Recursion Tree Method

Check in and Reminders

- Assignment 3 is due tonight at 11 pm
    - Adding figures to HW
- Camera policy for Zoom from next lecture:
    - Keep camera on by default; if unable to do so send me an email /message on Slack before the start of the class
    - Won't be recorded unless you are an active speaker
    - Ensures you are engaging similar to in-person students
- Assignment 4 will be released later today
    - Jeff Erickson's book has really good coverage of recursion trees and analyzing recurrences

Divide & Conquer: Quicksort

Quicksort(A)**:**

If $|A| < 3$: Sort(A) directly

Else: choose a pivot element $p \leftarrow A$

$A_{<p}, A_{>p} \leftarrow$ Partition around p

Quicksort$(A_{<p})$

Quicksort$(A_{>p})$

- Running time depends on the rank (position in sorted order) of the pivot

Quick Sort Analysis

- Partition takes $O(n)$ time
- Let r be the rank of the pivot, then:
- $T(n) = T(r-1) + T(n-r) + O(n),\ \ T(1) = 1$
- Let us analyze some cases for r
    - **Best case:** r is the median: $r = \lfloor n/2 \rfloor$
    - **Worst case:** $r = 1$ or $r = n$
    - **In between:** say $n/10 \leq r \leq 9n/10$
- In the worst-case analysis, we only consider the worst case for r. We are looking at the difference cases, just to get a sense for it.

Quick Sort Cases

- Suppose $r = n/2$ (pivot is the median element), then
  - $T(n) = 2T(n/2) + O(n)$
  - We have already solved this recurrence
  - $T(n) = O(n \log n)$

- Suppose $r = 1$ or $r = n - 1$, then
  - $T(n) = T(n-1) + T(1) + O(n)$
  - What running time would this recurrence lead to?
  - $T(n) = \Theta(n^2)$ (notice: this is tight!)

Quick Sort Cases

- Suppose $r = n/10$ (that is, you get a one-tenth, nine-tenths split
- $T(n) = T(n/10) + T(9n/10) + O(n)$
- Let's look at the recursion tree for this recurrence

- $T(n) = T(\alpha n) + T(\beta n) + O(n)$
  - If $\alpha + \beta < 1 : T(n) = O(n)$
  - If $\alpha + \beta = 1, T(n) = O(n \log n)$

QuickSort: Theory & Practice

- We can find the **median element in** $\Theta(n)$ time
    - We will go over this today
- In practice, the constants hidden in the Oh notation for median finding are too large to use for sorting
- **Common heuristic**: median of three (pick elements from the start, middle and end and take their median)
- If the pivot is chosen **uniformly at random**
    - quick sort runs in time $O(n \log n)$ in expectation and *with high probability*
    - We will prove this in the second half of the class

Challenge Recurrence

- Solve the following recurrence:

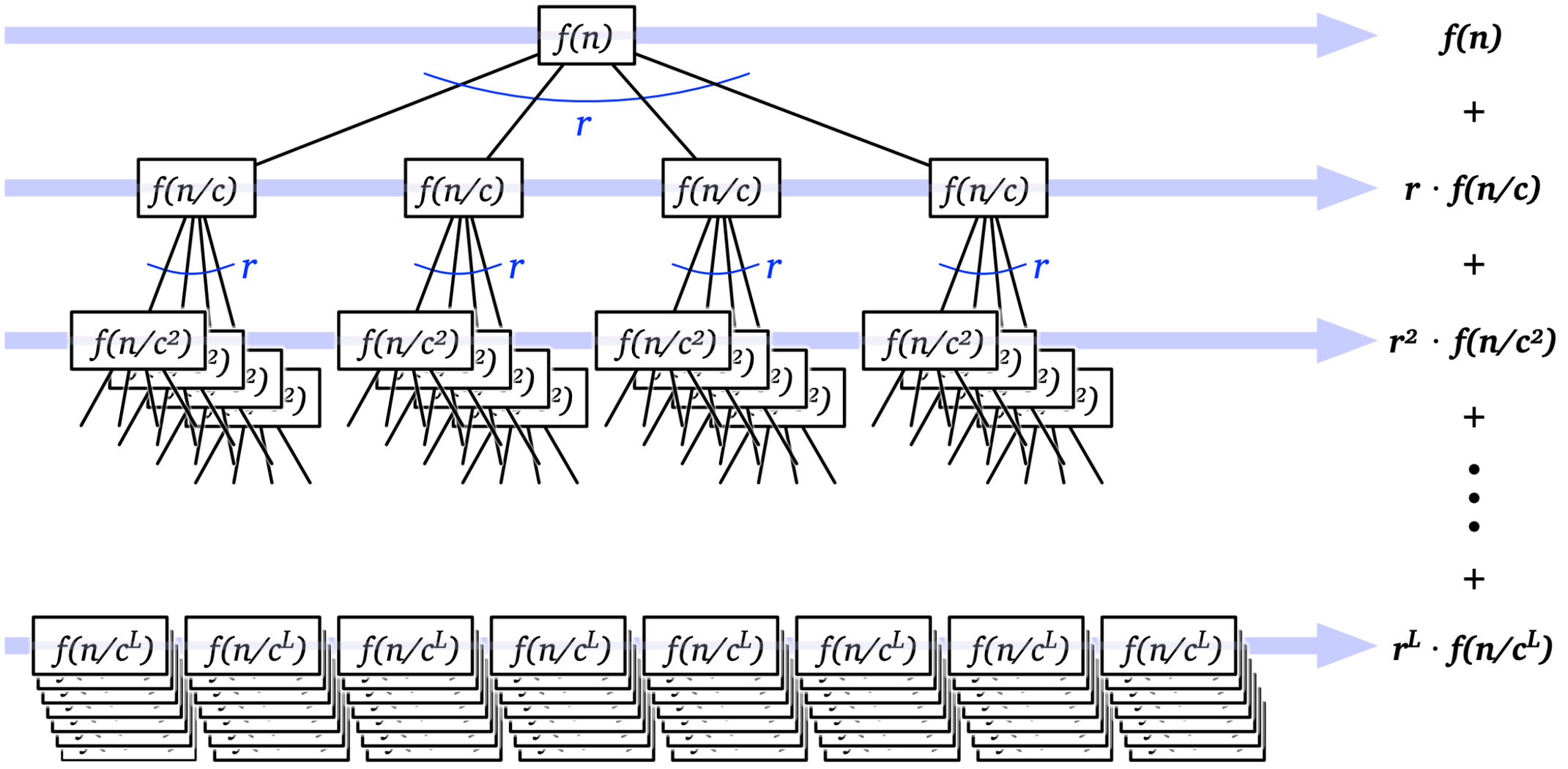
$$T(n) = \sqrt{n}T(\sqrt{n}) + n$$

- **Hint.** Try some change of variables

General Recursion Trees

- Consider a divide and conquer algorithm that
    - spends $O(f(n))$ time on non-recursive work and makes r recursive calls, each on a problem of size n/c
- Up to constant factors (which we hide in $O()$), the running time of the algorithm is given by what **recurrence**?
    - $T(n) = rT(n/c) + f(n)$
- Because we care about asymptotic bounds, we can assume base case is a small constant, say $T(n) = 1$

General Recursion Trees



A recursion tree for the recurrence $T(n) = rT(n/c) + f(n)$

- For each i, the ith level of tree has exactly r^i nodes
- Each node at level i, has cost $f(n/c^i)$

General Recursion Trees

- Running time $T(n)$ of a recursive algorithm is the sum of all the values (sum of work at all nodes at each level) in the recursion tree
- For each i, the ith level of tree has exactly r^i nodes
- Each node at level i, has cost $f(n/c^i)$
- Thus, $T(n) = \sum_{i=0}^{L} r^i \cdot f(n/c^i)$
- Here $L = \log_c n$ is the depth of the tree
- Number of leaves in the tree: $r^L = n^{\log_c r}$
- Cost at leaves: $O(n^{\log_c r} f(1))$

General Recursion Trees

- Running time $T(n)$ of a recursive algorithm is the sum of all the values (sum of work at all nodes at each level) in the recursion tree
- For each i, the ith level of tree has exactly r^i nodes
- Each node at level i, has cost $f(n/c^i)$
- Thus, $T(n) = \sum_{i=0}^{L} r^i \cdot f(n/c^i)$
- Here $L = \log_c n$ is the depth of the tree
- Number of leaves in the tree: $r^L = n^{\log_c r}$ (why?)
- Cost at leaves: $O(n^{\log_c r} f(1))$; usually just $O(r^L)$

$$r^L = r^{\log_c n} = (2^{\log_2 r})^{\log_c n} = (2^{\log_c n})^{\log_2 r} = (2^{\log_2 n})^{\frac{\log_2 r}{\log_2 c}} = n^{\log_c r}$$

Common Cases

$$T(n) = \sum_{i=0}^{L} r^i \cdot f(n/c^i)$$

Don't forget: $\sum_{i=0}^{L} a^i = \frac{a^{L+1} - 1}{a - 1}$

- **Decreasing series.** If the series decays exponentially (every term is a constant factor smaller than previous), cost at root dominates:
$$T(n) = O(f(n))$$

- **Equal.** If all terms in the series are equal:
$$T(n) = O(f(n) \cdot L) = O(f(n) \log n)$$

- **Increasing series.** If the series grows exponentially (every term is constant factor larger), then the cost at leaves dominates:
$$T(n) = O(n^{\log_c r})$$

Examples

- Use the recursion tree method to solve the following recurrences:

- $T(n) = 2(Tn/2) + n^2$
  - Notice that $f(n) = n^2$ here

- Try at home: $T(n) = 3T(n/2) + n$

Recurrences

So far we saw divide and conquer algorithms, where we split the problem in more than one subproblem.

Question. Can you think of some examples (that you have likely seen before) where we split the problem into **one** smaller subproblem?

D&C: One Smaller Subproblem

- Binary search in sorted array and binary search tree
  - $T(n) = T(n/2) + 1$
- Fast exponentiation (you may not have seen this)
  - Compute a^n, how many multiplications?
  - Naive way: $a \cdot a \cdot \ldots \cdot a$ (n times)
  - Faster way: $a^n = (a^{n/2})^2$ (suppose n is even)
  - $T(n) = T(n/2) + 1$
  - What does this solve to?
  - Think at home: What if n is odd?

Master Theorem (optional)

Set of rules to solve some common recurrences automatically

(Master Theorem) Let $a \geq 1$, $b > 1$ and $f(n) \geq 0$. Let $T(n)$ be defined by the recurrence $T(n) = aT(n/b) + f(n)$ and $T(1) = O(1)$.

Then $T(n)$ can be bounded asymptotically as follows.

- If $f(n) = n^{\log_b a - \epsilon}$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
- If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
- If $f(n) = \Omega(n^{\log_b a + \epsilon})$, for some constant $\epsilon > 0$, and if $af(n/b) \leq c_0 f(n)$ for some constant $c_0 < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Master Theorem

- It exists; it can make things easier. You don't need to know it

- OK to use in this class, but I don't encourage (nor discourage) it
    - Recursion trees promote a better understanding of the recurrence—and they can be simpler

- Master Theorem only applies to some recurrences (generalizations do exist)

Selection

Selection: Problem Statement

Given an array $A[1,\ldots,n]$ of size n, find the kth smallest element for any $1 \leq k \leq n$

- Special cases: min $k = 1$, max $k = n$:
    - Linear time, $O(n)$
- What about **median** $k = \lfloor n + 1 \rfloor / 2$?
    - Sorting: $O(n \log n)$
    - Binary heap: $O(n \log k)$

Question. Can we do it in $O(n)$?

- **Surprisingly yes.**
- Selection is easier than sorting.

Selection: Problem Statement

Example. Take this array of size 10:

$$A = 12|2|4|5|3|1|10|7|9|8$$

Suppose we want to find 4th smallest element

- First, take any pivot p from $A[1,\ldots n]$
- If p is the 4th smallest element, return it
- Else, we partition A around p and recurse

Selection Algorithm: Idea

Select (A, k):

If $|A| = 1$: return $A[1]$

Else:

- Choose a pivot $p \leftarrow A[1, \ldots, n]$; let r be the rank of p
- $r, A_{<p}, A_{>p} \leftarrow$ Partition$((A, p)$
- If $k == r$, return p
- Else:
    - If $k < r$: Select $(A_{<p}, k)$
    - Else: Select $(A_{>p}, k - r)$

Selection: Problem Statement

Example. Take this array of size 10:

$$A = 12|2|4|5|3|1|10|7|9|8$$

Suppose we want to find 4th smallest element

- Choose pivot 8
- What is its rank?
  - Rank 7
- So let's find all of the smaller elements of A:
  - $A' = 2|4|5|3|1|7$
- Want to find the element of rank 4 in this new array

Selection: Problem Statement

Example. Take this array of size 10:

$$A = 12|2|4|5|3|1|10|7|9|8$$

Suppose we want to find 4th smallest element

- Choose as pivot 3
- What is its rank?
  - Rank 3
- So let's find all of the **larger** elements of A:
  - $A' = 12|4|5|10|7|9|8$
- Want to find the element of rank $4 - 3 = 1$ in this new array

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
  - CLRS Algorithms book