Assignment 1

Ayman Shahriar UCID: 10180260 Tutorial: T02

October 7, 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.
- Since $n + 10 = O(n)$ and $n + 10 = O(n(\log n))$, then we can use the rule $[f = O(g) \ \wedge \ f = O(h) \quad \rightarrow \quad f = O(g \cdot h)]$
  to conclude than $n + 10 = O(n^2(\log n))$, 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 $n^2(\log n) = O(n^2 \cdot n^{\frac{1}{2}}) = O(n^{2.5})$, which means $f_6 = O(f_1)$
- f_1 is a polynomial 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 that $n^4 = O(2^n)$

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

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

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

Once more we apply LHopital's rule because

$\lim_{x\to\infty} 4n^3 = \lim_{x\to\infty} 2^n \ln 2 = \infty$

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

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

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

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

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

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 $20n^2 + n(\log n) = \Theta(n^2)$

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

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

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

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

Question 4

a) We require the following inputs for this algorithm:

1. A set of men of length n.
2. A set of women of length n
3. The preference lists of each man
4. The preference lists of each woman

Both of the sets of n men and women will be implemented as integer arrays M = $\{1, \ldots, n\}$ and W = $\{1, \ldots, n\}$. 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, we will use a 2D array ManPref where each element will be an integer array that represents the preference list of each man. So ManPref[m] will return the preference array of m where the first index stores the first woman that m prefers, the second index stores the second woman that m prefers, up to the last woman that m prefers. So each array in ManPref will be of length n. And since we represent each women as an integer $\leq n$, that's why the preference lists will be integer arrays.

Then ManPref[m][i] will represent the i^{th} woman on man $m's$ preference list

We will create a similar 2D array WomanPref to represent the preference lists of women. Then WomanPref[w][i] will represent the i^th man on woman $w's$ preference list.

b) This is the pseudocode for line 1:

```
1. Initialize empty linkedList FreeMen
2. For m in M:
3.     FreeMen.add(m)
4. Initialize int array Current of length n
5. For (int i = 0; i < Current.length; i++):
6.     Current[i] = -1
```

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 create an integer 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 of -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 C_1n steps, where C_1 is a constant. 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 will cost C_2n steps, where C_2 is a constant. So the total cost of this line is Cn, where C is a constant.

c) This is the pseudocode for line 2:

```
1. While (FreeMen.First != NULL):
```

To determine if there is a free man who hasn't proposed to all women yet, we have to check if the linked list of FreeMen is empty or not by checking the head of the linked list. If it's null, the list is empty so there are no free men left (they are all paired). If it's not null, then the list is not empty so 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 must be 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 if null or not), this line runs in $\Theta(1)$

d) This is the pseudocode for line 3:

```
1. Node e = FreeMen.First
2. int m = e.val
```

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 line runs in $\Theta(1)$.

e) This is the pseudocode for line 4:

```
1. int w = ManPref[Next[m]]
2. Next[m] = Next[m] + 1
```

We implement line 4 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 at the start of the algorithm.

And since m proposes to w in this line, we will increment Next[m] by one.

So this implementation consists of accessing and modifying arrays. Since accessing or modifying an element in an array take constant time, line 4 will run in $\Theta(1)$

f) This is the pseudocode for line 5:

```
1. if (Current[w] == -1):
```

As mentioned in part b), to check if a woman w is free, we need to see if Current[w] is set to -1. If it's set to -1, 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 runs in $\Theta(1)$.

g) This is the pseudocode for line 6:

```
1. Current[w] = m
2. Node secondElem = FreeMan.First.Next
3. FreeMan.First = secondElem
```

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, Current[w] is set to w's current partner.

And from line 5, we already know that w does not have a partner, so Current[w] must be set to null. So to pair up m and w, we need to set 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 it's 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 $\Theta(1)$.

h) This is the pseudocode for line 7:

```
1. Else:
2.     m_prime = current[w]
```

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 Current[w], which is the man that w is currently engaged with. Since accessing an element takes 1 step, this line runs in $\Theta(1)$.

i) When we reach line 9, we have not yet removed m from the linked list FreeMen, so we do not have to do anything to indicate that m is still free. So this line has no cost.

j) This is the pseudocode for line 11:

```
1. Current[w] = m
2. Node secondElem = FreeMan.First.Next
3. FreeMan.First = secondElem
```

As mentioned in part g), to pair up m and w, we need to do two things. First, we need to set 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 it's 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 $\Theta(1)$.

k) This is the pseudocode for line 12:

```
1. Create empty node e
2. e.val = m_prime
3. e.next = FreeMen.First
4. FreeMen.First = e
```

To show that m' is free again, we have to add m' back into the linked list FreeMen.

As seen from the pseudocode, adding something to a linked list will take a constant number of steps, so this line runs in $\Theta(1)$.

l) This is the pseudocode for line 16:

```
1. return Current
```

For line 16, we can simply return the array Current, which just takes 1 step. So this line runs in $\Theta(1)$

m) Except for line 9 (which has no cost), lines 2-12 all run in $\Theta(1)$. So we can define the runtime of lines 2-12 to be an arbitrary constant C_1.

From page 7, lemma (1.3) of 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 in the worst case, we get

$$C_1 \cdot n^2$$

And note that line 16 runs in constant time, while line 1 runs in $C_2 \cdot n$, where C_2 is a constant. So if we add lines 1 and 16 to our result, we get

$$C_1 n^2 + C_2 n$$

So our runtime of the algorithm is the function:

$$T(n) = C_1 n^2 + C_2 n$$

where C_1 and C_2 are constants.

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

$$\lim_{x \to \infty} \frac{C_1 n^2 + C_2 n}{n^2} = \lim_{x \to \infty} C_1 + \frac{C_2}{n} = C_1 + 0 = C_1$$

Since C_1 is a constant greater than 1, then by definition of the limit test we get $T(n) = \Theta(n^2)$

Question 5

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.

Note that the outer loop of this alrogithm at line 2 will always execute $n-1$ times

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.

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 funtion will be:

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

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

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

We can just combine $c1$ and c_7 into a single positive constant c_a

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

b)

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

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

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

$\leq \sum_{i=1}^{n-1}[(c_3 \cdot \log i) + (c_5 \cdot i)] + c_6 n + c_a$ (since c_6 is a positive constant)

$\leq \sum_{i=1}^{n-1}[(c_3 \cdot \log i) + (c_5 \cdot i)] + c_6 n^2 + c_a n^2$ (since c_6, c_a are positive constants)

$\leq \sum_{i=1}^{n-1}[(c_3 \cdot i) + (c_5 \cdot i)] + (c_6 + c_a)n^2$ (since $\log i \leq i$ for all $i \geq 1$)

$\leq [(c_3 + c_5) \cdot \sum_{i=1}^{n-1} i] + (c_6 + c_a)n^2$

$\leq [(c_3 + c_5) \cdot \frac{n^2-n}{2}] + (c_6 + c_a)n^2$ (by applying the rule $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$)

$\leq (c_3 + c_5)n^2 + (c_6 + c_a)n^2$ (since $n \geq 1$, so $n^2 \geq \frac{n^2-n}{2}$)

$\leq (c_3 + c_5 + c_6 + c_a)n^2$

$\leq C \cdot n^2$, as required

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 $n \geq 1$

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

$\geq \sum_{i=1}^{n-1}[(c_3 \cdot \log i) + (c_5 \cdot i)] - c_6$ (since c_6, c_a are positive constants)

$\geq \sum_{i=1}^{n-1}[c_5 \cdot i] - c_6$ (since $(c_3 \cdot \log i) + (c_5 \cdot i) \geq (c_5 \cdot i)$ for all $i \geq 1$)

$\geq c_5 \cdot (\sum_{i=1}^{n-1} i) - c_6$

$\geq (c_5 \cdot \frac{n^2-n}{2}) - c_6$ (by applying the rule $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$)

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

$\lim_{n\to\infty} \frac{c_5 \cdot \frac{n^2-n}{2} - c_6}{n^2} = \lim_{n\to\infty} \frac{c_5 n^2}{2n^2} - \frac{c_5 n}{2n^2} - \frac{c_6}{n^2} = \lim_{n\to\infty} \frac{c_5}{2} - \frac{c_5}{2n} - \frac{c_6}{n^2}$

$= \lim_{n\to\infty} \frac{c_5}{2}$, where $\frac{c_5}{2}$ is a positive constant.

So according to the limit test, $(c_5 \cdot \frac{n^2-n}{2}) - c_6 = \Theta(n^2)$

And by 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 $T(n) = \Omega(n^2)$

d)

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, $T(n) = \Theta(n^2)$.