CSC236 Tutorial #5

Sample Solutions

1. Let $T(n)$ denote the worst-case running time of the algorithm below on inputs of size n.

```
# A is a non-empty list of integers, i is a natural number.
    def recSS(A, i):
1.      if i < len(A) - 1:
2.          small = i
3.          for j in range(i + 1, len(A)):
4.              if A[j] < A[small]:
5.                  small = j
6.          temp = A[i]
7.          A[i] = A[small]
8.          A[small] = temp
9.          recSS(A, i + 1)
```

Note that the above algorithm has an implicit base case $i =$ `len(A) - 1`, for which it does nothing.

(a) Write a recurrence relation satisfied by T. Make sure to define n precisely (as a function of the algorithm's parameters) and justify that your recurrence is correct (by referring to the algorithm to describe how you obtained each term in your answer).

(b) Give an asymptotic upper-bound for the worst-case running time of the algorithm.

Solutions:

(a) For any natural number n, let $T(n)$ denote the maximum number of steps executed by a call to $recSS(A, i)$, where $n = len(A) - i$.

If $n = 1$, then it does nothing except evaluating the if-condition, which take constant time, represented by a constant value a.

Otherwise, lines 2 to 9 execute. The for-loop in line 3 executes $n - 1$ times. Therefore, lines 3–5 take $b * (n - 1)$, where b is a constant value. There is a recursive call in line 9 on a list of size $(n - 1)$. Therefore, line 9 takes $T(n - 1)$.

Lines $6 - 8$, and all other instructions in lines 2–9 take constant time, represented by a constant value c. Putting all together, we get the following definition for $T(n)$:

$$T(n) = \begin{cases} a, & \text{n} = 1 \\ T(n-1) + b(n-1) + c, & n \geq 2 \end{cases}$$

(b) Assume $n \geq 2$. Then

$$\begin{aligned}
T(n) &= T(n-1) + b(n-1) + c \\
&= [T(n-2) + b(n-2) + c] + b(n-1) + c \\
&= T(n-2) + b((n-1) + (n-2)) + 2c \\
&= [T(n-3) + b(n-3) + c] + b((n-1) + (n-2)) + 2c \\
&= T(n-4) + b((n-1) + (n-2) + (n-3)) + 3c
\end{aligned}$$

It seems that after i applications of the recursive definition we have

$$T(n) = T(n-i) + b((n-1) + (n-2) + (n-3) + ... + (n-i)) + i * c$$

Therefore, after n applications of the recursive definition we have

$$\begin{aligned}
T(n) &= T(n-n) + b((n-1) + (n-2) + (n-3) + ... + (n-n)) + n * c \\
&= a + b\frac{(n-1)n}{2} + cn = \frac{b}{2}n^2 - \frac{b}{2}n + cn + a
\end{aligned}$$

Note that in a test/assignment you are expected to prove the correctness of the closed-form expression you obtained for T by induction. However, here we skip this step as the proof for this part is tedious.

Finally, we can conclude that $T(n) \in \mathcal{O}(n^2)$.

2. When an annual interest rate of i is compounded m times per year, the interest rate paid per period is $\frac{i}{m}$. For instance, if $3\% = 0.03$ annual interest is compounded quarterly, then the interest rate paid per quarter is $\frac{0.03}{4} = 0.0075$.
For each integer $k \geq 0$, let $Q(k)$ denote the amount on deposit at the end of the kth period, assuming no additional deposits or withdrawals.

   (a) Let d denote the amount of an initial deposit into a bank account earning interest at a rate of i which is compounded m times per year.
   Find a recurrence relation relating $Q(k)$ to $Q(k-1)$.

   (b) Find a closed-form formula for $Q(k)$.
   Note that as discussed in class, you are required to do repeated substitutions, guess a pattern, use the pattern to find a closed-form expression for Q, and finally proved the correctness of the closed-form expression using induction.

Solutions:

(a) The initial deposit into the account is d. Thus, $Q(0)$ is equal to d.

The interest earned during the kth period equals the amount on deposit at the end of the $(k-1)$st period times the interest rate for the period. In other words, interest earned during kth period is equal to $Q(k-1)\frac{i}{m}$.
The amount on deposit at the end of the kth period, $Q(k)$, equals the amount at the end of the $(k-1)$st period, $Q(k1)$, plus the interest earned during the kth period. That is,

$$Q(k) = Q(k-1) + (\frac{i}{m})Q(k-1) = (1 + \frac{i}{m})Q(k-1).$$

Putting all together, we get the following definition for $Q(k)$:

$$Q(k) = \begin{cases} d, & \text{k} = 0 \\ (1 + \frac{i}{m})Q(k-1), & k \geq 1 \end{cases}$$

(b) Assume $k \geq 1$. Let $b = (1 + \frac{i}{m})$. Then

$$\begin{aligned} Q(k) &= b\,Q(k-1) \\ &= b[b\,Q(k-2)] \\ &= b^2 Q(k-2) \\ &= b^2[bQ(k-3)] \\ &= b^3 Q(k-3) \end{aligned}$$

It seems that after i applications of the recursive definition we have

$$Q(k) = (b)^i\,Q(k-i).$$

Therefore, after k applications of the recursive definition we have

$$\begin{aligned} Q(k) &= b^k\,Q(0) \\ &= d * b^k \end{aligned}$$

Let $P(k)$ denote the assertion that $Q(k) = d * b^k$.
Using induction it can be proved that $k \in \mathbb{N}$, $P(k)$ holds.
The proof is straightforward but tedious.

3. Give an asymptotic upper bound for each of the following functions.

(a)

$$T_1(n) = \begin{cases} a, & n = 1 \text{ or } n = 2 \\ 9T_1(\frac{n}{3}) + \frac{n^3}{\log n}, & n \geq 3 \end{cases}$$

Solutions: Since $n^3 \log n$ is not $\Theta(n^k)$ for any $k \geq 0$, we cannot use the Master theorem. However, observe that $\frac{n^3}{\log n} \in \mathcal{O}(n^3)$, so we can use the Master Theorem for

$$T'(n) = 9T'(\frac{n}{3}) + n^3$$

to get that $T'(n) \in \mathcal{O}(n^3)$. Since $T_1(n) \leq T'(n)$, n^3 is an upper bound for $T_1(n)$ as well.

(b)

$$T_2(n) = \begin{cases} a, & n = 1 \\ 2T_2(n/2) + 4n, & n \geq 2 \end{cases}$$

Solutions: We will use the Master Theorem.

Here, $c = 2, d = 2$, and $k = 1$. Since $\log_2 2 = 1 = k$, by the Master Theorem, $T_2(n) \in \mathcal{O}(n \log n)$.

(c)

$$T_3(n) = \begin{cases} a, & n = 1 \\ 2T_3(n/7) + \log n + \sqrt{n}, & n \geq 2 \end{cases}$$

Solutions: We will use the Master Theorem.

Here $c = 2$, $d = 7$, and $f(n) = \log n + \sqrt{n}$. So, $k = \frac{1}{2}$.
Since $\log_7 2 \approx 0.3562 < 0.5$, by the Master Theorem, $T_3(n) \in \mathcal{O}(\sqrt{n})$