CSC510 Homework 2

Chris Camano: ccamano@sfsu.edu

October 7, 2022

1. Consider the recurrence

$$T(n) = 2T(n/3) + O(n2).$$

Using the method of recursion trees, draw three layers (root, children, grandchildren) of the recursion tree and use it to give a big-O estimate for T(n). (You may upload a photo of a drawing for this problem)

$$\begin{aligned} T(n) =& 2T\left(\frac{n}{3}\right) + cn^2 \\ & 2\left(2T\left(\frac{n}{9}\right) + \frac{cn^2}{9}\right) + cn^2. \\ & 2\left(2\left(2T\left(\frac{n}{81}\right) + \frac{cn^2}{81}\right) + \frac{cn^2}{9}\right) + cn^2. \end{aligned}$$

$$T(n) = \sum_{i=0}^{L} r^i f\left(\frac{n}{c^i}\right) = \sum_{i=0}^{L} 2^i c\left(\frac{n}{3^i}\right)^2 = cn^2 \sum_{i=0}^{L} \left(\frac{2}{9}\right)^i$$

To solve for the height of the recursion tree L we solve

$$L := \left(\frac{h}{3^i}\right) = 1 = \log_3(n)$$

The final layer has alternate cost then layers above so so let $L = \log_3(n) - 1$

There are a total of $2^{(\log_3(n))}$ nodes at layer $T(1)$ $\therefore$ T(n)=

$$\sum_{i=0}^{\log_3(n)-1} \left(\frac{2}{9}\right)^i cn^2 + \theta\left(2^{\log_3(n)}\right)$$

$$\sum_{i=0}^{\log_3(n)-1} \left(\frac{2}{9}\right)^i cn^2 + \theta\left(n^{log_3(2)}\right)$$

$$\left[\frac{1 - \left(\frac{2}{9}\right)^{\log_3(n)}}{\frac{7}{9}}\right] + \theta\left(n^{\log_3(2)}\right)$$

However,

$$\sum_{i=0}^{\log_3(n)-1}\left(\frac{2}{9}\right)^i cn^2 = \left[\frac{1-\left(\frac{2}{9}\right)^{\log_3(n)}}{\frac{7}{9}}\right] < \sum_{i=1}^{\infty}\left(\frac{2}{9}\right)^i cn^2 = \frac{1}{\frac{7}{9}}cn^2 = \frac{9}{7}cn^2$$

$$T(n) = \frac{9}{7}cn^2 + \theta(n^{\log_3(2)})$$

But note that $n^{\log_3(2)} \in O(n^2)$So we have that T(n) is $O(n^2)$ Also note that the recursion tree is root heavy confirming our analysis

2. Use the master theorem to solve each of the following recurrences:

   (a) $A(n) = 5A(n/3) + \Theta(n)$

   $$a = 5, b = 3, k = 1$$
   $$l = \log_3(5) \approx 1.46$$
   $$l > k \therefore$$
   $$A(h) = \theta\left(n^{\log_3(5)}\right)$$

   (b) $B(n) = 8B(n/2) + \Theta\left(n^3\right)$

   $$q = 8, b = 2, k = 3$$
   $$l = \log_2 8 = 3$$
   $$l = k \therefore$$
   $$B(n) = \theta\left(n^3 \log(n)\right)$$

   (c) $C(n) = 2C(n/9) + \Theta(\sqrt{n})$

   $$a = 2, b = 9, k = 1/2$$
   $$l = \log_9(2) \approx .31$$
   $$l < k$$
   $$c(n) = \theta\left(n^{1/2}\right)$$

   (d) $D(n) = 4D(n/4) + \Theta(1)$

   $$a = 4, b = 4, k = 0$$
   $$l = \log_4(4) = 1$$
   $$l > k$$
   $$D(n) = \theta(n)$$

   (e) $E(n) = 21E(n/5) + \Theta\left(n^2\right)$

   $$a = 21, b = 5, k = 2$$
   $$l = \log_5(21) \approx 1,89$$
   $$l < k$$
   $$E(n) = \theta\left(n^2\right)$$

3. Hindsight is $20/20$ when it comes to investing in Pokemon cards. Suppose you're given an array $P[1,\ldots,n]$, where $P[i]$ is the price of a certain card on day i. The goal is to find the best days to buy and sell in order to maximize your profit. The return value of your algorithms should be the profit made, i.e., the difference between the selling price and the buying price.

   You may assume that you're only buying and selling one card, the price doesn't change during a given day, and that the card needs to be bought before it is sold. In other words, you are trying to maximize $P[j]-P[i]$ over all pairs of indices i and j, where $i \leq j$. For example, if the array of prices is

   $$[2,4,3,5,1],$$

   the maximum profit is 3 , which is achieved when you buy on day 1 and sell on day 4 . You can't make a profit of 4 because 1 comes after 5.

   For both parts below, write down an algorithm in pseudocode that takes in an array $P[1,\ldots,n]$ of integers.

   (a) Design a brute force algorithm for solving this problem. How long does your algorithm take? This Algorithm has a time complexity of $\mathcal{O}(n^2)$ since we have two for loops running here.

Algorithm 1 PokeProfit(P[1,...,n])

```
Profit:=0
for i=1,...,n do
    for j=i+1,..n do
        profit:=maximum(profit,P[j]-P[i])
    end for
end for
```

   (b) Use divide and conquer to design an algorithm that runs in time $O(n\log n)$ (hint: see the maximum subarray problem from class). Give a brief justification for why your algorithm works. Justify the runtime of your algorithm. (Yes, there is a faster way of doing this problem.)

Algorithm 2 PokeProfitDnC(P[1,...,n])

```
if n=1 then
    return 0
end if
y=⌊(n/2)⌋
bestL=PokeProfitDnC(P[1,...,y])
bestR=PokeProfitDnC(P[y+1,...,n])
alt=maximum(P[y+1,...,n])-minimum(P[1,...,y])
return maximum(bestL,bestR,alt)
```

   This algorithm works by reducing the problem space into two subdivisions and then recursivley finding local solutions. Then after identifying these local solutions a global solution is found by computing the maximum and minimum over the two paritions. Of these three cases one of them is the solution to the problem which is evaluated using a maximum argument on the return statement. The runtime of this algorithm is$\mathcal{O}(n\log n)$ Logarithimic time for the rescurisve structure and linear time from the maximum array traversals.

4. Use Karatsuba's algorithm to multiply 4671 and 8535 . Work out all the recursive calls (i.e., including multiplications involving two 2-digit numbers) using the same algorithm. How many 1-digit multiplications did you perform? How many 1-digit multiplications would regular "elementary school" multiplication use? (Hint: regular multiplication is the same as the first divide and conquer algorithm we gave in class) Since I cannot draw out a nice tree stucture I will manually trace the multiplication paths:

$$4671(8535) \mapsto \quad \{(46\times 85), -(25\times 14), (71\times 35)\}$$

$$(46\times 85) \mapsto \quad \{(4\times 8), -(2\times 3), (6\times 5)\} \mapsto \quad (10^2(32)+10(32+30-(-6))+30) = 3910$$

$$-(25\times 14) \mapsto \quad \{(2\times 1), (-3\times -3), (5\times 4)\} \mapsto \quad (10^2(2)+10(2+30-(9))+20) = -350$$

$$(71\times 35) \mapsto \quad \{(7\times 3), (6\times -2), (1\times 5)\} \mapsto \quad (10^2(21)+10(21+5-(-12))+5) = 2485$$

$$4671(8535) \mapsto (10^4(3910)+10^2(3910+2485-(-350))+2485) = 39866985$$

   (a) How many 1-digit multiplications did you perform?
   9 one digit computations were computed 3 for each child node

   (b) How many 1-digit multiplications would regular "elementary school" multiplication use? (Hint: regular multiplication is the same as the first divide and conquer algorithm we gave in class)
   Since regular elementary arithmetic is $\mathcal{O}(n^2)$ we expect to see 16 computations. Here to check our work since this algorithm is $\mathcal{O}(n^{log_2(3)})$ we see

   $$4^{log_2(3)} = 9$$

   as expected.