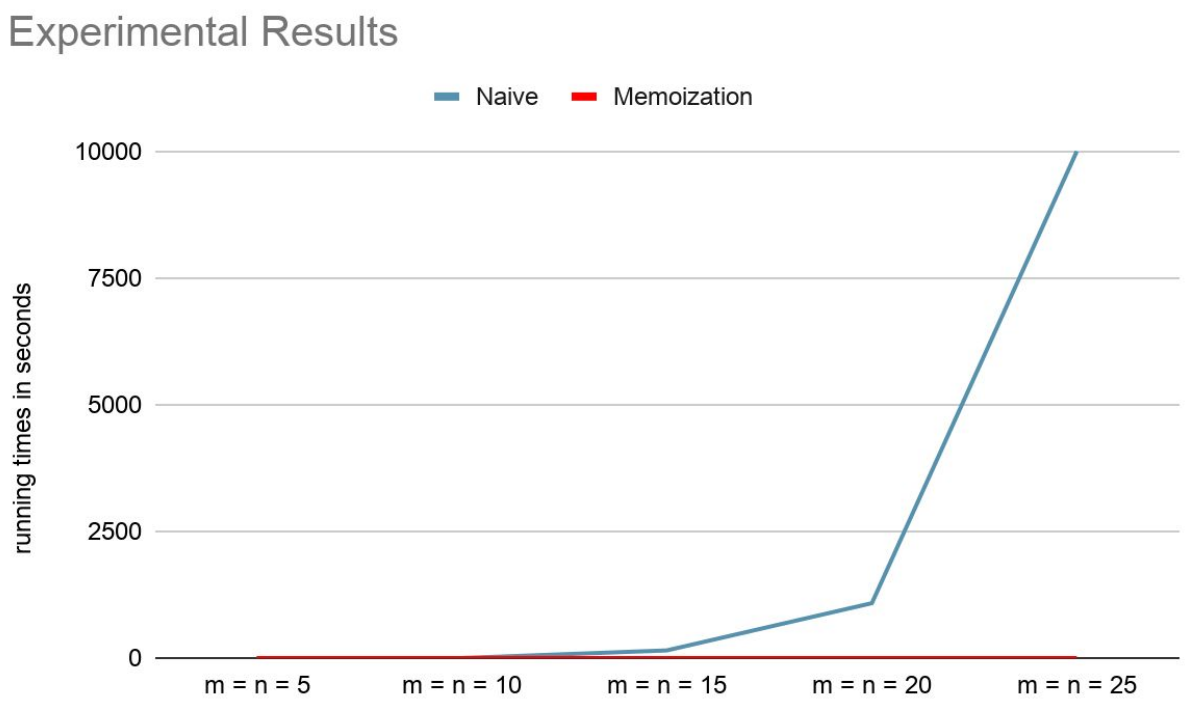
b)

i)

Algorithm	m = n = 5	m = n = 10	m = n = 15	m = n = 20	m = n = 25
Naive	0.0	0.18	150.62	1082.78	>10000
Memoization	0.0	0.0	0.00399	0.000992	0.000998

OS: Windows 10 Home
CPU: Intel Core i7-7500U
RAM: 8,00 GB

ii)



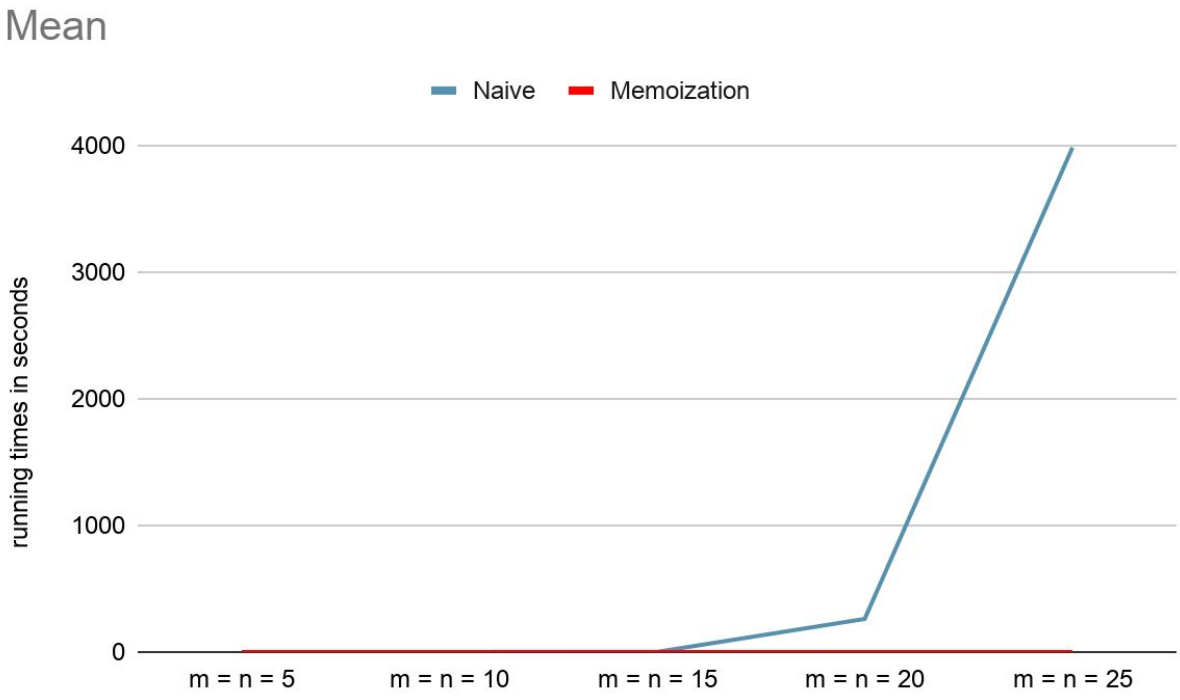
iii) As you can clearly see from the graph, the scalability of the algorithms with respect to these experimental results are really different. The Naive Algorithm is really slow for the large size of inputs because of being an exponential algorithm. The Memoization algorithm is consistent, so it gives nearly the same results for the large and small size of inputs because of being a linear algorithm. So, the experimental results confirm the theoretical results that I found in a).

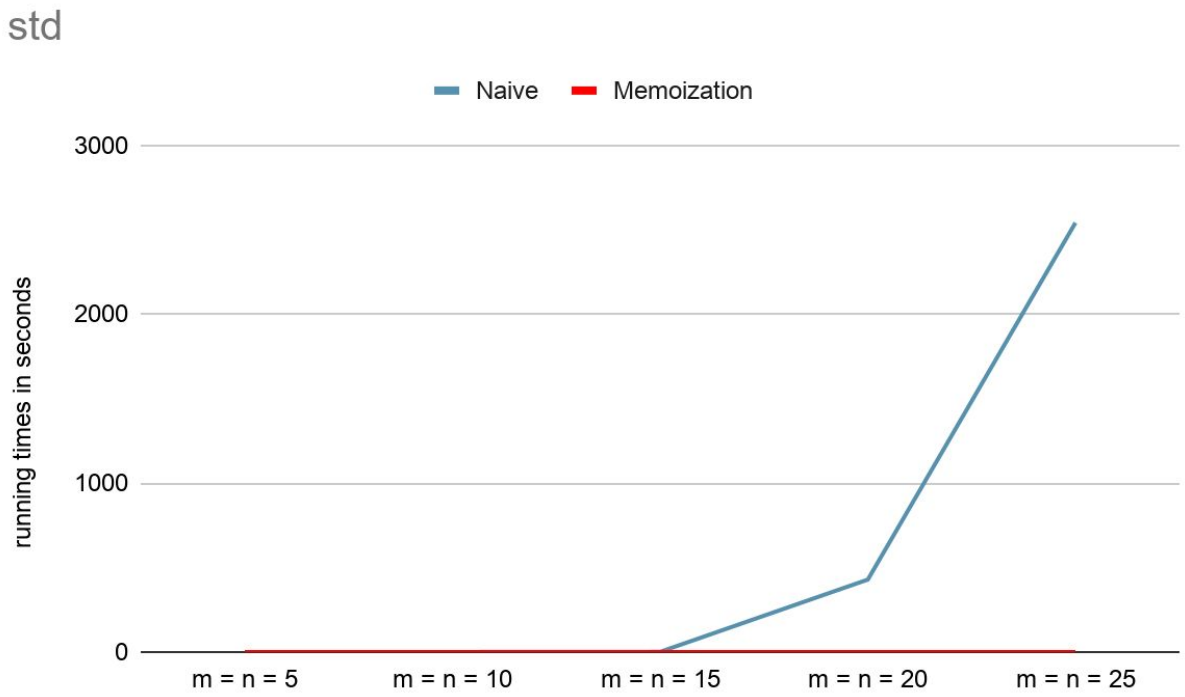
c)

i)

Algorithm	m = n = 5		m = n = 10		m = n = 15		m = n = 20		m = n = 25	
	mean	std	mean	std	mean	std	mean	std	mean	std
Naive	0.0	0.0	0.01583	0.0313	0.91907	0.962736	261.4703	428.802	4936.42	3713.2692
Memoization	0.00016	0.00037	0.00009	0.00030	0.00036	0.000555	0.000629	0.00055	0.00133	0.0006624

ii)





iii) Generally, in the memoization algorithm, the worst case running times observed in b) are really similar to the average running times observed in my experiments; however the average running times are lower than the worst case running times.
In the naive algorithm, the worst case running times observed in b) are higher than the average running times observed in my experiments.
If these situations are compared, then it can be clearly seen that the difference between the lower terms and the higher terms in the naive algorithm has a large difference compared to the memoization algorithm. Because, the naive algorithm is exponential, but the memoization algorithm is linear.