# CS 3510 Honors Algorithms
# Solutions : Homework I

1. (a) For a function $f(n)$ which satisfies $\sum_{i=1}^{n} f(i) = \Theta(f(n))$, consider $f(n) = 2^n$. Clearly, $\sum_{i=1}^{n} f(i) = \sum_{i=1}^{n} 2^i = 2^{n+1} - 1$. We can say that

$$0 \leq 1 \cdot 2^n \leq 2^{n+1} - 1 \leq 2 \cdot 2^n$$

Hence $\sum_{i=1}^{n} 2^i = \Theta(2^n)$.

For a function which does not satisfy the above, consider $f(n) = n$. We know that $\sum_{i=1}^{n} f(i) = n(n+1)/2 = \Theta(n^2) \neq \Theta(n)$. Hence $\sum_{i=1}^{n} i \neq \Theta(n)$

(b) We know that $\log(n!) \leq \log(n^n) = n \log n$. Also, $\log(n!) \geq \log(n \cdot (n-1) \cdot (n-2) \cdots (\frac{n}{2}+1)) \geq \log(\frac{n}{2}^{\frac{n}{2}}) = \frac{n}{2} \log(\frac{n}{2}) = \frac{n}{2} \log n - \frac{n}{2}$. By picking for example $c = 1/3$, we can ensure that $\log(n!) \geq cn \log n$, for large enough $n$. Now that we have proved inequalities for both sides, we can say that $\log(n!) = \Theta(n \log n)$.

2. (a) $T(n) = T(\sqrt{n})+1$. Substitute $n = 2^m$. We get $T(2^m) = T(2^{\frac{m}{2}})+1$. Writing $T(2^m) = S(m)$, we get $S(m) = S(\frac{m}{2})+1$. It is easy to see that $S(m) = \Theta(\log m)$ works. That is, $T(n) = \Theta(\log \log n)$.

On substitution, one can verify that

$$\begin{aligned} \Theta(\log \log \sqrt{n}) + 1 &= \Theta(\log(1/2 \log n)) + 1 \\ &= \Theta(\log \log n) - \log 2 + 1 \\ &= \Theta(\log \log n) \end{aligned}$$

(b)

$$\begin{aligned} T(n) &= 1 + 2T(n-1) \\ &= 1 + 2(1 + 2T(n-2)) \\ &= 1 + 2 + 4 + 8T(n-3) \\ &\vdots \\ &= 1 + 2 + 4 + \cdots + 2^{n-1}T(1) \\ &= 2^n - 1 = \Theta(2^n) \end{aligned}$$

(c) $T(n) = 2T(n/3) + 1$. Substitute $n = 3^m$. We get $T(3^m) = 2T(3^{m-1})+1$. Using $S(m) = T(3^m)$, we get $S(m) = 2S(m-1)+1$, which is same as the part (b) of this question. The solution is $T(n) = \Theta(2^m) = \Theta(2^{\log_3 n} = \Theta(n^{1/\log 3})$.

(d)

$$\begin{aligned} T(n) &= 49T(n/25) + (\sqrt{n})^3 \log n \\ &= (\sqrt{n})^3 \log n + 49(\sqrt{\frac{n}{25}}^3 \log \frac{n}{25} + 49T(\frac{n}{25^2})) \\ &= (\sqrt{n})^3 \log n + \frac{49}{125}\sqrt{n}^3 \log n - \frac{49}{125}\sqrt{n}^3 \log 25 + 49^2 T(\frac{n}{25^2}) \end{aligned}$$

Ignoring constants, we get $T(n) = \Theta((\sqrt{n})^3 \log n)$.

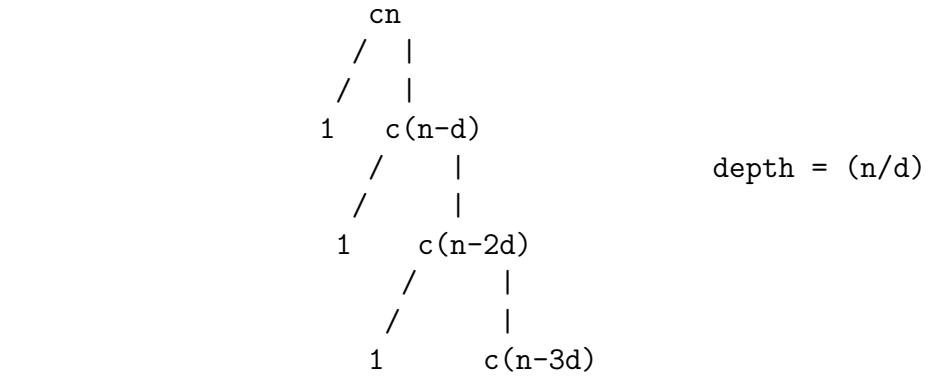
(e) $T(n) = 9T(n/3) + n^2 \log n$. Substitute $n = 3^m$. We get $T(3^m) = 9T(3^{m-1}) + 3^{2m} m \log 3$. Using $S(m) = T(3^m)$, we get

$$\begin{aligned} S(m) &= 9S(m-1) + m9^m \log 3 \\ &= m9^m \log 3 + 9((m-1)9^{m-1} \log 3 + 9S(m-2)) \\ &= 9^m \log 3(m + (m-1) + (m-2) + \cdots + 1) \\ &= \Theta(9^m m^2) \\ T(n) &= \Theta(n^2 \log_3^2 n) \\ &= \Theta(n^2 \log^2 n) \end{aligned}$$

(f)

$$\begin{aligned} T(n) &= 8T(n/2) + n^3 \\ &= n^3 + 8((\frac{n}{2})^3 + 8T(n/4)) \\ &= n^3 + n^3 + 8^2 T(n/4) \\ &= \Theta(n^3 \log n) \end{aligned}$$

3. (a) The recursion tree is as given below.




So the $T(n)$ is given by,

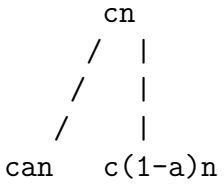
$$T(n) = 1 \cdot \lceil n/d \rceil + (cn + c(n-d) + c(n-2d) + \cdots c(n - \lfloor n/d \rfloor d))$$

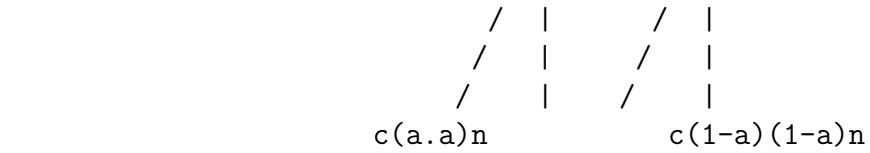
This can be written as $T(n) = O(n^2)$. Now let us prove this using induction. Say $T(n) \leq kn^2$, for a constant $k > 0$.

$$\begin{aligned} T(n) &\leq T(n-d) + T(d) + cn \\ &\leq k(n-d)^2 + cn + 1 \\ &\leq k(n^2 - 2dn + d^2) + cn + 1 \\ &\leq kn^2 + n(c - 2kd) + (kd^2 + 1) \\ &\leq kn^2 \end{aligned}$$

when $k$ is chosen such that $k < c/2d$. Thus we have proved by induction that $T(n) = O(n^2)$.

(b) We can draw the recursion tree as follows. Note that `a` denotes $\alpha$, and the depth of the tree is $\log_\alpha n$ where without any loss of generality, we can assume that $\alpha \geq 1/2$. Note that the depth is the longest path from the root to any leaf, which will be determined by $\alpha$.






Summing all levels up, we get the sum as $T(n) = cn \cdot \log_\alpha n$. This is of the form $T(n) = O(n \log n)$. Now let us try to prove the same by induction. Let $T(n) = kn \log n$.

$$\begin{aligned} T(n) &\leq T(\alpha n) + T((1-\alpha)n) + cn \\ &\leq k\alpha n \log(\alpha n) + k(1-\alpha)n \log((1-\alpha)n) + cn \\ &= k\alpha n \log n + k\alpha n \log \alpha + k(1-\alpha)n \log n + k(1-\alpha)n \log(1-\alpha) + cn \\ &= kn \log n + k(\alpha \log \alpha + (1-\alpha) log(1-\alpha))n + cn \\ &\leq kn \log n \end{aligned}$$

This inequality holds provided we choose a $k$ such that $k(\alpha \log \alpha + (1-\alpha) log(1-\alpha)) + c \geq 0$ which we can choose.

4. (a) We can start off weighing two coins against each other. If they balance, then none of them are the fake coin. Else, we know for sure that the lighter one is the fake one. After we weigh two coins and found that they balance, then we need to check only the remaining coins. So in the best case (lower bound), we might find the fake coin in the first weighing. In the worst case (upper bound), the fake coin might elude us till the last weighing, which will be $\lfloor n/2 \rfloor$th weighing. In the worst case we make $O(n)$ weighings.

   (b) Here we can reduce the number of weighings by a great deal because we can weigh any number of coins. First we shall divide the coins into three equal heaps (if we have any leftover, leave one or coins more in the third heap) and balance heap one against heap two. If the two heaps balance, the third heap has the fake coin. Else the lighter set contains the lighter fake coin. Now on, we can take the lighter set and recursively do the same operation. It is easy to see that (if we have $n$ is power of 3 to start with) we will find the fake coin at the last weighing which is $\log_3 n$th weighing.

This is the upper bound. The lower bound is also same if $n$ is a power of 3.

(c) Here we can first divide the whole set of coins into heaps of $k$ coins each. Take one such heap on each side to start, and then weigh them against each other. First we will find out which of the $k$ piles contain the fake coin. This requires (by an argument similar to part (a)) a maximum of $\lfloor n/2k \rfloor$ weighings. If we are lucky, we might find the fake heap in the first weighing itself. After we have isolated the heap containing the fake coin, we can go on and split it into halves as in part (b). Here we would require $\log_3 k$ weighings. Here the upper bound is $\lfloor n/2k \rfloor + \log_3 k$ weighings and lower bound is $1 + \log_3 k$ weighings.

5. (a) We could try the following strategy. Assume that we run the algorithm for finding the $\gamma$ approximate median. We can go through the set and divide the remaining set into numbers less than the returned approximate median, say $S_1$, and the numbers greater than the returned median $S_2$ (Think and convince yourself that this division can be done in linear time). Because of the $\gamma$ approximation guarantee, we are sure that $|S_1|, |S_2| > \gamma n - 1$. Depending on which of $|S_1|$ and $|S_2|$ is bigger, we can discard the other set, since we know for sure that the exact median would be in the bigger set. The remaining set would have size atmost $(1 - \gamma)$ of the original set. We can recursively do the same procedure for a linear time algorithm.

For the running time analysis, assume the approximate median algorithm, and the splitting of the sets take $cn$ time. Now we have $T(n) \leq T((1-\gamma)n) + cn$

$$\begin{aligned} T(n) &\leq T((1-\gamma)n) + cn \\ &\leq cn + c(1-\gamma)n + T((1-\gamma)^2 n) \\ &\leq cn + c(1-\gamma)n + c(1-\gamma)^2 n + \cdots \\ &= cn\frac{1}{\gamma} \\ &= O(n) \end{aligned}$$

Hence this algorithm has linear running time on the size of the input.

(b) We could complete the algorithm in part (a) if we can construct the algorithm which can find the $\gamma$ approximate median in linear time. We can divide the input into sets of 7 elements each, there would be $n/7$ such sets. If we take the median of each of these sets, and then the median of these medians, then we know for certain that this median is greater than half ($n/14$) of the other medians. Since all these are respective medians, they must be in turn greater than 3 other elements in their set. Thus, the median of medians is greater than at least $\frac{n}{14} \cdot 4$ ($n/14$ medians and the other three elements of those sets) elements of the original set. Similarly it is smaller than $\frac{n}{14} \cdot 4$ elements of the original set. Hence this would be a $\gamma$ approximate median, where $\gamma = 4/14 = 2/7$.

Now we have to argue that this can be done in linear time. For a 7 element set, we can do the sorting in constant time and thus find the median. So we need $O(n)$ time to find out all the medians. To find out the median of this group, we know that the set is 1/7 the size of the original set and we can recursively do the same procedure. By an analysis similar to that in part (a), we can see that this will be a geometric series summation with ratio 1/7 and thus we get linear time algorithm for finding a $\gamma$ approximate median.