# EECS 3101: Design and Analysis of Algorithms
# Tutorial 6 - Solutions

1. Solve the following recurrence:

   $T(1) = 1$ and for all $n \geq 2$, $T(n) = nT(n-1) + n$.

   **Solution:** It is possible to do this with repeated substitution. This was done in the tutorial. Here is an alternative solution. Define $S(n) = T(n)/n!$. Then since

$$\begin{aligned}
T(n) &= nT(n-1) + n, \text{ dividing both sides by } n! \text{ we have} \\
\frac{T(n)}{n!} &= \frac{T(n-1)}{(n-1)!} + \frac{1}{(n-1)!} \\
S(n) &= S(n-1) + \frac{1}{(n-1)!} \\
&= S(n-2) + \frac{1}{(n-2)!} + \frac{1}{(n-1)!} \\
&\quad \ldots \\
&= S(1) + \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{(n-2)!} + \frac{1}{(n-1)!} \\
&= 1 + \frac{1}{1!} + \frac{1}{2!} + \ldots + \frac{1}{(n-2)!} + \frac{1}{(n-1)!} \\
&= \sum_{i=0}^{n-1} \frac{1}{i!}, \text{ so substituting for } T(n) \text{we have} \\
T(n) &= n! \sum_{i=0}^{n-1} \frac{1}{i!}
\end{aligned}$$

   Similarly if $n$ is odd, we get $T(n) = T(1) + \lg 3^2 5^2 \ldots (n-2)^2 n^2$.

   We see that $2 \leq \sum_{i=0}^{n-1} \frac{1}{i!} \leq e$ for $n \geq 1$, so it follows that $T(n) = \Theta(n!)$.

2. Solve the following recurrence to get a big-O bound, where $T(1) = 1$ and for $n > 1$,

$$T(n) = 2\sqrt{n}T(\sqrt{n}) + 4n$$

   You may ignore floors and ceilings.

   **Solution:** Let us first substitute $S(n) = T(n)/n$. Then, we have $S(n) = 2S(\sqrt{n}) + 4$. We can now substitute $n = 2^m$, so that $m = \lg n$. This yields $S(2^m) = 2S(2^{m/2}) + 4$. Now define $R(m) = S(2^m)$. Then we have the following recurrence for $R(m)$:

$$R(m) = 2R(m/2) + 4$$

   Case 1 of the Master Theorem holds, since $n^{\log_b a} = n^1$ and $f(n) = 4 = 4n^0 = 4n^{1-1}$ so $\epsilon = 1$ works. Using case 1 of the Master Theorem, we get $R(m) = O(m)$. Therefore $S(2^m) = O(m)$, or $S(n) = O(\lg n)$. Therefore $T(n) = O(n \lg n)$.

3. $T(n) = 4T(\sqrt[3]{n}) + n$.

   **Solution:** We can't use the master method directly because the argument to $T$ on the RHS is not of the form $n/b$. Changing the variable, we rename $n = 2^m$. So $T(2^m) = 4T(2^{m/3}) + 2^m$. Define $S(m) = T(2^m)$. Then $S(m) = 4S(m/3) + 2^m$. We can use the Master Theorem to solve this recurrence, **provided** $S(1)$ is defined. It is important to note that we need $S(1)$ as a boundary condition if we ignore floors, because for any finite $k$, $m/3^k > 0$. We can compute $S(1)$ using $S(1) = T(2) = 4T(1) + 2 = 6$.

   Here, $a = 4, b = 3$, so $n^{\log_b a} = n^{\log_3 4}$. We will show that $f(m) = 2^m = \Omega(n^{\log_3 4+\epsilon})$. Choose $\epsilon$ such that $n^{\log_3 4+\epsilon} = n^2$. We can prove that $2^m = \omega(n^2)$ (you can prove this using limits as in the last problem). We check the regularity condition next. This requires us to prove that $af(n/b) < cf(n)$ for some $c < 1$ and sufficiently large $n$. Since $4 \cdot 2^{n/3} = 2^{n/3+2} = \frac{1}{2}2^{n/3+3} < \frac{1}{2}2^n$ when $n/3 + 3 < n$ or $n \geq 5$.

   Thus case 3 of the master method applies. Hence, $S(m) = \Theta(2^m)$. Changing back from $S(m)$ to $T(n)$, we obtain $T(n) = \Theta(n)$.

4. Consider the following code for Build-Heap, which operates on a heap stored in an array $A[1 \ldots n]$:

```
Build-Heap($A$)
1  $heap - size[A] \leftarrow length[A]$
2  for $i = \lfloor length[A]/2 \rfloor$ down to 1
3  do Max-Heapify($A, i$)
```

   Show how this procedure can be implemented as a recursive divide-and-conquer procedure Build-Heap($A, i$), where $A[i]$ is the root of the sub-heap to be built. To build the entire heap, we would call Build-Heap($A, 1$).

   **Solution:** Recall that Max-Heapify($A, i$) requires that the two subtrees of node $i$ must already be heaps (the iterative version given above starts building the heap bottom-up for this reason). The recursive version makes the two subtrees of node $i$ heaps by calling itself recursively, and then invokes Max-Heapify on node $i$.

```
Build-Heap-R($A, i$)
1  if $i \leq \lfloor length[A]/2 \rfloor$
2     then Build-Heap-R($A, 2i$)
3          Build-Heap-R($A, 2i + 1$)
4          Max-Heapify($A, i$)
```

5. Give a recurrence that describes the worst-case running time of the previous procedure. Solve the recurrence using the Master theorem.

   **Solution:** Since Max-Heapify takes time $O(\lg n)$ and there are two recursive calls to Build-Heap-R, the running time can be expressed using the following equations: $T(1) = O(1)$, and for $n > 1$,

$$T(n) = 2T(n/2) + O(\lg n)$$

Solution of the recurrence: First, we must show that case 1 of the Master Theorem applies. Since $a = 2, b = 2$, so $n^{\log_b a} = n^{\log_2 2} = n$, and $f(n) = \lg n$, we can use the Master Theorem using $\epsilon = 0.5$ (for example). The proof that $\lg n = O(n^{0.5})$ is omitted. One way to do this is by showing that $\lg n = o(n^{0.5})$ using the limit definition of $o()$.

Then, $T(n) = \Theta(n)$ by case 1 of the Master theorem.