

Assignment 1

Ayman Shahriar UCID: 10180260

October 5, 2020

Question 1

a)

Double input size: $\frac{(2n)^2}{n^2} = \frac{4n^2}{n^2} = 4$

So runtime increases by factor of 4.

Increase input by 1: $\frac{(n+1)^2}{n^2} = \frac{n^2+2n+1}{n^2} = 1 + \frac{2}{n} + \frac{1}{n^2}$

So runtime increases by factor of $1 + \frac{2}{n} + \frac{1}{n^2}$

b)

Double input size: $\frac{(2n)^3}{n^3} = \frac{8n^3}{n^3} = 8$

So runtime increases by factor of 8

Increase input by 1: $\frac{(n+1)^3}{n^3} = \frac{n^3+3n^2+3n+1}{n^3} = 1 + \frac{3}{n} + \frac{3}{n^2} + \frac{1}{n^3}$

So runtime will increase by factor of $1 + \frac{3}{n} + \frac{3}{n^2} + \frac{1}{n^3}$

c)

Double input size: $\frac{100(2n)^2}{100n^2} = \frac{100*4*n^2}{100n^2} = 4$

So runtime increases by factor of 4

Increase input by 1: $\frac{100(n+1)^2}{100n^2} = \frac{n^2+2n+1}{n^2} = 1 + \frac{2}{n} + \frac{1}{n^2}$

So runtime will increase by factor of $1 + \frac{2}{n} + \frac{1}{n^2}$

d)

Double input size: $\frac{2n(\log 2n)}{\log n} = \frac{2(\log 2)(\log n)}{\log n} = 2(\log 2)$

So runtime increases by factor of $2(\log 2)$

Increase input by 1: $\frac{(n+1)(\log(n+1))}{n(\log n)} = \frac{(n+1)}{n} \times \frac{\log(n+1)}{\log n} = \frac{(n+1)(\log_n(n+1))}{n}$
(based on the rule that $\log_a x = \frac{\log_b x}{\log_b a}$)

So runtime will increase by factor of $\frac{(n+1)(\log_n(n+1))}{n}$

e)

Double input size: $\frac{2^{2n}}{2^n} = 2^{2n-n} = 2^n$

So runtime increases by factor of 2^n

Increase input by 1: $= \frac{2^{n+1}}{2^n} = 2^{n+1-n} = 2$

So runtime will increase by factor of 2

Question 2

The functions in order of increasing growth rate:

$f_2(n), f_3(n), f_6(n), f_1(n), f_4(n), f_5(n)$

$f_2(n) = 2n^{\frac{1}{2}}, f_3(n) = n + 10, f_6(n) = n^2(\log n), f_1(n) = n^{2.5}, f_4(n) = 10^n,$
 $f_5(n) = 100^n$

Justification:

- f_2 is a square root function and f_3 is a linear function, so f_2 grows slower than f_3 .
- f_3 is a linear function, and f_1 is a polynomial function of degree greater than 1, so f_3 grows slower than f_1
- Since $n + 10 = O(n)$ and $n + 10 = O(n(\log n))$, then we can use the rule $[g = O(f) \wedge h = O(f) \rightarrow gh = O(f)]$ to conclude that $n + 10 = O(n^2(\log n))$, so $f_3 = O(f_6)$
- Since logarithmic functions grow slower than square root functions, this means that $\log n = O(n^{\frac{1}{2}})$. So $n^2(\log n) = O(n^2 * n^{\frac{1}{2}})$, which means $f_6 = O(f_1)$

- f_1 is a square root function, and f_4 is an exponential function, so f_1 grows slower than f_4
- Both f_4 and f_5 are exponential functions, but the base of f_5 is greater than f_4 , so $f_4 = O(f_5)$

Question 3

a)

We will use the definition of Big-Omega to prove $2n^2 + \sqrt{n} = \Omega(n)$:

Let $n_0 = 1$ and $c = 1$.

Note that $2n^2 \geq n$ for $n \geq 1$

Also note that $\sqrt{n} \geq 0$ for $n \geq 1$

So we can derive that $2n^2 + \sqrt{n} \geq n \geq c \cdot n$ for all $n \geq 1$

So by definition of Big-Omega, $2n^2 + \sqrt{n} = \Omega(n)$

b)

We will use the definition of Big-O to prove that

$5n^3 + 3.5n^2 - 7n + 19 = O(n^3)$:

Let $c = 28, n_0 = 1$. Suppose $n \geq n_0$.

Then $5n^3 + 3.5n^2 - 7n + 19 \leq 5n^3 + 3.5n^2 + 19$ (since $n \geq 1$)

$\leq 5n^3 + 3.5n^3 + 19n^3$ (since $n \geq 1$)

$\leq 27.5n^3$

$\leq 28n^3$

$\leq c \cdot n^3$

So by definition of Big-O, $5n^3 + 3.5n^2 - 7n + 19 = O(n^3)$

c)

We will use limit test to prove that $n^4 = O(2^n)$

$\lim_{x \rightarrow \infty} \frac{n^4}{2^n}$

Since $\lim_{x \rightarrow +\infty} n^4 = \lim_{x \rightarrow +\infty} 2^n = \infty$, we can apply L'Hopital's rule.

Then $\lim_{x \rightarrow \infty} \frac{n^4}{2^n} = \lim_{x \rightarrow \infty} \frac{(n^4)'}{(2^n)'} = \lim_{x \rightarrow \infty} \frac{4n^3}{2^n \ln 2}$

Once more we apply L'Hopital's rule because
 $\lim_{x \rightarrow \infty} 4n^3 = \lim_{x \rightarrow \infty} 2^n \ln 2 = \infty$

Then $\lim_{x \rightarrow \infty} \frac{4n^3}{2^n \ln 2} = \lim_{x \rightarrow \infty} \frac{(4n^3)'}{(2^n \ln 2)'} = \lim_{x \rightarrow \infty} \frac{12n^2}{2^n (\ln 2)^2}$

Once more we apply L'Hopital's rule because
 $\lim_{x \rightarrow \infty} 12n^2 = \lim_{x \rightarrow \infty} 2^n (\ln 2)^2 = \infty$

Then $\lim_{x \rightarrow \infty} \frac{12n^2}{2^n (\ln 2)^2} = \lim_{x \rightarrow \infty} \frac{(12n^2)'}{(2^n (\ln 2)^2)'} = \lim_{x \rightarrow \infty} \frac{24n}{2^n (\ln 2)^3}$

Once more we apply L'Hopital's rule because
 $\lim_{x \rightarrow \infty} 24n = \lim_{x \rightarrow \infty} 2^n (\ln 2)^3 = \infty$

Then $\lim_{x \rightarrow \infty} \frac{24n}{2^n (\ln 2)^3} = \lim_{x \rightarrow \infty} \frac{(24n)'}{(2^n (\ln 2)^3)'} = \lim_{x \rightarrow \infty} \frac{24}{2^n (\ln 2)^4} = \infty$

So using the limit test, we can say that $n^4 = o(2^n)$

Thus, $n^4 = O(2^n)$ because $o(2^n) \subset O(2^n)$

d)

We will use the limit test to prove that $20n^2 + n(\log n) = \Theta(n^2)$

$$\lim_{x \rightarrow \infty} \frac{20n^2 + n(\log n)}{n^2} = \lim_{x \rightarrow \infty} \frac{20n + (\log n)}{n}$$

We can apply L'Hopital's rule because $\lim_{x \rightarrow \infty} 20n + (\log n) = \lim_{x \rightarrow \infty} n = \infty$

Then
 $\lim_{x \rightarrow \infty} \frac{20n + (\log n)}{n} = \lim_{x \rightarrow \infty} \frac{(20n + (\log n))'}{n'} = \lim_{x \rightarrow \infty} 20 + \frac{1}{\ln 10 \cdot n} = 20 + 0 = 20$

So using the limit test, we have proved that $20n^2 + n(\log n) = \Theta(n^2)$

Question 4

a) Both of the sets of n men and women will be implemented as arrays $M = \{1, \dots, n\}$ and $W = \{1, \dots, n\}$. We can order the men and women alphabetically and let the i^{th} man or woman correspond with the i^{th} element in the array. To represent the preference lists of men, there will be an array `ManPref` where each element will be a preference list of a single man. The order these preference lists appear in `ManPref` will be in the same order as their owners appear in array M . We will create a similar list `WomanPref` to represent the preference lists of women. Note that we will use `ManPref[m][i]` to represent the i^{th} woman on man m 's preference list, and `WomanPref[w][i]` to

represent the i^{th} man on woman w 's preference list.

b) We will denote the set of free men as a linked list called FreeMen. When we need to select a free man, we will choose the man at the head of this list. To see whether a woman is free or is engaged to someone, we can create an array Current where Current[w] is the man that w is currently engaged to. If w is not engaged, we can set Current[w] to be a null value (for example: -1). Initially all men and women will be free and unpaired, so the linked list of free men will contain every single man, and all elements of Current will be set to null. Since initializing a linked list of n men involves adding n men to the list, this will cost n steps. And to initialize an array of size n containing all null values, we will have to set all the elements to be null, so this cost is n steps. So the total cost of this line is $n + n = 2n$.

c) To determine if there is a free man who hasn't proposed to all women yet, all we have to do is simply check if the linked list of free men is empty or not. If it's empty, then there are no free man (they are all paired) and if it's not empty, then there is at least one free man left. We do not have to check whether the free man has proposed to all women yet because according to theorem 1.4, page 8 of the textbook, if a man is free then there is a woman that he hasn't proposed to yet. Since checking if a linked list is empty costs 1 step (we just check whether the front of the list is null or not), this line costs 1 step.

d) To choose a free man, all we have to do is select the man at the head of the linked list FreeMen, we do not have to remove that man from the list. Selecting the first element of a linked list costs 1 step, so this operation costs 1 step.

e) In line 4, we need to select the highest ranked woman w that m has not proposed to before. We can implement this by having an array called Next, where Next[m] is the index of the highest ranked woman in m's preference list that he has not proposed to yet. Assuming array indices start at 1, we will initialize Next[m] = 1 for all men. So the woman that m will propose to will be identified as $w = \text{ManPref}[m, \text{Next}[m]]$. And once m proposes to w, we will increment Next[w] by one.

So this implementation consists of accessing and modifying elements of arrays. And since accessing and changing an element in an array take constant time, line 4 will run in around 3 steps (1 to access an element in Next, 1 to access an element in ManPref, and 1 to modify an element in Next).

f) As mentioned in part b), to check if a woman w is free, we need to see if Current[w] is set to null. If it's null, then the woman is free, otherwise Current[w] will be set to the w's current partner. Since accessing an array element takes constant time, this line will run in 1 step.

g) As mentioned in part b), we will use an array called Current to keep track of engaged pairs. If w does not have a partner, the Current[w] is set to null.

Otherwise, $\text{Current}[w]$ is set to w 's current partner. But from line 5, we already know that w does not have a partner, so $\text{Current}[w]$ must be set to null.

So to pair up m and w , we need to set $\text{Current}[w] = m$. We also need to indicate that m is not free anymore by removing m from the linked list FreeMen , where m is the first man in that list. We can do this by changing the pointer that FreeMen uses to point to its first element, to point to its second element. That way m will be removed from the list.

Modifying Current and FreeMen will take 1 step each, so this line runs in 2 steps.

h) At line 7, we know that w is engaged to some man, and we need to determine who that man is. To identify the man, we just need to access $\text{Current}[w]$, which is the man that w is currently engaged with. Since accessing an element takes 1 step, this line runs in 1 step.

i) Since we have not removed m from the linked list FreeMen , we do not have to do anything to indicate that m is still free. So this line has no cost.

j) As mentioned in part g), to pair up m and w , we need to do two things. First, we need to set $\text{Current}[w]$ to be m . This will indicate that the current partner of w is m .

Second, we will need to remove m from the linked list FreeMen , where m is at the head. We can do this by changing the pointer that FreeMen uses to point to its first element, to point to its second element. That way m will be removed from the list.

Since modifying an array and removing the head of the linked list takes 1 step each, this line runs in 2 steps.

k) To show that m' is free again, we have to add m' back into the head of the linked list FreeMen . This will involve 4 steps:

1. Creating a linked list element e
2. Setting $e.\text{val} = m'$
3. Setting $e.\text{Next} = \text{FreeMen}.\text{First}$
4. Setting $\text{FreeMen}.\text{First} = e$

So this line will run in 4 steps.

l) For line 16, we can simply return the array Current , which just takes 1 step. Or, we can use Current to create arrays $\{m_i, w_i\}$ for each pair i (where m_i is the man and w_i is the woman in the pair), and put all those arrays into a larger array of size n , and return that instead. Using this option will involve around n steps, since the number of operations will be a multiple of the size n .

So depending on how we implement it, this operation can cost either 1 step or n steps.

m) Note: this question does not ask us to determine the runtime of line 8. But according to chapter 2.3 of the textbook, if we create an $n \times n$ array called Ranking (where Ranking[w][m] contains the ranking of man m according to w's preference), then we can execute line 8 in constant time.

Total cost of lines 2 to 12 = $1 + 1 + 3 + 1 + 2 + 1 + 0 + 2 + 4 + c = 15 + c$ steps, where c is the constant number of steps needed to execute line 8.

From the textbook, we know that the maximum number of times the loop executes is n^2 . So if we multiply the last result with the number of times the loop is executed, we get $15n^2 + cn^2$ steps

From part b) and l), we know that lines 1 and 16 will cost $2n + 1$ steps. So the total cost of executing this algorithm on input of size n will not exceed $T(n) = 15n^2 + cn^2 + 2n + 1$ steps

n) We will use the limit test to prove that $T(n) = \Theta(n^2)$:

$$\lim_{x \rightarrow +\infty} \frac{15n^2 + cn^2 + 2n + 1}{n^2} = \lim_{x \rightarrow +\infty} 15 + c + \frac{2}{n} + \frac{1}{n^2} = 15 + c + 0 + 0 = 15 + c$$

Since c is a constant greater than 1, then $15 + c \geq 1$, so by definition of the limit test we get $T(n) = \Theta(n^2)$

Question 4

a) Note that lines 1, 5, 6 and 7 all run in constant time because they involve either returning a value or accessing and modifying array elements, all of which take constant time. Let the positive constants c_1, c_5, c_6, c_7 denote the runtimes of lines 1, 5, 6 and 7 respectively.

By itself, line 3 takes $(c_3 \cdot \log i)$ steps, where c_3 is a constant and i is the i_{th} iteration of the outer loop (the outer loop is located at line 2).

And note that in the worst case, the k that is returned in line 2 will always be 0. So in the worst case the inner loop (by itself) will run i times. The inner loop is located at line 3

So if we combine everything, our runtime function will be:

$$T(n) = c_1 + \sum_{i=1}^{n-1} [(c_3 \cdot \log i) + (c_5 \cdot i) + c_6] + c_7$$

$$\begin{aligned}
&= \sum_{i=1}^{n-1} [(c_3 \cdot \log i) + (c_5 \cdot i)] + c_6(n-1) + c_1 + c_7 \\
&= \sum_{i=1}^{n-1} [(c_3 \cdot \log i) + (c_5 \cdot i)] + c_6n + (c_1 + c_7 - c_6)
\end{aligned}$$

b) We will use the definition of Big-O to prove that $T(n) = O(n^2)$

Let $C = (c_1 + c_3 + c_5 + c_6 + c_7)$. Suppose $n \geq 1$. Then:

$$\begin{aligned}
T(n) &= \sum_{i=1}^{n-1} [(c_3 \cdot \log i) + (c_5 \cdot i)] + c_6n + (c_1 + c_7 - c_6) \\
&\leq \sum_{i=1}^{n-1} [(c_3 \cdot \log i) + (c_5 \cdot i)] + c_6n + (c_1 + c_7) \\
&\leq \sum_{i=1}^{n-1} [(c_3 \cdot \log i) + (c_5 \cdot i)] + c_6n^2 + c_1n^2 + c_7n^2 \\
&\leq \sum_{i=1}^{n-1} [(c_3 \cdot i) + (c_5 \cdot i)] + (c_1 + c_6 + c_7)n^2 \quad (\text{since } \log i \leq i \text{ for all } i \geq 1) \\
&\leq [(c_3 + c_5) \cdot \sum_{i=1}^{n-1} i] + (c_1 + c_6 + c_7)n^2 \\
&\leq [(c_3 + c_5) \cdot \frac{n^2-n}{2}] + (c_1 + c_6 + c_7)n^2 \quad (\text{by applying the rule } \sum_{i=1}^n i = \frac{n(n+1)}{2}) \\
&\leq (c_3 + c_5)n^2 + (c_1 + c_6 + c_7)n^2 \quad (\text{since } n \geq 1, \text{ so } n^2 \geq \frac{n^2-n}{2}) \\
&\leq (c_1 + c_3 + c_5 + c_6 + c_7)n^2 \\
&\leq C \cdot n^2, \text{ as required}
\end{aligned}$$

So by the definition of Big-O, we have proved that $T(n) = O(n^2)$

c)

To prove that $T(n) = \Omega(n^2)$, we will use both the definition of Big-Omega and the limit test.

Suppose $n \geq 1$

$$\begin{aligned}
&\text{Then } \sum_{i=1}^{n-1} [(c_3 \cdot \log i) + (c_5 \cdot i)] + c_6n + (c_1 + c_7 - c_6) \\
&\geq \sum_{i=1}^{n-1} [(c_3 \cdot \log i) + (c_5 \cdot i)] - c_6 \quad (\text{since } c_6, c_1, c_7 \text{ are positive constants}) \\
&\geq \sum_{i=1}^{n-1} [c_5 \cdot i] - c_6 \quad (\text{since } (c_3 \cdot \log i) + (c_5 \cdot i) \geq (c_5 \cdot i) \text{ for all } i \geq 1) \\
&\geq c_5 \cdot (\sum_{i=1}^{n-1} i) - c_6 \\
&\geq (c_5 \cdot \frac{n^2-n}{2}) - c_6 \quad (\text{by applying the rule } \sum_{i=1}^n i = \frac{n(n+1)}{2})
\end{aligned}$$

Now let us use the limit test to prove that $(c_5 \cdot \frac{n^2-n}{2}) - c_6 = \Omega(n^2)$:

$$\begin{aligned}\lim_{n \rightarrow \infty} \frac{c_5 \cdot \frac{n^2-n}{2} - c_6}{n^2} &= \lim_{n \rightarrow \infty} \frac{c_5 n^2}{2n^2} - \frac{c_5 n}{2n^2} - \frac{c_6}{n^2} = \lim_{n \rightarrow \infty} \frac{c_5}{2} - \frac{c_5}{2n} - \frac{c_6}{n^2} \\ &= \lim_{n \rightarrow \infty} \frac{c_5}{2}, \text{ where } \frac{c_5}{2} \text{ is a positive constant.}\end{aligned}$$

So by the definition of Θ , $(c_5 \cdot \frac{n^2-n}{2}) - c_6 = \Omega(n^2)$

And since $T(n) \geq (c_5 \cdot \frac{n^2-n}{2}) - c_6$ for all $n \geq 1$, we can use the definition of Big Omega to conclude that $T(n) = \Omega(n^2)$

c)

From parts a) and b), we know that $T(n) = O(n^2)$ and $T(n) = \Omega(n^2)$. Therefore according to the definition of Θ , $T(n) = \Theta(n^2)$.