CS2040S AY19/20 S1 - Tutorial Week 3 Solutions

1. Asymptotic Analysis

1.1. **Time complexity:** $\mathcal{O}(n)$

Since j is initialized to $n+1$, the condition $j < i$ is always false and hence, the inner loop never runs.

Space complexity: $\mathcal{O}(1)$

1.2. **Time complexity:** $\mathcal{O}(1)$

The inner loop is not dependent on the input parameter x, and hence will always run for a constant (exactly 14) iterations.

Space complexity: $\mathcal{O}(1)$

1.3. **Time complexity:** $\mathcal{O}(1)$

Remember that in asymptotic analysis, we only care about the behaviour of the function for large values of n, that is, as n tends to Infinity.

Space complexity: $\mathcal{O}(1)$

1.4. **Time complexity:** $\mathcal{O}(n \log n)$

The recurrence relation is $T(n) = 3T(n/3) + cn$.

Draw out the recursion tree and you'll notice the sum of work done at each level is cn.

Starting with the top, the 0th level represents $T(n)$, the 1st level represents $T(n/3)$, and the k-th level represents $T(n/3^k)$. The base case of the recursion is when $n = 1$. Solving for k in the expression $n/3^k = 1$, we get $k = \log_3 n$, which means there are $\log_3 n$ levels[1] in the recurrence tree.

The final time complexity is $cn \log_3 n$

Space complexity: $\mathcal{O}(\log n)$

For a recursive function, every individual recursive call takes up some space. Here, each individual call takes up a constant amount of space (There are only 4 or so variables being initialized). So, the space complexity depends on the maximum height of the recursion stack, which corresponds to the maximum depth of your recursion tree.

As discussed earlier, there are $\mathcal{O}(\log n)$ levels in the recursion tree which is equal to the maximum depth.

1.5. **Time complexity:** $\mathcal{O}(n)$

The loop runs for $n/2$ iterations, and remember that we drop constant factors in Big-O notation. $\mathcal{O}(n/2) = \mathcal{O}(n)$.

Space complexity: $\mathcal{O}(1)$

Since each iteration of the loop pops and pushes into the stack, the size of the stack only changes by at most 1 element throughout the loop.

1.6. **Time complexity** $\mathcal{O}(n)$

The recurrence relation is $T(n) = 2T(n/2) + c$.

Draw out the recursion tree (with the top level being the 0th level) to determine that:

- In the k-th level, $2^k c$ work is being done.
- For the bottom level, $k = \log_2 n$, which corresponds to $2^{\log_2 n} c = nc$ work done.

Hence, the recurrence relation is given by the following sum:

$$
\begin{aligned}
T(n) &= c + 2c + 2^2 c + \cdots + \frac{nc}{2} + nc \\
&= nc + \frac{nc}{2} + \frac{nc}{2^2} + \cdots + c && \text{(Reverse the sum)} \\
&= nc\left(1 + \frac{1}{2} + \frac{1}{2^2} \cdots\right) && \text{(Take out the factor of } nc\text{)} \\
&\leq nc \cdot C && \text{(A geometric series with a factor} < 1 \text{ is bounded by a constant)} \\
&= \mathcal{O}(n)
\end{aligned}
$$

Space complexity: $\mathcal{O}(n)$

There are two things to consider:

- The maximum recursion depth is $\log_2 n$, which means a space complexity of $\mathcal{O}(\log n)$ from the recursion stack.
- A value is enqueued into the Queue at every recursive call. From the analysis within the Time complexity, there are $\mathcal{O}(n)$ recursive calls, which results in $\mathcal{O}(n)$ values enqueued into the Queue.

The larger of the two is $\mathcal{O}(n)$.

[1] There are actually $(\log_3 n + 1)$ levels, but I don't really care toooo much about it since it won't affect the final complexity.

2. Moar Asymptotic Analysis

2.a. F: $\mathcal{O}\left(n^4\right)$

Within the expression, there are three individual terms.

$$\left(\frac{3n^2}{32}\right)\left(\frac{n^2}{2}\right) = \mathcal{O}\left(n^4\right) \qquad \frac{n^2}{n-10} = \mathcal{O}\left(n\right) \qquad n^2 \log n = \mathcal{O}\left(n^2 \log n\right)$$

Among the terms, the most dominating (the 'largest') term is the answer.

2.b. C: $\mathcal{O}\left(n\right)$

Expand the recurrence relation (using a recursion tree or otherwise) and you have the following expression:

$$\begin{aligned} T(n) &= T(n/4) + 3n \\ &= 3n + \frac{3n}{4} + \frac{3n}{4^2} + \cdots \\ &= 3n\left(1 + \frac{1}{4} + \frac{1}{4^2} + \cdots\right) \\ &\leq 3n \cdot C \\ &= \mathcal{O}\left(n\right) \end{aligned}$$

2.c. D: $\mathcal{O}\left(n \log n\right)$

This recurrence relation splits the input n into two separate recursive calls of size $\frac{n}{4}$ and $\frac{3n}{4}$. This is similar to the analysis of Paranoid Quicksort which splits the input n into two calls of size $\frac{n}{10}$ and $\frac{9n}{10}$.

Drawing the recursion tree, each level does $3n$ work and there are $\log_{4/3} n$ levels, for a final complexity of $\mathcal{O}\left(3n \log_{4/3} n\right) = \mathcal{O}\left(n \log n\right)$.

2.d. E: $\mathcal{O}\left(n^2\right)$

Expand the recurrence relation and you have the following expression:

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + 2n^2 \\ &= 2n^2 + \frac{2n^2}{4} + \frac{2n^2}{4^2} + \cdots \\ &= 2n^2\left(1 + \frac{1}{4} + \frac{1}{4^2} + \cdots\right) \\ &\leq 2n^2 \cdot C \\ &= \mathcal{O}\left(n^2\right) \end{aligned}$$

2.e. H: $\mathcal{O}\left(5^{\log_2 n}\right) = \mathcal{O}\left(n^{\log_2 5}\right) \approx \mathcal{O}\left(n^{2.3219}\right)$

Draw the recurrence tree as per usual, but be a bit more careful.

- The k-th level corresponds to $T(\frac{n}{2^k})$.
- The k-th level does $5^k c$ work.
- Solving for k in $\frac{n}{2^k} = 1$, you have $k = \log_2 n$ as the bottom level.

Then form the sum as per usual.

$$\begin{aligned} T(n) &= c + 5c + 5^2 c + \cdots + 5^{\log_2 n} c \\ &= 5^{\log_2 n} c + \frac{5^{\log_2 n} c}{5} + \frac{5^{\log_2 n} c}{5^2} + \cdots + c \\ &= 5^{\log_2 n} c\left(1 + \frac{1}{5} + \frac{1}{5^2} + \cdots\right) \\ &\leq 5^{\log_2 n} c \cdot C \\ &= \mathcal{O}\left(5^{\log_2 n}\right) \end{aligned}$$

2.f. E: $\mathcal{O}(2^n)$

To make my life simpler, I'm just going to assume that the recurrence relation is $T(n) = 2T(n-1) + n$ instead of $T(n) = 2T(n-1) + 5n$. The change from $5n$ to n won't affect the answer; I assure you.

Drawing the recurrence tree,

- The k-th level corresponds to $T(n-k)$.
- The k-th level does $(n-k)2^k$ work.
- There are n levels in this recurrence tree.

Expand out the term and you have this mess:

$$\begin{aligned} T(n) &= 2T(n-1) + n \\ &= n + 2(n-1) + 2^2(n-2) + \cdots + 2^{n-1}(1) \end{aligned}$$

Another way to look at the mess is as such:

- I have n copies of 2^0.
- I have $(n-1)$ copies of 2^1.
- I have $(n-2)$ copies of 2^2.

  $\vdots$
- I have 1 copy of 2^{n-1}

Using this idea, I shall split it into n separate sums:

$$\begin{aligned} S_1 &= 1 + 2 + 2^2 + \cdots + 2^{n-3} + 2^{n-2} + 2^{n-1} && = 2^n - 1 \\ S_2 &= 1 + 2 + 2^2 + \cdots + 2^{n-3} + 2^{n-2} && = 2^{n-1} - 1 \\ S_3 &= 1 + 2 + 2^2 + \cdots + 2^{n-3} && = 2^{n-2} - 1 \\ &\vdots \\ S_n &= 1 && = 2^1 - 1 \end{aligned}$$

You can verify that $(S_1 + S_2 + \cdots + S_n)$ is equal to the mess we started with. Finally, to finish it off:

$$\begin{aligned} T(n) &= n + 2(n-1) + 2^2(n-2) + \cdots + 2^{n-1}(1) \\ &= (2^n - 1) + \left(2^{n-1} - 1\right) + \cdots + \left(2^1 - 1\right) \\ &= \left(2^n + 2^{n-1} + \cdots + 2^1\right) - n \\ &= 2^{n+1} - 2 - n \\ &= \mathcal{O}(2^n) \end{aligned}$$

There are other ways to solve this.

If you happen to know[2] that $\sum_{i=0}^{n-1} ir^i = \frac{r - nr^n + (n-1)^{n+1}}{(1-r)^2}$, substituting $r = 2$ will give us:

$$\sum_{i=0}^{n-1} i2^i = 2 - n2^n + (n-1)^{n+1}$$

From there,

$$\begin{aligned} T(n) &= \sum_{i=0}^{n-1} 2^i(n-i) \\ &= n\sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} i2^i \\ &= n\left(2^n - 1\right) - \left(2 - n2^n + (n-1)2^{n+1}\right) \\ &= n2^n - n - 2 + n2^n - 2n^2n + 2^{n+1} \\ &= 2^{n+1} - n - 2 \\ &= \mathcal{O}(2^n) \end{aligned}$$

[2] I definitely didn't know this off the top of my head.