University of Houston

COSC 3320: Algorithms and Data Structures
Spring 2016

Solutions for Homework 3

1. Consider the following recurrence relation, and assume n to be a power of four.

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 2T(n/4) + \sqrt{n} & \text{if } n = 4^d, d > 0. \end{cases}$$

   (a) Apply the Master Theorem to have an asymptotic bound for $T(n)$.
   (b) Determine the exact value of $T(n)$ using the unfolding technique.
   (c) Prove by induction the correctness of the above solution.
   (d) Verify that the above solution is coherent with the asymptotic bound obtained in (a).

Solution:

(a) We are in the second case of the Master Theorem, with $a = 2$, $b = 4$, and $k = 0$. The asymptotic estimate provided by the Master Theorem is therefore $T(n) = \Theta(\sqrt{n}\log n)$.

(b) Let $n = 4^d$ with $d > 0$ sufficiently big. By applying a few times the definition of $T(n)$ we get

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{4}\right) + \sqrt{n} \\ &= 2\left(2T\left(\frac{n}{4^2}\right) + \sqrt{\frac{n}{4}}\right) + \sqrt{n} = 2^2T\left(\frac{n}{4^2}\right) + 2\sqrt{n} \\ &= 2^2\left(2T\left(\frac{n}{4^3}\right) + \sqrt{\frac{n}{4^2}}\right) + 2\sqrt{n} = 2^3T\left(\frac{n}{4^3}\right) + 3\sqrt{n} \\ &\vdots \\ &= 2^iT\left(\frac{n}{4^i}\right) + i\sqrt{n}. \end{aligned}$$

Choosing i such that $n/4^i = 1$ and substituting in the above formula we obtain $T(n) = (1 + \log_4 n)\sqrt{n}$.

(c) By induction on d. Base case: $d = 0$ (i.e., $n = 1$) is easy to check. Then suppose the formula true for $d - 1$, with $d > 0$. Let $n = 4^d$. We have

$$\begin{aligned} T(n) &= 2T(n/4) + \sqrt{n} \\ &= 2(1 + \log_4(n/4))\sqrt{n/4} + \sqrt{n} \quad \text{(by inductive hypothesis)} \\ &= 2(\log_4 n)\sqrt{n/4} + \sqrt{n} \\ &= (1 + \log_4 n)\sqrt{n}. \end{aligned}$$

(d) $(1 + \log_4 n)\sqrt{n} = \sqrt{n} + \sqrt{n}\log_4 n = \Theta(\sqrt{n}\log n)$.

2. Consider the following recurrence relation, and assume n to be a power of four.

$$T(n) = \begin{cases} \sqrt{n} & \text{if } n = 1, \\ 4T(n/4) + n/2 & \text{if } n = 4^d, d > 0. \end{cases}$$

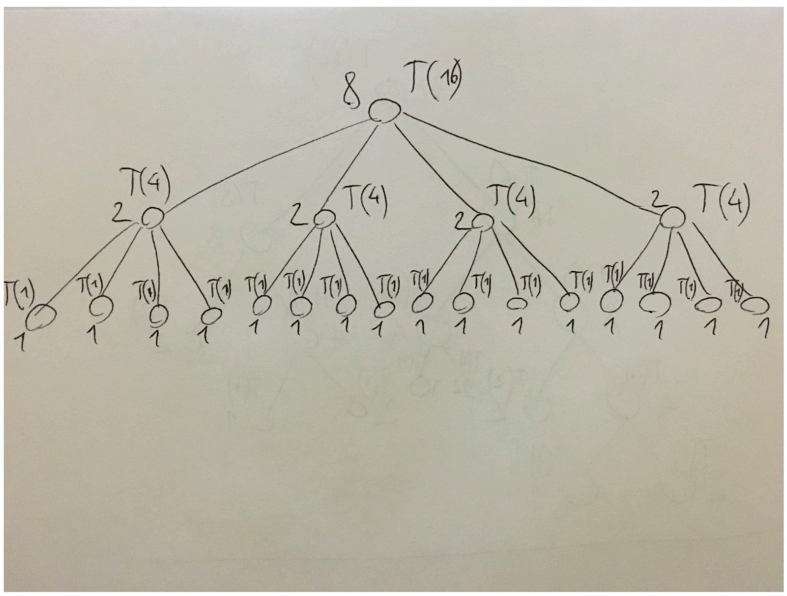
(a) Draw the recursion tree for $n = 16$.

(b) Determine the number of levels of the recursion tree, and the total cost associated to each level.

(c) From (b), determine the exact value of $T(n)$.

Solution:

(a)



(b) The number of the levels of the tree is $\log_4 n + 1$, because the leaves are associated to sizes that start from n and get divided by four at each level, till reaching the value 1 for which we have the base case of the recurrence relation. The total cost associated to each non-leaf level is $n/2$, while the total cost of the level of the leaves is n.

(c) From (b) we have $T(n) = n/2 \log_4 n + n$.

3. Consider the following recurrence relation, and assume n to be a power of two.

$$T(n) = \begin{cases} n & \text{if } n \in \{1, 2\}, \\ T(n/2) + T(n/4) + n & \text{if } n = 2^d, d > 1. \end{cases}$$

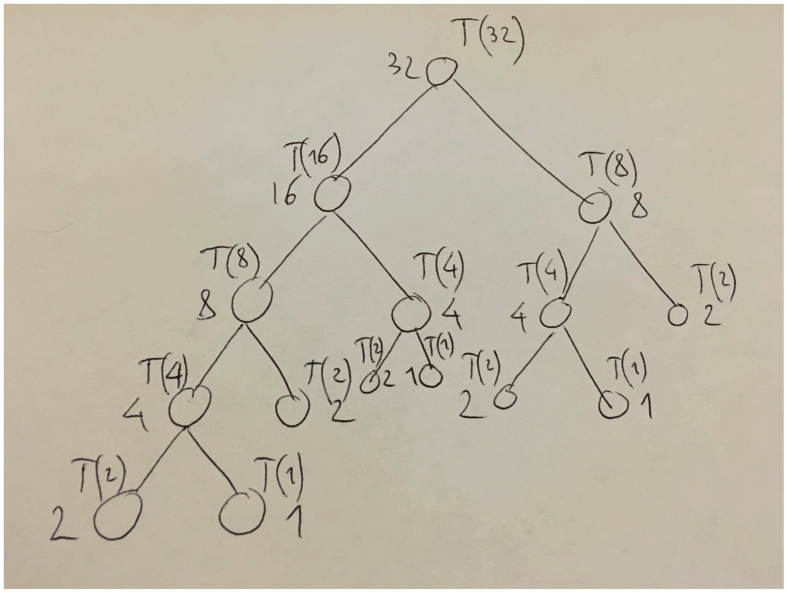
(a) Draw the recursion tree for $n = 32$.

(b) Determine the number ℓ of levels of the recursion tree, and an upper bound to the total cost associated to level i, with $0 \leq i < \ell$.

(c) From (b), determine an upper bound to $T(n)$, and argue that this bound is asymptotically tight.

Solution:

(a)



(b) The number of the levels of the tree is $\log_2 n$, because the nodes of the deepest branch are associated to sizes that start from n and halve at each level, till reaching the value 2 for which we have the base case of the recurrence relation. Consider a node v with associated cost $m \geq 4$. The costs associated to the children of v are $m/2$ and $m/4$, hence $3m/4$ overall. Hence, if level i has total cost x_i, the total cost of level $i+1$ is at most $(3/4)x_i$. Since $x_0 = n$, we have that x_i is at most $n(3/4)^i$.

(c) From (b) we have

$$T(n) = \sum_{i=0}^{\log_2 n - 1} x_i \leq n \sum_{i=0}^{\log_2 n - 1} (3/4)^i = n \frac{1 - (3/4)^{\log_2 n}}{1 - 3/4} = \Theta(n).$$

By definition, $T(n) \geq n$, hence the above bound is asymptotically tight.

4. Let $S = S[0], S[1], \ldots, S[n-1]$ be a sequence of n elements on which a total order relation is defined. An *inversion* in S is a pair of elements $S[i], S[j]$ such that $S[i] > S[j]$ and $i < j$. Give a recursive algorithm that determines the number of inversions in S in time $O(n \log n)$. (Hint: adapt the Merge-Sort strategy.)

Solution:

The idea is to divide S in two subsequences S_1 and S_2. Let m_1 (resp., m_2) be the number of inversions involving elements in S_1 (resp., S_2). Then the inversions in S will be m_1 plus m_2 plus the "half-crossing" inversions, where the latter is the number of inversions that involve one element in S_1 and one in S_2. m_1 and m_2 can be determined recursively, and the half-crossing inversions can be quickly determined once S_1 and S_2 are ordered, during the merge of them, in the following way: when merging, increment the counter by the number of the remaining elements in the left subsequence S_1 if the pointed element in the left subsequence S_1 is greater than the pointed element in the right subsequence S_2. The pseudocode follows. For simplicity, this solution assumes that the n input elements are all distinct.

```
Sort-and-Inversions(S)
input: Sequence S of n distinct elements
output: Sequence S ordered and m = number of inversions in S
if (n=1) then return {S,0}
S_1 <- S[0,...,\lceil n/2 \rceil - 1]  \\ \lceil and \rceil denote the ceiling function
S_2 <- S[\lceil n/2 \rceil,..., n-1]
{S_1,m_1} <- Sort-and-Inversions(S_1)
{S_2,m_2} <- Sort-and-Inversions(S_2)
m <- m_1 + m_2
r <- 0; t <- 0; k <- 0
while ((r < S_1.size) AND (t < S_2.size)) do
  if (S_1[r] < S_2[t]) then
    S[k++] <- S_1[r++]
  else
    S[k++] <- S_2[t++]
    m <- m + (S_1.size - r)
while (r < S_1.size) do
  S[k++] <- S_1[r++]
while (t < S_2.size) do
  S[k++] <- S_2[t++]
return {S,m}
```

The complexity of the algorithm is $T(n) = 2T(n/2) + cn$, for some constant c, and this is $O(n \log n)$.