

CS375 Weekly Assignment 5

Due Mar 17th 10PM ET

A general note for CS375: When writing up your homework, please write neatly and explain your answers clearly, giving all details needed to make your answers easy to understand. Graders may not award full credit to incomplete or illegible solutions.

You are encouraged to discuss the problems with your classmates, **but your submission should be your own**. When writing your solutions up, you should not be looking at any other student's submission.

Clear communication **is the point**, on every assignment. In general in CS375, unless explicitly specified otherwise, answers should be accompanied by explanations. Answers without explanations may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

Necessary acknowledgements: many of these questions are reproductions or adaptations of questions of Eric Aaron's devising. Others are from our course texts as listed on the syllabus.

1. In 1854, archaeologists discovered Sumerians clay tablets, carved around 2000 BCE, that list the squares of integers up to 59. This discovery led some scholars to conjecture that ancient Sumerians performed multiplication by reduction to squaring, using an identity like $x \times y = x^2 + y^2 - (x - y)^2$. Unfortunately, those same scholars are silent on how the Sumerians supposedly squared larger numbers. Four thousand years later, we can finally rescue these Sumerian mathematicians from their lives of drudgery through the power of recursion!

- (a) Describe a variant of Karatsuba's algorithm that squares any n -digit number in $\Theta(n^{\lg 3})$ time (by reducing it to squaring three $\frac{n}{2}$ digit numbers). (Karatsuba actually did this in 1960.) Use the master theorem to verify that your runtime is correct.

Solution: Given an n digit number x , we'll write $x = a10^{\frac{n}{2}} + b$ where a and b are $\frac{n}{2}$ digit numbers. Then

$$\begin{aligned}x^2 &= (a10^{\frac{n}{2}} + b)^2 \\&= a^2 10^n + 2ab10^{\frac{n}{2}} + b^2 \\&= a^2 10^n + ((a+b)^2 - a^2 - b^2)10^{\frac{n}{2}} + b^2\end{aligned}$$

So by computing a^2 , b^2 , and $(a+b)^2$ and by spending the linear time to combine them in the way above, we get the following recursive formula for the runtime: $T(n) = 3T(\frac{n}{2}) + \Theta(n)$. Note that $\log_2(3) > 1.58$, so $n^1 = O(n^{\log_2(3)-\epsilon})$ for any $\epsilon < .58$, which by the Master Theorem we get that $T(n) = n^{\log_2(3)}$.

- (b) Describe a recursive algorithm that squares any n digit number in $\Theta(n^{\log_3(6)})$ time. As a bit of a hint, if you are trying to compute x^2 where $x = 10^{\frac{2n}{3}}a + 10^{\frac{n}{3}}b + c$ and a , b , and c are all $\frac{n}{3}$ digits in length, try computing:
 - i. a^2
 - ii. b^2
 - iii. c^2
 - iv. $(a+b)^2$
 - v. $(b+c)^2$
 - vi. $(a+c)^2$

and then putting them together in a nice way. Use the master theorem or a recursion tree to verify the correctness of your runtime.

Solution: As above,

$$\begin{aligned} x^2 &= (10^{\frac{2n}{3}}a + 10^{\frac{n}{3}}b + c)^2 \\ &= a^2 10^{\frac{4n}{3}} + 2ab 10^{\frac{3n}{3}} + (b^2 + 2ac) 10^{\frac{2n}{3}} + 2bc 10^n 3 + c^2 \\ &= a^2 10^{\frac{4n}{3}} + ((a+b)^2 - a^2 - b^2) 10^{\frac{3n}{3}} + (b^2 + (a+c)^2 - a^2 - c^2) 10^{\frac{2n}{3}} \\ &\quad + ((b+c)^2 - b^2 - c^2) 10^{\frac{n}{3}} + c^2 \end{aligned}$$

So by computing the above 6 squares and spending the linear time necessary to recombine all the parts as described above, we can build back x^2 . By following this recursively, our runtime is thus given by $T(n) = 6T(\frac{n}{3}) + \Theta(n)$. Note that $\log_3(6) = 1.63\dots$, so since $n^1 = O(n^{\log_3(6)-.53})$ by the master theorem we get that $T(n) = n^{\log_3(6)}$, which is definitely worse than $n^{\log_2(3)} = n^{1.58\dots}$, so we've made negative progress.

- (c) Unfortunately, $\lg 3 < \log_3(6)$, so we've made negative progress. Let's fix that by squaring in $\Theta(n^{\log_3(5)})$ time. Using the same notation as above, compute

- i. a^2
- ii. $(4a + 2b + c)^2$
- iii. c^2
- iv. $(a + b + c)^2$
- v. $(a - b + c)^2$

and put the terms together nicely. Use the master theorem or a recursion tree to verify the correctness of your runtime.¹

Solution: Consider the following values:

$$\begin{aligned} \alpha &= (a + b + c)^2 = a^2 + b^2 + c^2 + 2(ab + bc + ac) \\ \beta &= (a - b + c)^2 = a^2 + b^2 + c^2 + 2(ac - bc - ab) \\ \gamma &= (4a + 2b + c)^2 = 16a^2 + 4b^2 + c^2 + 2(8ab + 4ac + 2bc) \\ \delta &= \frac{\alpha + \beta + -a^2 - c^2}{2} = b^2 + 2ac \\ \epsilon &= \frac{\alpha - \beta}{2} = 2(ab + bc) \\ \zeta &= \frac{\gamma - 2\epsilon - 4\delta - 16a^2 - c^2}{6} = 2ab \end{aligned}$$

As a result, we can write:

$$\begin{aligned} x^2 &= (10^{\frac{2n}{3}}a + 10^{\frac{n}{3}}b + c)^2 \\ &= a^2 10^{\frac{4n}{3}} + 2ab 10^{\frac{3n}{3}} + (b^2 + 2ac) 10^{\frac{2n}{3}} + 2bc 10^n 3 + c^2 \\ &= a^2 10^{\frac{4n}{3}} + \zeta 10^{\frac{3n}{3}} + \delta 10^{\frac{2n}{3}} + (\epsilon - \zeta) 10^n 3 + c^2 \end{aligned}$$

Note that all the above terms are computable from the recursed terms in linear time, albeit a big linear time. Since we're making 5 recursive calls on inputs of size $\frac{n}{3}$, this gives us the following recursive runtime formula: $T(n) = 5T(\frac{n}{3}) + \Theta(n)$. Since $\log_3(5) = 1.46\dots$, we know that $n^1 = O(n^{\log_3(5)-\epsilon})$ for $\epsilon < .46$, which by the master theorem we get that $T(n) = \Theta(n^{\log_3(5)})$.

2. Suppose we are given a set $S = \{x_1 = (v_1, w_1), \dots, x_n = (v_n, w_n)\}$ of n items where v_i denotes the value of x_i and w_i denotes the weight of x_i . For any element x , let $S_{<x}$ be the set of items of less value than x and let $S_{>x}$ be the set of items of more value than x . For any subset R of S , let $w(R)$ denote the

¹This was originally going to be your exam question :)

sum of the weights of elements in R . We'll say a **weighted median** of S is *any* element x in S that satisfies $w(S_{<x}) \leq \frac{w(S)}{2}$ and $w(S_{>x}) \leq \frac{w(S)}{2}$.

Describe and analyze a recursive algorithm for computing a weighted median of a set in $O(n)$ time.

Solution: The basic algorithm will look quite similar to the algorithm for SELECTION from class - the main difference will be just that we need to keep track of the weights of values we've 'discarded' from previous recursive steps. Consider the following pseudocode:

```
def weighted_median(S[1, ..., n], lower_weight, upper_weight):
    if n <= 10:
        just calculate it manually
    total_weight = lower_weight + upper_weight
    for i in range(n):
        total_weight += w[S[i]]

    medians = []
    for i in range(n / 5):
        add weighted_median(S[5*i: 5*i + 5]) to medians

    median_of_medians = weighted_medians(medians, 0, 0)

    mom_index = index of median_of_medians in S

    # Partition S around our median_of_medians BY VALUE
    mom_index = Partition(S, mom_index)

    # This assumes that the weights have also been permuted by Partition
    left_weight = sum([w[s] for s in S[:mom_index]]) + lower_weight
    right_weight = sum([w[s] for s in S[mom_index + 1:]]) + upper_weight

    if left_weight <= total_weight / 2 and right_weight <= total_weight / 2:
        return median_of_medians
    else if left_weight > total_weight / 2:
        right_weight += w[median_of_medians]
        return weighted_median(S[:mom_index], lower_weight, right_weight)
    else: # if right_weight > total_weight / 2:
        left_weight += w[median_of_medians]
        return weighted_median(S[mom_index:], left_weight, upper_weight)
```

The initial call to the algorithm will be `weighted_median(S[1, ..., n], 0, 0)`.

First, let's prove this thing is correct: namely, that `weighted_median(S[1, ..., n], lower_weight, upper_weight)` returns an item s such that

$$\text{lower_weight} + \sum_{t \text{ in } S_{<s}} w[t] \leq \frac{\text{total_weight}}{2} \geq \text{upper_weight} + \sum_{t \text{ in } S_{>s}} w[t]$$

where $\text{total_weight} = \text{lower_weight} + \text{upper_weight} + \sum_{i=1}^n w[S[i]]$.

There is a little bit of subtlety to quickly assess - do all lists have a weighted median? We need to be a little more precise - if I specify an additional `lower_weight` and `upper_weight`, is there a weighted median satisfying the above inequalities? I'm going to leave that part as an exercise for you guys, but the proof just amounts to sorting the input by value and asking some basic questions about where the weighted median has to be.

So with that in mind, let's show that our algorithm is correct. We'll follow our standard inductive methods - since we just manually perform the calculation for small values, our base case is handled. So let's think about what the algorithm fundamentally does - more specifically, let's make sure that we don't delete the weighted median when we recurse. So long as we haven't deleted it, we can appeal to induction to say "we're right for smaller array sizes, so we're correct for the bigger ones too".

The main thing our algorithm does is call PARTITION - it restructures the array around the pivot we chose so that the left side of the array consists of smaller values and the right side of values that are at least as big as the pivot. Afterwards, we look at the pivot itself and sum the things that are to its

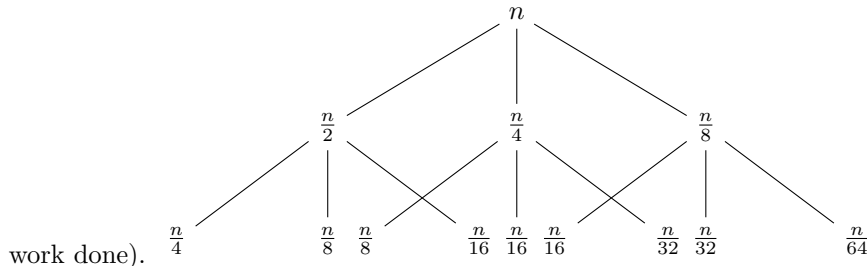
left and right and accumulate those values with the associated parameters. If the pivot is too big to be the median (if the summed weight of the things smaller than it is too big), then neither the pivot nor any value as large could be the pivot, so we can recurse on only the items of smaller value. If the pivot is too small (is the summed weight of things of value at least as large is too big), then none of the items of smaller value could be the pivot, so we can discard those. As a result, either way we recurse we are discarding items that could not be the weighted median. By induction, this guarantees that we eventually find the weighted median.

Now, let's check its runtime. It's runtime is actually precisely the same as SELECTION's - the work per function call is maybe a slightly bigger linear function, but *it's still a linear function*. This means we get the same recurrence $T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + \Theta(n)$, which we know from class gives us $T(n) = \Theta(n)$.

3. For the following functions of T , pick 2 more (that you didn't do last week) and determine their asymptotic complexity **by means of a recursion tree**. After, confirm your answer to these **2 and the 3 from last week** by means of applying the Master Theorem, **or state that the Master Theorem doesn't apply to the chosen function**. You can assume that $T(n) = 1$ for all $n < 10$ (or any reasonable constant) and that all relevant mathematical operations drop remainders (so $\frac{3}{2} = \sqrt{2} = 1$, etc). Note that I have purposely shuffled the list so that the order of these is not in increasing difficulty.

(a) $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n$

Solution: In the following recursion tree, since the input size is always equal to the work done, I'm only putting one value per node (and understanding that it is both the input size and the



As we can see, the work on

- the top level is n ,
- the second level is $\frac{7}{8}n$,
- the work on the third level is $\frac{49}{64}n$,
- and the work on the i th level is $(\frac{7}{8})^i n$.

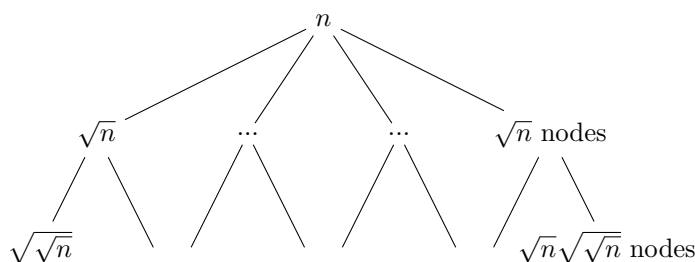
I claim that the runtime is $T(n) = \Theta(n)$: because n work happens on just the top node it's clear that $T(n) = \Omega(n)$, so we just need an upper bound to confirm it. Pretending that the tree went on for forever, the work done in the tree would be

$$\begin{aligned} T(n) &\leq \sum_{i=0}^{\infty} \left(\frac{7}{8}\right)^i n = n \sum_{i=0}^{\infty} \left(\frac{7}{8}\right)^i \\ &= n \frac{1}{1 - \frac{7}{8}} = 8n = O(n) \end{aligned}$$

Note that the master theorem doesn't apply to problems like this as it doesn't fit the form $aT(\frac{n}{b}) + \Theta(f(n))$.

(b) $T(n) = \sqrt{n}T(\sqrt{n}) + n$

Solution:



Going level by level, we get:

- On level 0 (the root), there is one node of n work, giving n total work on this level.
- On level 1 (below the root), there are $\sqrt{n}$ nodes, each of work $\sqrt{n}$, making the total work n on this level.
- On level 2, there are $\sqrt{n}\sqrt{\sqrt{n}}$ nodes, each with $\sqrt{\sqrt{n}}$ work to be done, giving $\sqrt{n}\sqrt{\sqrt{n}}\sqrt{\sqrt{n}} = \sqrt{n}\sqrt{\sqrt{n}^2} = \sqrt{n}\sqrt{n} = n$ total work...
- Each level, there's n work! Showing this by induction would be a nice exercise.

So now we just need to determine the number of levels. Notice that the input size goes $n, \sqrt{n}, \sqrt{\sqrt{n}}$, or phrased with exponents that's $n^1, n^{\frac{1}{2}}, n^{\frac{1}{4}}, n^{\frac{1}{8}} \dots$. So how many times do we need to apply the radical to get down to a small constant, say 2? That involves solving the equation:

$$2 = n^{(\frac{1}{2})^L}$$

$$1 = \left(\frac{1}{2}\right)^L \lg(n)$$

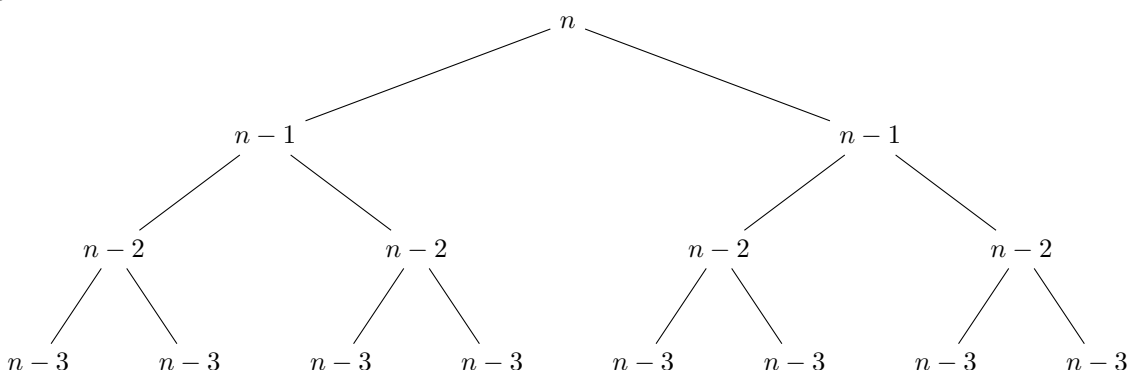
$$2^L = \lg n$$

$$L = \lg \lg n$$

As a result, we get the number of levels to this recursion tree is $\lg \lg n$, so the result is $T(n) = n \lg \lg n$.

(c) $T(n) = 2T(n-1) + 1$

Solution: In the following recursion tree I'll only track in the input size since all nodes have just 1 unit of work:



Since the input is only decreasing by 1 per level, there are n levels to this tree. The amount of work per level doubles each time you go down, giving us $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ total work. As with the two previous functions, the master theorem doesn't apply as the function's form doesn't fit the form $aT(\frac{n}{b}) + \Theta(f(n))$.

(d) $T(n) = 8T(\frac{n}{2}) + n^3\sqrt{n}$

Solution: Let's start with the master theorem on this one, since we finally have a case that will apply. Note that $n^3\sqrt{n} = n^3 \cdot 5 = n^{3.5}$ and $\log_b(a) = \log_2(8) = \log_2(2^3) = 3$, so the work-function is bigger. Recall that the 'bigger' case in the master theorem says that: if $T(n) =$

$aT\left(\frac{n}{b}\right) + \Theta(f(n))$ where $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ for some $\epsilon > 0$ and if $af\left(\frac{n}{b}\right) < kf(n)$ for some $k < 1$, then $T(n) = \Theta(f(n))$. First, note that $f(n) = \Omega(n^{\log_2(8)+.5})$, which handles the first condition. Next, $af\left(\frac{n}{b}\right) = 8\left(\frac{n}{2}\right)^3 \cdot 5 = \frac{8}{2^{3.5}}n^{3.5} = \frac{8}{8\sqrt{2}}n^{3.5} < \frac{8}{9}f(n)$ where $\frac{8}{9} < 1$, handling the second condition. This tells us that $T(n) = \Theta(n^3\sqrt{n})$.

Let's do the recursion tree to double-check. Your tree should be branching out with 8 children per node with the input size halving at each level. This gives us that the total work is:

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_2(n)} 8^i \left(\frac{n}{2^i}\right)^3 \cdot 5 = n^{3.5} \sum_{i=0}^{\log_2(n)} \frac{8^i}{(2^i)^{3.5}} \\ &= n^{3.5} \sum_{i=0}^{\log_2(n)} \frac{8^i}{(2^3 \cdot 5)^i} = n^{3.5} \sum_{i=0}^{\log_2(n)} \frac{8^i}{(2^3 \sqrt{2})^i} \\ &= n^{3.5} \sum_{i=0}^{\log_2(n)} \frac{8^i}{8^i \sqrt{2}^i} = n^{3.5} \sum_{i=0}^{\log_2(n)} \frac{1}{\sqrt{2}^i} \\ &= \Theta(n^{3.5}) \end{aligned}$$

which matches the answer from the master theorem.

(e) $T(n) = 2T\left(\frac{n}{2}\right) + n \lg n$

Solution: Let's start with the recursion tree. We'll get a binary tree where the input size is going down by a factor of 2 (halving) each level downwards, giving us the following summation of work:

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_2(n)-1} 2^i \frac{n}{2^i} \lg\left(\frac{n}{2^i}\right) = \sum_{i=0}^{\log_2(n)} n \lg\left(\frac{n}{2^i}\right) \\ &= n \sum_{i=0}^{\lg n-1} (\lg n - i) = n \sum_{i=0}^{\lg n-1} \lg n - n \sum_{i=0}^{\lg n-1} i \\ &= n \lg(n) \lg(n-1) - n \frac{\lg(n) \lg(n-1)}{2} \\ &= n \frac{\lg(n) \lg(n-1)}{2} = \Theta(n \lg^2 n) \end{aligned}$$

We can also get there from the master theorem: observe that $n^{\log_2(2)} = n^1$, so $f(n) = n \lg n = \Theta(n^{\log_2(2)} \log^1(n))$, so we get that $T(n) = \Theta(f(n) \log n) = \Theta(n \log^2 n)$, which matches the bound above, as desired.