COSC 2006 FA 2020 SOLUTION
Assignment 2
Distributed: November 9, 2020
Due: November 15, 2020, end of the day

75 marks

Work the following problems.

1. **(10 marks)** Prove by mathematical induction that

$$\frac{1}{1(2)}+\frac{1}{2(3)}+\frac{1}{3(4)}+\cdots+\frac{1}{n(n+1)}=\sum_{i=1}^{n}\frac{1}{i(i+1)}=\frac{n}{n+1}\quad \forall n\in\mathbb{Z}^{+}.$$

PROOF

Base case:

$$n=1\Rightarrow\sum_{i=1}^{n}\frac{1}{i(i+1)}=\frac{1}{1(1+1)}=\frac{1}{1(2)}=\frac{1}{2}\ (LHS)$$
$$=\frac{n}{n+1}=\frac{1}{1+1}=\frac{1}{2}\ (RHS)$$

Induction hypothesis:

Assume that for some $k > 1$,

$$\frac{1}{1(2)}+\frac{1}{2(3)}+\frac{1}{3(4)}+\cdots+\frac{1}{k(k+1)}=\sum_{i=1}^{k}\frac{1}{i(i+1)}=\frac{k}{k+1}.$$

Induction step:

For $k + 1$,

$$\sum_{i=1}^{k+1}\frac{1}{i(i+1)}=\sum_{i=1}^{k}\frac{1}{i(i+1)}+\frac{1}{(k+1)([k+1]+1)}$$
$$=\frac{k}{k+1}+\frac{1}{(k+1)(k+2)}$$
$$=\frac{k(k+2)}{(k+1)(k+2)}+\frac{1}{(k+1)(k+2)}$$
$$=\frac{k^2+2k+1}{(k+1)(k+2)}$$

$$= \frac{(k+1)^2}{(k+1)(k+2)}$$
$$= \frac{k+1}{k+2} = \frac{[k+1]}{([k+1]+1)}$$

which was to be proven.

QED

2. **(10 marks)** Prove that $n^3 - n$ is divisible by 6 for all $n \in \mathbb{Z}^+$.

PROOF

There are multiple correct approaches to this problem, one of which follows.

Base case: (Trivial) For $n = 1$, $1^3 - 1 = 0$, which is divisible by 6.
(Nontrivial) For $n = 2$, $n^3 - 2 = 8 - 2 = 6$, which is divisible by 6.

Induction hypothesis: Assume that $k^3 - k$ is divisible by 6 for $k > 2$. That is, $k^3 - k = 6m$ for some positive integer m.

Induction step: For $k + 1$ (substitute $k + 1$ for k),
$(k + 1)^3 - (k + 1) = (k + 1)[(k + 1)^2 - 1]$ (simplifying)
$= (k + 1)(k^2 + 2k + 1 - 1) = (k + 1)(k^2 + 2k) = k^3 + k^2 + 2k^2 + 2k = k^3 + 3k^2 + 2k$

To complete the proof, one must relate this expression to the IH. The expression can be rewritten as:
$[k^3 - k] + 3k^2 + 3k$.

From the IH, this becomes: $6m + 3k^2 + 3k$.

Further simplification yields: $6m + 3(k^2 + k) = 6m + 3k(k + 1)$.

$6m$ is divisible by 6, so it needs to be shown that $3k(k + 1)$ is divisible by 6. This is readily proved by noting that k is a positive integer, so that if k is even, $k + 1$ is odd, and if k is odd, $k + 1$ is even. Any even number can be written as $2p$, where p is some integer.

Without loss of generality, assume that k is even. Then $k = 2p$ for some positive integer p. Then the expression becomes:

$6m + 3k(k + 1) = 6m + (3)(2p)(2p + 1) = 6m + 6p(2p + 1) = 6[m + p(2p + 1)]$, which is clearly divisible by 6.

The same can be proven if k is odd. Then $k + 1 = 2p$, and $k = 2p - 1$.

$\therefore$ $6m + 3k(k + 1) = 6m + (3)(2p - 1)(2p) = 6m + 6p(2p - 1) = 6[m + p(2p - 1)]$, which is also divisible by 6.

QED

3. **(10 marks)** Suppose that a particular algorithm requires the following number of assignment operations, as a function of the problem size n.

$$g(n) = 8n^{3.5} \lg n^{n^2} + 120n^3 + 5n^2$$

Determine the order of $g(n)$. In other words, find $f(n)$ such that $g(n) = O(f(n))$, and prove it.

SOLUTION

$$\begin{aligned} g(n) &= 8n^{3.5} \lg n^{n^2} + 120n^3 + 5n^2 \\ &= (n^2)(8n^{3.5})(\lg n) + 120n^3 + 5n^2 \\ &= 8n^{5.5} \lg n + 120n^3 + 5n^2 \end{aligned}$$

By inspection, the highest order term is $8n^{5.5} \lg n$, so $f(n) = n^{5.5} \lg n$.

$$\begin{aligned} &g(n) = 8n^{5.5} \lg n + 120n^3 + 5n^2 \\ &\leq 8n^{5.5} \lg n + 120n^{5.5} \lg n + 5n^{5.5} \lg n \\ &= 133n^{5.5} \lg n \end{aligned}$$

This is true because $n^{5.5} \geq n^3$ for $n \geq 1$ and $n^{5.5} \geq n^2$ for $n \geq 1$. Additionally, $n^{5.5} \lg n \geq n^{5.5}$ for $n \geq 2$. Recall that $\lg 1 = 0$.

$\therefore$ For $c = 133$ and $n_0 = 2$, $g(n) \leq cf(n)$ for $n \geq n_0$.

QED

4. **(20 marks total)** Algorithm A requires $T_A(n) = 32\ n \lg(n^3) + 64n$ operations to perform a specific task of size n, while algorithm B requires $T_B(n) = n^2 + 22n + 100$ operations to perform the same task.

   a. **(5 marks)** For which values of n is algorithm A preferable to algorithm B (i.e. A grows more slowly than B)?

   SOLUTION

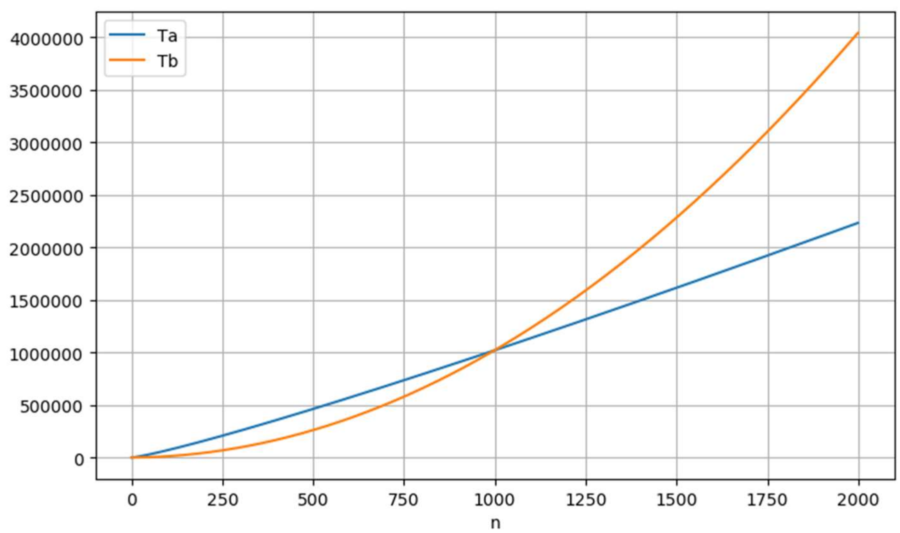
   Observe the following plots, generated by the Python code below:

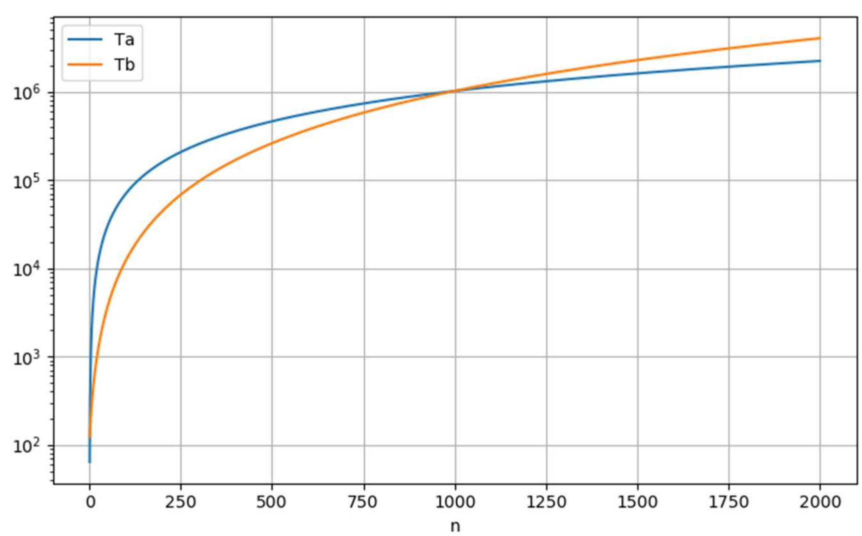
```
import numpy as np
import matplotlib.pyplot as plt
n = np.arange(1,2000)
Ta = (32 * 3) * n * np.log2(n) + 64*n
Tb = n**2 + (22*n) + 100
plt.plot(n, Ta, label = 'Ta')
plt.plot(n, Tb, label = 'Tb')
plt.xlabel('n')
```

```
plt.grid(axis = 'both')
plt.legend()
plt.show(block=False)
```

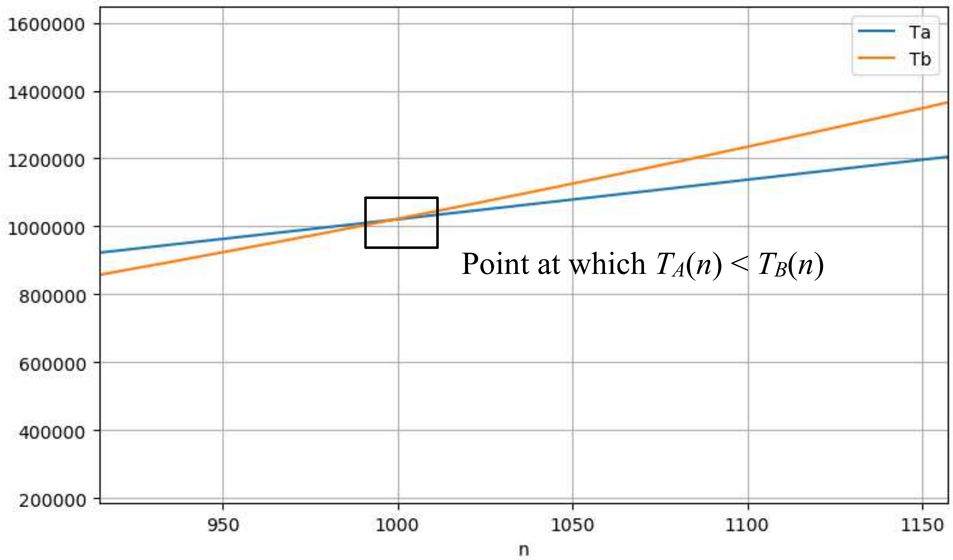
and

```
plt.figure(2)
plt.semilogy(n, Ta, label='Ta')
plt.semilogy(n, Tb, label='Tb')
plt.xlabel('n')
plt.grid(axis = 'both')
plt.legend()
plt.show(block=False)
```





A zoomed-in view of the first plot where the plots of T_A and T_B cross.



$T_A(n) < T_B(n) \Rightarrow 96\ n \lg(n) + 64n < n^2 + 22n + 100$ for $n = 1$ and for $n \geq 999$, as determined by visual analysis of the plots of $T_A(n)$ and $T_B(n)$.

b. **(5 marks)** For which values of n is B preferable to A?

$T_B(n) < T_A(n)$ for $n > 1$ and $n \leq 998$.

c. **(5 marks)** As n grows, which algorithm is preferable?

As n grows, algorithm A is clearly preferable. This can be seen by comparing the highest order terms in $T_A(n)$ and $T_B(n)$, which are, respectively, $n \lg n$ (because $n \lg(n^3) = 3\ n \lg n = O(n \lg n)$) and $n^2 = O(n^2)$. Consequently, Algorithm A is preferable as n gets large. Specifically, for $n \geq 999$, $T_A(n) < T_B(n)$. For $n = 999$, $T_A(n) \approx 1019556 < T_B(n) \approx 1020079$.

d. **(5 marks)** Determine the value of n_0 beyond which the algorithm chosen in Part (c) is preferable. After finding this n_0, prove (analytically – without a computer) that your chosen algorithm is better $\forall\ n \geq n_0$. Remember that n_0 must be integral.

SOLUTION

$96\ n \lg(n) + 64n < n^2 + 22n + 100 \Rightarrow 96\ n \lg(n) + 42n - 100 < n^2$.

Dividing both sides by n (which can be done because $n \geq 1$) yields:

$96 \lg(n) + 42 - 100/n < n$. As shown above, $n_0 = 999$. For $n_0 = 999$, $998.4766 < 999$.

It is known that $\lg(n) < n$ for $n \geq 1$, and therefore, $\lg(n)$ grows more slowly than n. 42 is a constant and is not dependent upon n. The $-100/n$ term is negative and approaches zero as n grows. Consequently, if $96 \lg(n_0) + 42 - 100/n_0 < n_0$, then $96 \lg(n) + 42 - 100/n < n$ for $n > n_0$.

Note: The above explanation also holds for $n = 1$, the first value of n where $T_B(n) < T_A(n)$. If $n = 1$, then $\lg(n) = 0$, and $42 - 100 = -58 < 1$. However, if $n = 2$, then $96 + 42 - 50 = 88 > 2$.

5. **(10 marks)** Calculate $T(n)$ mathematically (the number of times `f` is called as a function of n) for the pseudocode below.

```
for i = 0; i < n; i += 1                // LOOP 0
   for j = 0; j < i; j += 1             // LOOP 1
      for k = n; k > 0; k /= 2          // LOOP 2
         f(i, j)

      for k = 0; k < n; k += 1          // LOOP 3
         f(i, j)

   end for   // j
end for      // i
```

<u>SOLUTION</u>

Loop 1, which is dependent on Loop 0, executes $n(n-1)/2$ times (you should be able to easily prove this).
Loop 2 executes (computes $f(i, j)$) floor($\lg n$) + 1 for each iteration of Loop 1.
Loop 3 executes (computes $f(i, j)$) n times for each iteration of Loop 1.

$\therefore$ $T(n)$ (the number of times $f(i, j)$ is called) = $\frac{n(n-1)}{2}\big((\lfloor \lg n \rfloor + 1) + n\big)$.

6. **(5 marks)** Determine the computational complexity (big-O) of $T(n)$ calculated in Q6. Prove it.

$T(n) \in O(n^3)$

<u>Proof</u>

$$\frac{n(n-1)}{2}\big((\lfloor \lg n \rfloor + 1) + n\big) = \frac{n(n-1)(\lfloor \lg n \rfloor + 1)}{2} + \frac{n^2(n-1)}{2}$$

It is known that floor($\lg n + 1$) $\leq \lg n + 1 \leq n$ for $n \geq 1$.

$$\therefore \frac{n(n-1)(\lfloor \lg n \rfloor + 1)}{2} + \frac{n^2(n-1)}{2} \leq \frac{n^2(n-1)}{2} + \frac{n^2(n-1)}{2} = n^2(n-1) = n^3 - n^2 < n^3$$

$\therefore \frac{n(n-1)}{2}\big((\lfloor \lg n \rfloor + 1) + n\big) \leq cn^3$ for $c = 1$ and $n \geq 1$.

QED

7. **(10 marks)** An array `A` is given as:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	4	3	17	-1	-9	11	22	33	44	-44	-45	1	2	5	7

Perform one iteration of partitioning. In other words, show the array `A` after partitioning once with the MoT pivot selection technique.

SOLUTION

By the MoT pivot selection technique, `first` = 0, `last` = 14, `mid` = `first` + (`last` – `first`)/2 = 0 + (14 – 0)/2 = 7

`A[first]` = 4, `A[mid]` = 33, `A[last]` = 7.

After MoT, the array `A` is as follows, with 7 as the pivot value:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	4	3	17	-1	-9	11	22	7	44	-44	-45	1	2	5	33

After repositioning the pivot, the array `A` is as follows, with `pivotIndex` == 13:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	4	3	17	-1	-9	11	22	5	44	-44	-45	1	2	7	33

The following sequence of operations is performed:

`indexFromLeft` = 1 `indexFromRight` = 12
`A[indexFromLeft]` = 3 < 7 (pivot), `A[indexFromRight]` = 2 < 7 (pivot) ⇒ `indexFromLeft++`

`indexFromLeft` = 2 `indexFromRight` = 12
`A[indexFromLeft]` = 17 > 7 (pivot), `A[indexFromRight]` = 2 < 7 (pivot) ⇒ swap.
`indexFromLeft++` `indexFromRight--`

Result:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	4	3	2	-1	-9	11	22	5	44	-44	-45	1	17	7	33

`indexFromLeft` = 3 `indexFromRight` = 11
`A[indexFromLeft]` = -1 < 7 (pivot), `A[indexFromRight]` = 1 < 7 (pivot) ⇒ `indexFromLeft++`

`indexFromLeft` = 4 `indexFromRight` = 11
`A[indexFromLeft]` = -9 < 7 (pivot), `A[indexFromRight]` = 1 < 7 (pivot) ⇒ `indexFromLeft++`

`indexFromLeft` = 5 `indexFromRight` = 11
`A[indexFromLeft]` = 11 > 7 (pivot), `A[indexFromRight]` = 1 < 7 (pivot) ⇒ swap.
`indexFromLeft++` `indexFromRight--`

Result:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	4	3	2	-1	-9	1	22	5	44	-44	-45	11	17	7	33

`indexFromLeft` = 6 `indexFromRight` = 10
`A[indexFromLeft]` = 22 > 7 (pivot), `A[indexFromRight]` = -45 < 7 (pivot) ⇒ swap.
`indexFromLeft++` `indexFromRight--`

Result:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	4	3	2	-1	-9	1	-45	5	44	-44	22	11	17	7	33

`indexFromLeft` = 7 `indexFromRight` = 9
`A[indexFromLeft]` = 5 < 7 (pivot), `A[indexFromRight]` = -44 < 7 (pivot) ⇒ `indexFromLeft++`

`indexFromLeft` = 8 `indexFromRight` = 9
`A[indexFromLeft]` = 44 > 7 (pivot), `A[indexFromRight]` = -44 < 7 (pivot) ⇒ swap.
`indexFromLeft++` `indexFromRight--`

Result:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	4	3	2	-1	-9	1	-45	5	-44	44	22	11	17	7	33

`indexFromLeft` = 9 > `indexFromRight` = 8 ∴ Get out of the `while` loop (`done ← true`).

Interchange `A[pivotIndex]` = 7 and `A[indexFromLeft]` = 44.

Result:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	4	3	2	-1	-9	1	-45	5	-44	7	22	11	17	44	33

After partitioning, all array elements with values ≤ the pivot value (in this case, 7) are placed to the left of the pivot, and all array elements with values ≥ the pivot value are placed to the right of the pivot.