Recitation 1

Solving Recurrences

1.1 Announcements

- Welcome to 15210!
- The course website is http://www.cs.cmu.edu/~15210/. It contains the syllabus, schedule, library documentation, staff contact information, and other useful resources.
- We will be using Piazza (https://piazza.com/) as a hub for course announcements and general questions pertaining to the course. Please check it frequently to make sure you don't miss anything.
- The office hours schedule is posted on the course website as well as Piazza. Come meet all of your TAs!
- The first homework assignment, *IntegralLab*, has been released! It's due **Friday at 5pm**, but don't worry – it's quite short.
- Homeworks will be distributed through Autolab (https://autolab.andrew.cmu.edu/). Most homework assignments will be released on Fridays and will be due one week later. You will submit coding tasks on Autolab.
- This semester, we will begin integrating Diderot (https://diderot-cmu.appspot.com/) into the course. Interactive lecture notes will be available on Diderot, as well as possibly written assignments.

1.2 The Tree Method

The cost analysis of our algorihms usually comes down to finding a closed form for a recurrence. Using the tree method to derive the closed form consists of finding a cost bound for each level of the recursion tree and then summing the costs over the levels.

Task 1.1. *Using the tree method, solve the following recurrences:*

$$f(n) = 4f\left(\frac{n}{2}\right) + n^2$$
$$f(n) = 2f(n-1) + 1$$
$$f(n) = 2f\left(\frac{n}{2}\right) + n^{\frac{1}{4}}$$

$$f(n) = 4f\left(\frac{n}{2}\right) + n^2$$

- Counting from level $i = 0$ at the root, the *i*th level of the recursion tree has 4^i nodes.
- Each of these nodes performs $(\frac{n}{2^i})^2$ work.
- Thus, each level i of the tree has a cost of $4^i(\frac{n}{2^i})^2 = n^2$.
- Summing up across $\log_2 n$ levels as the problem size decreases by 2 every time, we obtain the final answer: $\sum_{i=0}^{\log n-1} n^2 = n^2 \log n \in \Theta(n^2 \log n)$.

$$f(n) = 2f(n-1) + 1$$

- In this recurrence, the *i*th level of the recursion tree has 2^i nodes.
- Each of these nodes has a cost of 1.
- Thus, each level has a cost of 2^i at level i.
- As the problem size decreases by 1 each time, there are n levels.
- Summing across the n levels: $\sum_{i=0}^{n-1} 2^i = \frac{1-2^n}{1-2} = 2^n - 1 \in \Theta(2^n)$.

$$f(n) = 2f\left(\frac{n}{2}\right) + n^{\frac{1}{4}}$$

- The *i*th level of the recursion tree in this example also has 2^i nodes.
- Each of these nodes has a cost of $(\frac{n}{2^i})^{frac14}$.
- Each level then costs $2^i(\frac{n}{2^i})^{frac14})$, which simplifies to $(2^{\frac{3}{4}})^i n^{\frac{1}{4}}$.
- As the problem size is divided by 2 each time, the number of levels is $\log_2 n$.
- Summing across the levels to find the closed form: $\sum_{i=0}^{\log n-1}(2^{\frac{3}{4}})^i n^{\frac{1}{4}} = n^{\frac{1}{4}}\sum_{i=0}^{\log n-1}(2^{\frac{3}{4}})^i = n^{\frac{1}{4}}\frac{1-(2^{\frac{3}{4}})^{\log n}}{1-2^{\frac{3}{4}}} = n^{\frac{1}{4}}\frac{n^{\frac{3}{4}}-1}{2^{\frac{3}{4}}-1} \in \Theta(n)$.

1.3 The Brick Method

If the cost at every successive level is a multiplicative factor away from the cost of the previous level, we can use the brick method. First, determine whether the recurrence conforms to one of the three cases below and then apply the next step for that case. Otherwise, use the tree or the substitution method.

Definition 1.2. ***Balanced*** *The cost of every level is roughly equal. With a maximum cost of* L *and* d *levels, the total cost will be* $O(dL)$.

Root Dominated *Each level is a constant factor smaller than the previous level. With a cost of* L *at the root, the total cost will be* $O(L)$.

Leaf Dominated *Each level is a constant factor larger than the previous level. With a cost of* L_d *at the bottom-most (*d*th) level, the total cost will be* $O(L_d)$.

Task 1.3. *Solve the following recurrences using the brick method:*

$$f(n) = 2f\left(\frac{n}{4}\right) + \sqrt{n}$$
$$f(n) = f(\sqrt{n}) + \log n$$
$$f(n) = 2f(\sqrt{n}) + 1$$

$$f(n) = 2f\left(\frac{n}{4}\right) + \sqrt{n}$$

- We start from level $i = 0$. The ith level of the recursion tree has 2^i nodes.
- Each node has a cost of $\sqrt{\frac{n}{4^i}}$. The cost of each level is then $2^i\sqrt{\frac{n}{4^i}} = \frac{2^i}{2^i}\sqrt{n} = \sqrt{n}$.
- Since each level has the same cost, the recursion tree is **balanced**.
- Following the definition, with $\sqrt{n}$ cost per level and $\log_4 n$ levels, the final answer is $\in \Theta(\sqrt{n}\log n)$.

$$f(n) = f(\sqrt{n}) + \log n$$

- Starting at level $i = 0$, there is 1 node at each level. The cost of that node is $\log(n^{\frac{1}{2}})^i$, or simplified $\frac{1}{2^i}\log n$.

- Thus, each following level has a cost of $\frac{1}{2}$ of the previous level and the recursion tree is **root dominated**.
- The final result is simply the cost at the root of the tree: $\Theta(\log n)$.

$$f(n) = 2f(\sqrt{n}) + 1$$

- There are once more 2^i nodes at each level i, from $i = 0$. The cost at each node is 1.
- The cost at each level is then the number of nodes: 2^i. In other words, each following level has twice the work of the previous level.
- As the cost at each level geometrically increases, this recurrence is **leaf dominated** and the total cost is the number of leaves at the last level d, 2^d.
- How many levels are there if the problem size has a square root function applied to it? When the problem size is divided and we are trying to find out the number of levels, we are solving the problem $\frac{n}{2^d} = 1$ for d where d is the lowest level. In other words, we are solving for how many divisions we need to make until the problem size is 1.
- In this case, we can use the same approach, but to find when the problem size is 2 for simplicity:

$$\begin{aligned} n^{\frac{1}{2^d}} &= 2 \\ \log_2 n^{\frac{1}{2^d}} &= \log_2 2 \\ \frac{1}{2^d} \log_2 n &= 1 \\ \log_2 n &= 2^d \\ \log_2 \log_2 n &= d \end{aligned}$$

- The number of leaves at the $\log \log n$th level is $2^{\log \log n} = \log n$; the final answer is $\Theta(\log n)$.

1.4 The Substitution Method.

We can also use mathematical induction to solve recurrences. If you want to go via this route (and you don't know the answer a priori), you'll need to guess the answer first and check it. Since this technique relies on guessing an answer, you can sometimes fool yourself by giving a false proof. The following are some tips:

1. Spell out the constants. Do not use big-O—we need to be precise about constants, so big-O makes it super easy to fool ourselves.
2. Be careful that the induction goes in the right direction.
3. Add additional lower-order terms, if necessary, to make the induction go through.

Task 1.4. *Solve the following recurrence using (strong) induction:*

$$W(n) = 2W\left(\frac{n}{2}\right) + O(n)$$

Theorem 1.5. *Let a constant $k > 0$ be given. If $W(n) \leq 2W(n/2) + k \cdot n$ for $n > 1$ and $W(1) \leq k$ for $n \leq 1$, then we can find constants κ_1 and κ_2 such that*

$$W(n) \leq \kappa_1 \cdot n \lg n + \kappa_2.$$

Proof. Let $\kappa_1 = 2k$ and $\kappa_2 = k$. For the base case ($n = 1$), we check that $W(1) \leq k \leq \kappa_2$. For the inductive step ($n > 1$), we assume that

$$W(n/2) \leq \kappa_1 \cdot \tfrac{n}{2} \lg(\tfrac{n}{2}) + \kappa_2,$$

And we'll show that $W(n) \leq \kappa_1 \cdot n \lg n + \kappa_2$. To show this, we substitute an upper bound for $W(n/2)$ from our assumption into the recurrence, yielding

$$\begin{aligned} W(n) &\leq 2W(n/2) + k \cdot n \\ &\leq 2(\kappa_1 \cdot \tfrac{n}{2} \lg(\tfrac{n}{2}) + \kappa_2) + k \cdot n \\ &= \kappa_1 n(\lg n - 1) + 2\kappa_2 + k \cdot n \\ &= \kappa_1 n \lg n + \kappa_2 + (k \cdot n + \kappa_2 - \kappa_1 \cdot n) \\ &\leq \kappa_1 n \lg n + \kappa_2, \end{aligned}$$

where the final step follows because $k \cdot n + \kappa_2 - \kappa_1 \cdot n \leq 0$ as long as $n > 1$. □

1.5 Additional Exercises

Exercise 1.6. *There is a well known deterministic linear-work algorithm for finding the kth smallest value of a set of values. It uses the median of medians as the pivot. (The median value is the value v such that if the values are sorted, v would be in the middle). You don't need to understand why the algorithm works, but to be able to analyze its costs based on a description of its steps:*

1. *If the input has 5 or fewer values, find the median by brute force, otherwise:*
2. *Group the input into $n/5$ groups of 5 and find the median of each group in parallel.*
3. *Find the median of the $n/5$ medians recursively. Call this p.*
4. *Use p to filter out $3/10^{th}$s of the values in $\Theta(n)$ work and $\Theta(\log n)$ span.*
5. *Recurse on the remaining $7/10^{th}$s of the values.*

Task 1.7.

Write down recurrences for work and span.

Solve the recurrence for work in terms of Θ.

Solve the recurrence for span in terms of Θ. Warning: this is pretty hard.

.