Assignment 5

Unmarked Questions

Remember to comment out this section when submitting your assignment.

Here are a few simple warm-up problems. Make sure you are able to do them before proceeding to the marked questions.

Q1. Master Method

Use a recursion tree to determine a good asymptotic upper bound on the recurrence

$$T(n) = 3T\left(n/2\right) + n.$$

Use the substitution method to verify your answer.

Solution. The recurrence tree is shown in Figure 1

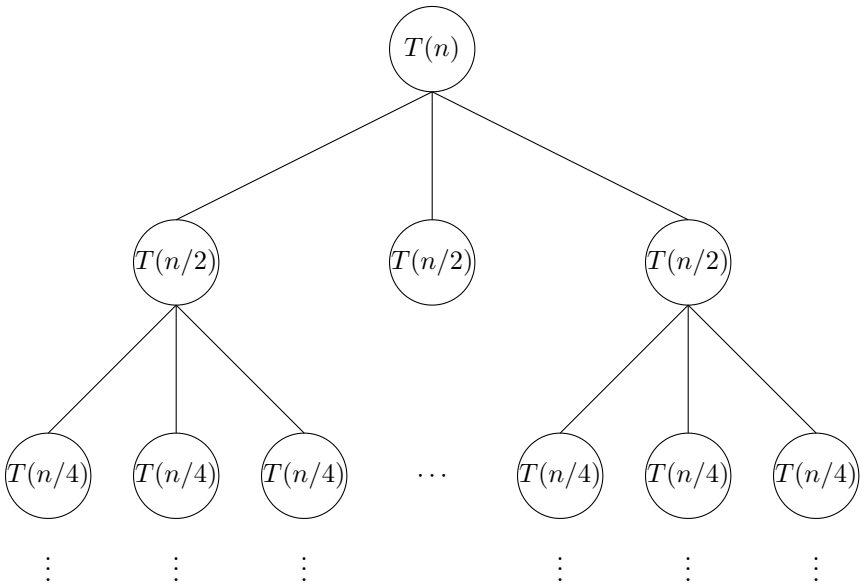

Figure 1: Recurrence tree of the recurrence relation $T(n) = 3T\left(n/2\right) + n$.

Thus, for the substitution method, we will guess that $T(n) = \Theta\left(n^{\log_2 3}\right)$. Typically we will need to show that $T(n) = \Omega\left(n^{\log_2 3}\right)$ and $T(n) = O\left(n^{\log_2 3}\right)$, but we will only show the former (the latter is the same).

We will assume that $T(n) \leq c$ is for $n \leq 2$. Let the predicate $P(n)$ be $T(n) \geq cn^{\log_2 3}$. We will prove $P(n)$ is true for all $n \in \mathbb{N}$. The base case is taken care of by the assumption.

In the inductive case for $k > 2$, suppose we have $P(0) \wedge \cdots \wedge P(k-1)$ and we want to show $P(k)$ is true. From the definitions, we have that $T(k) = 3T(k/2) + k$. By the inductive hypothesis, we

have $T(k/2) \geq c\,(k/2)^{\log_2 3}$. It follows that

$$\begin{aligned} T(k) &= 3T\,(k/2) + k \\ &\geq 3c\left(\frac{k}{2}\right)^{\log_2 3} + k \\ &\geq ck^{\log_2 3} + k. \end{aligned}$$

Thus, by the principle of mathematical induction, $P(n)$ is true for all n.

Q2. Recursive Minimum

Algorithm 1 recmin(A)

```
1: if |A| = 1 then
2:     return A[0]
3: end if
4: m ← |A|/2                                   ▷ integer division
5: m1 ← recmin(A[0..m − 1])
6: m2 ← recmin(A[m..|A| − 1])
7: return min(m1, m2)
```

a. State the pre- and post-condition of Algorithm 1.

Solution. The pre-condition is that A is a non-empty list. The post-condition is that Algorithm 1 will return the minimum element of A.

b. Prove that Algorithm 1 is correct.

Solution. We apply structural induction on the sub-arrays of A.

In the base case, a sub-array consisting of a single element has that element as the minimum. It is returned on line 2.

In the inductive step, when the sub-array B has more than one element, $m \geq 1$ and $B[0..m-1]$ and $B[m..|B| - 1]$ are both arrays of length at least one. By the inductive hypothesis, the recursive calls on line 5 and 6 satisfy the post-condition: m_1 is the minimum element of $B[0..m - 1]$ and m_2 is the minimum element of $B[m..|A| - 1]$. It follows that the minimum of B is $\min(m_1, m_2)$.

It follows by structural induction that the recursive algorithm applied to A returns the minimum element of A.

c. Analyse the worst-case running time of Algorithm 1.

Solution. The running time on an input of size n can be written as $T(n) = 2T(n/2) + c$ for some constant c. We make two recursive calls in lines 5 and 6 on inputs of half the size and constant work outside of the recursive calls in lines 1 through 4. By the Master Method, we have that $T(n) = \Theta(n)$.

Marked Questions

Q1. [10 Points] Recurrence Relations *(maximum 3 pages)*

The following are some questions related to finding the closed form for various recurrence relations.

a. Use the recursion tree method to guess an asymptotically tight bound for the recurrence

$$T(n) = T(\alpha n) + T((1-\alpha)n) + cn,$$

where α is a constant in the range $0 < \alpha < 1$ and $c > 0$ is also a constant and *formally prove it is correct* using the substitution method.

Solution.

b. Some recurrence relation cannot be solved using the Master Method. For example, for

$$T(n) = 2T(n/2) + n \log_2 n$$

none of the three cases of the Master Method were appropriate.

Instead, use the recurrence tree method and give the asymptotically tight bound for the above recurrence relation. *You only need to formally prove that the lower bound is tight it formally.*

Solution.

c. Do the same as the previous part for the following recurrence relation

$$T(n) = 2T(n/2) + n \log_2^k n$$

where $k \in \mathbb{N}$ is a fixed constant with $k \geq 1$.

Solution.

Q2. [10 Points] Dynamic Programming *(maximum 3 pages)*

This is a preview on a topic related to recursive algorithms that you will learn more about in CSC373. In this question we consider the *rod-cutting* problem.

You operate a company which buys n inches long steel rods, cuts them into shorter rods (or leaves them the same length), and then sell the cut segments. All cuts are free. Your job is to determine the best way to cut the rods. You are given as input a pricing table which lists the price p_i you can sell an i inch long rod (rod lengths are always an integral number of inches).

Table 1 gives an example pricing table for $n = 4$. Note that there are many different ways to cut the rod. If it is sold as is (no cuts) then we get \$8. If we cut it into two pieces of length one and three respectively, we would get \$7. If we cut it into two pieces both two inches long, we would get \$10. If we cut it into four pieces each of which is one inch long then we would get \$4. It follows, for this pricing table, that the optimum value \$10 is obtained by cutting the full 4 inch rod into two pieces both two inches in length.

Length i	1	2	3	4
Price p_i	1	5	6	8

Table 1: Example pricing table for input rods of length 4.

a. Consider the following *greedy algorithm*[1]. We compute the value per unit length (e.g., for the pricing table shown in Table 1, these "per-unit values" are shown in Table 2) and cut off a length with maximum price per unit length which fits in the remaining part of the rod.

Length i	1	2	3	4
Price p_i per unit length	1	2.5	2	2

Table 2: Price per unit length for input rods of length 4 and pricing table as shown in Table 1.

For the pricing table shown in Table 1, the greedy algorithm would return the optimum cut into two rods of length two inches.

Give a counter-example for the optimality of this algorithm, i.e., give a pricing table where the greedy algorithm *does not* obtain the best value.

Solution.

b. For this part and the next part, define $OPT(i)$ to be the maximum value we can obtain by cutting a rod which is i inches long. Let n be the length of the uncut rod in inches and $p_1, ..., p_n$ be the value of selling rods of different lengths (i.e., $p_i > 0$ is the value of selling a rod of length i). What is $OPT(1)$ with respect to the p_is?

Solution.

c. Express $OPT(k+1)$ with respect to $OPT(1), ..., OPT(k)$ and the p_is. Justify your recurrence relation.

Solution.

d. *Memoization*[2] is a technique of storing pre-computed values so that the same computation executed later runs in constant time. This is typically implemented by keeping a global look-up table which all recursive calls can update. Use memoization and the previous two parts to come up with an algorithm which solves the rod-cutting problem.

Write your pseudo-code in Algorithm 2 (you can change the input and output parameters to suit your needs).

e. Determine the upper bound on the running time of your algorithm by determining (1) the size of the look-up table T and (2) the time it takes to fill an entry of T.

Solution.

Additional Questions

Remember to comment out this section when submitting your assignment.

If you would like more exercises consider trying the following problems from your primary and supplementary textbooks. *We will not be providing solutions to these questions* though you are free to find the solution online and discuss them with your peers.

[1] *Greedy algorithms* are a class of algorithms where we assign a value to each element of the input and at each iteration take the element which has the best value subject to some constraint. Prims' algorithm, that we saw in-class, is a greedy algorithm as we always add the *lightest weight edge* subject to the constraint that it is incident to one of the vertices currently in our tree.

[2] Read more about memoization. You can efficiently memoize any function in Python using a decorator.

Algorithm 2 CutRod(G)

Require: Pre-condition.
Ensure: Post-condition.

```
u ← 1                                   ▷ this is how you assign variables
for i ∈ [n] do                          ▷ this is a for-loop
    while i > 0 do                      ▷ this is a while-loop
        if i < 10 then                  ▷ this is an if-else-statement
            u ← 2
        else
            u ← 3
        end if
    end while
end for
return A, B                             ▷ this how you return something
```

1. David Liu's notes Chapter 4: Exercises 2, 3, 4, 5, 7, 8
2. Vassos' notes Chapter 2: Exercises 4, 5, 7.
3. CLRS Chapter 4.4: 4, 5, 6, 7