Recurrences

(CLRS 4.1-4.2)

Today:

1. Introduce the divide-and-conquer algorithm technique.
2. Discuss a sorting algorithm obtained using divide-and-conquer.
3. Introduce reccurences as a means to express the running time of recursive algorithms; discuss the two ways to solve a reccurrence: recursion tree/iteration and induction.
4. Prove the Master method for solving reccurences.

1 Divide-and-conquer

We saw a couple of $O(n^2)$ algorithms for sorting. We'll see a different approach today that runs in $O(n \lg n)$. It uses one of the most powerful techniques for algorithm design, divide-and-conquer.

Divide-and-Conquer

To Solve P:

1. *Divide* P into smaller problems $P_1, P_2, P_3.....P_k$.
2. *Conquer* by solving the (smaller) subproblems recursively.
3. *Combine* solutions to $P_1, P_2, ...P_k$ into solution for P.

Analysis of divide-and-conquer algorithms and in general of recursive algorithms leads to recurrences.

2 MergeSort

- Using divide-and-conquer, we can obtain a mergesort algorithm.
  - Divide: Divide n elements into two subsequences of $n/2$ elements each.
  - Conquer: Sort the two subsequences recursively.
  - Combine: Merge the two sorted subsequences.
- Assume we have procedure Merge(A, p, q, r) which merges sorted A[p..q] with sorted A[q+1....r]
- We can sort A[p...r] as follows (initially p=1 and r=n):

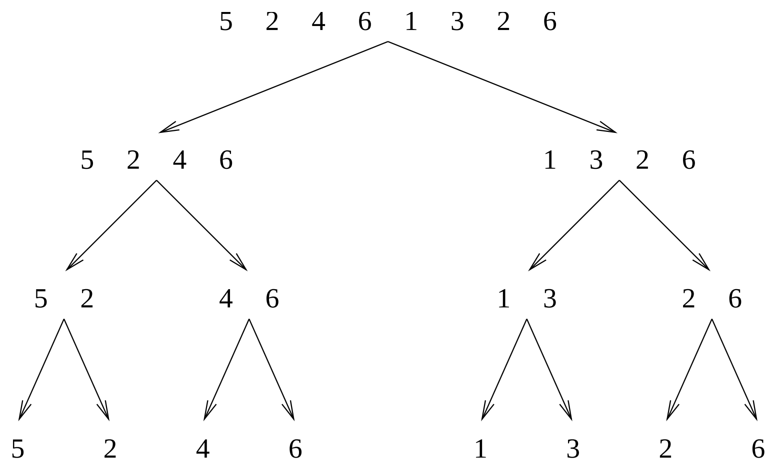
```
Merge Sort(A,p,r)
    If p < r then
        q = ⌊(p + r)/2⌋
        MergeSort(A,p,q)
        MergeSort(A,q+1,r)
        Merge(A,p,q,r)
```

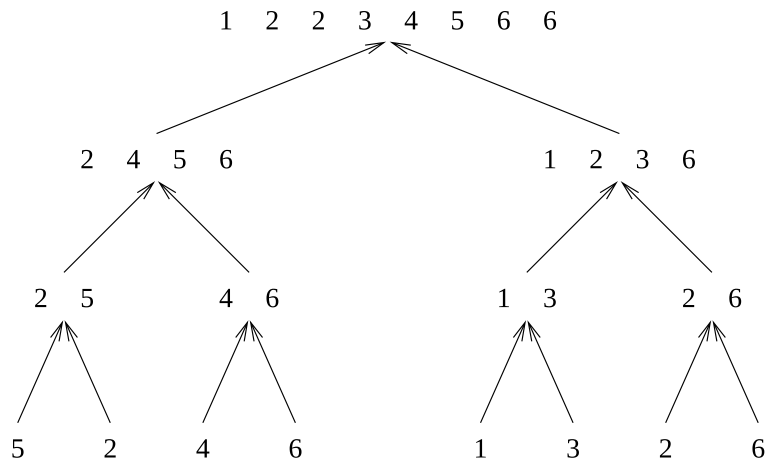
- How does Merge(A, p, q, r) work?
  - Imagine merging two sorted piles of cards. The basic idea is to choose the smallest of the two top cards and put it into the output pile.
  - Running time: $(r - p)$
  - Implementation is a bit messier..

2.1 Mergesort Correctness

- Prove that Merge() is correct (what is the invariant?)
- Assuming that Merge is correct, prove that Mergesort() is correct.
  - Induction on n

2.2 Mergesort Example





2.3 Mergesort Analysis

- To simplify things, let us assume that n is a power of 2, i.e $n = 2^k$ for some k.
- Running time of a recursive algorithm can be analyzed using a **recurrence equation/relation**. Each "divide" step yields two sub-problems of size $n/2$.

$$\begin{aligned} T(n) &\leq c_1 + T(n/2) + T(n/2) + c_2 n \\ &\leq 2T(n/2) + (c_1 + c_2 n) \end{aligned}$$

- Soon we will prove that $T(n) \leq cn \log_2 n$. Intuitively, we can see why the recurrence has solution $n \log_2 n$ by looking at the **recursion tree**: the total number of levels in the recursion tree is $\log_2 n + 1$ and each level costs linear time.
- Note: If $n \neq 2^k$ the recurrence gets more complicated.

  $T(n) = \begin{cases} \Theta(1) & \text{If } n = 1 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(n) & \text{If } n > 1 \end{cases}$

  But we are interested in the order of growth, not in the exact answer. So we first solve the simple version (equivalent to assuming that $n = 2^k$ for some constant k, and leaving out base case and constant in Θ). Once we know the solution for the simple version, one needs solve the original recursion by substitution. This step is necessary for a complete proof, but it is rather mechanical, so we will skip it.
- So even if we are "sloppy" with ceilings and floors, the solution is the same. We usually assume $n = 2^k$ or whatever to avoid complicated cases.

3 Solving recurrences

Methods for solving recurrences:

1. Substitution method.

   With substitution you need to "guess" the result and prove it by induction.

2. Iteration method

   - Recursion-tree method
   - (Master method)

   Iteration is constructive, i.e. you figure out the result; somewhat tedious because of summations.

4 Solving recurrences by induction (the Substitution method)

- Idea: Make a guess for the form of the solution and prove by induction.
- Can be used to prove both upper bounds $O()$ and lower bounds $\Omega()$.
- Let's solve $T(n) = 2T(n/2) + n$ using substitution
  - Guess $T(n) \leq cn \log n$ for some constant c (that is, $T(n) = O(n \log n)$)
  - Proof:
    * Base case: we need to show that our guess holds for some base case (not necessarily $n = 1$, some small n is ok). Ok, since function constant for small constant n.
    * Assume holds for $n/2$: $T(n/2) \leq c\frac{n}{2} \log \frac{n}{2}$ (Question: Why not $n - 1$?)
      Prove that holds for n: $T(n) \leq cn \log n$

$$\begin{aligned}
T(n) &= 2T(n/2) + n \\
&\leq 2(c\frac{n}{2} \log \frac{n}{2}) + n \\
&= cn \log \frac{n}{2} + n \\
&= cn \log n - cn \log 2 + n \\
&= cn \log n - cn + n
\end{aligned}$$

      So ok if $c \geq 1$

- Similarly it can be shown that $T(n) = \Omega(n \log n)$

  Exercise!

- Similarly it can be shown that $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n$ is $\Theta(n \lg n)$.

  Exercise!

- The hard part of the substitution method is often to make a good guess. How do we make a good (i.e. tight) guess??? Unfortunately, there's no "recipe" for this one. Try iteratively $O(n^3), \Omega(n^3), O(n^2), \Omega(n^2)$ and so on. Try solving by iteration to get a feeling of the growth.

5 Solving Recurrences with the Iteration/Recursion-tree Method

- In the iteration method we iteratively "unfold" the recurrence until we "see the pattern".
- The iteration method does not require making a good guess like the substitution method (but it is often more involved than using induction).
- Example: Solve $T(n) = 8T(n/2) + n^2 \quad (T(1) = 1)$

$$\begin{aligned}
T(n) &= n^2 + 8T(n/2) \\
&= n^2 + 8(8T(\frac{n}{2^2}) + (\frac{n}{2})^2) \\
&= n^2 + 8^2T(\frac{n}{2^2}) + 8(\frac{n^2}{4})) \\
&= n^2 + 2n^2 + 8^2T(\frac{n}{2^2}) \\
&= n^2 + 2n^2 + 8^2(8T(\frac{n}{2^3}) + (\frac{n}{2^2})^2) \\
&= n^2 + 2n^2 + 8^3T(\frac{n}{2^3}) + 8^2(\frac{n^2}{4^2})) \\
&= n^2 + 2n^2 + 2^2n^2 + 8^3T(\frac{n}{2^3}) \\
&= \ldots \\
&= n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \ldots
\end{aligned}$$

  - Recursion depth: How long (how many iterations) it takes until the subproblem has constant size? i times where $\frac{n}{2^i} = 1 \Rightarrow i = \log n$
  - What is the last term? $8^iT(1) = 8^{\log n}$

$$\begin{aligned}
T(n) &= n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \ldots + 2^{\log n - 1}n^2 + 8^{\log n} \\
&= \sum_{k=0}^{\log n - 1} 2^k n^2 + 8^{\log n} \\
&= n^2 \sum_{k=0}^{\log n - 1} 2^k + (2^3)^{\log n}
\end{aligned}$$

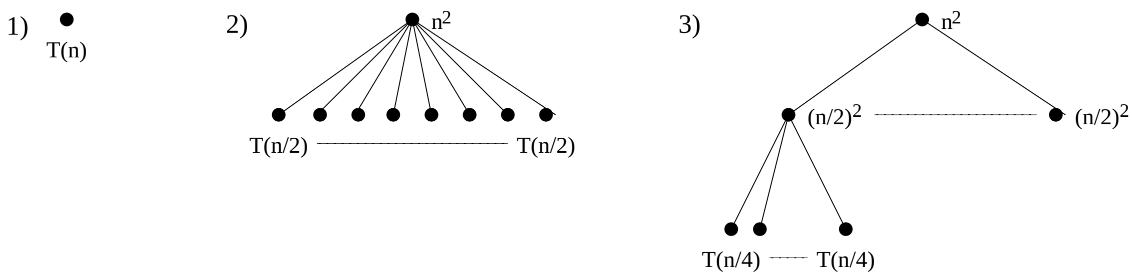
- Now $\sum_{k=0}^{\log n-1} 2^k$ is a geometric sum so we have $\sum_{k=0}^{\log n-1} 2^k = \Theta(2^{\log n - 1}) = \Theta(n)$
- $(2^3)^{\log n} = (2^{\log n})^3 = n^3$

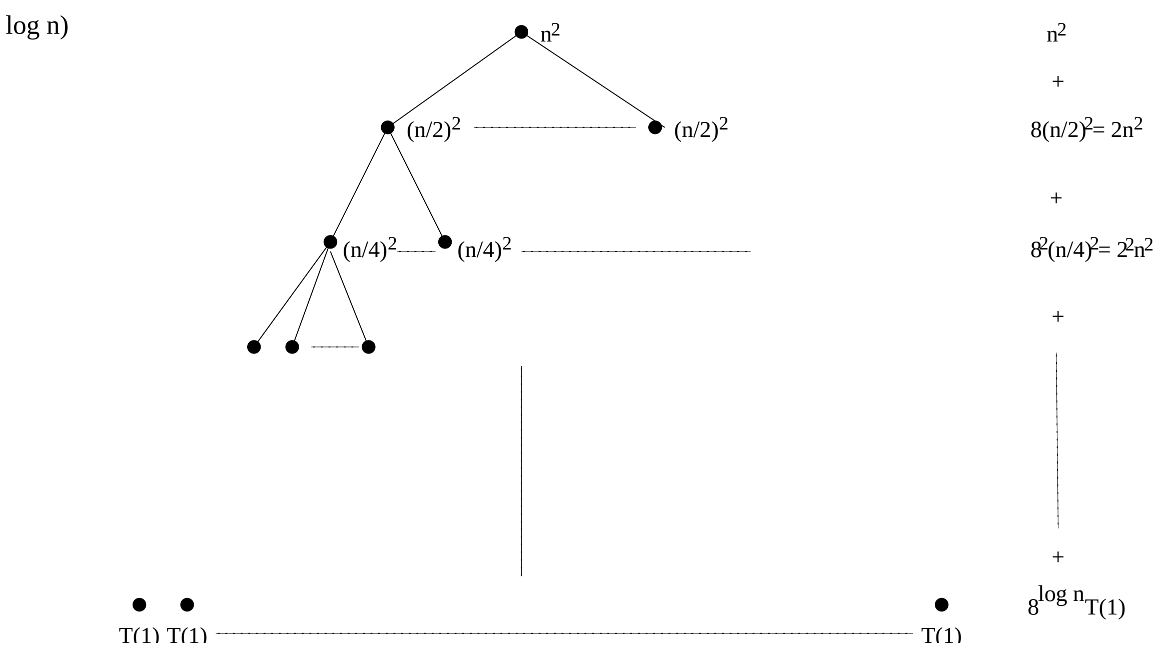
$$\begin{aligned}
T(n) &= n^2 \cdot \Theta(n) + n^3 \\
&= \Theta(n^3)
\end{aligned}$$

5.1 Recursion tree

A different way to look at the iteration method: is the recursion-tree, discussed in the book (4.2).

- we draw out the recursion tree with cost of single call in each node—running time is sum of costs in all nodes
- if you are careful drawing the recursion tree and summing up the costs, the recursion tree is a direct proof for the solution of the recurrence, just like iteration and substitution
- Example: $T(n) = 8T(n/2) + n^2 \quad (T(1) = 1)$





$$T(n) = n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \ldots + 2^{\log n - 1}n^2 + 8^{\log n}$$

6 Master Method

- We have solved several recurrences using *substitution* and *iteration*.
- we solved several recurrences of the form $T(n) = aT(n/b) + n^c \quad (T(1) = 1)$.
  - Merge-sort $\Rightarrow T(n) = 2T(n/2) + n$ ($a = 2, b = 2$, and $c = 1$).
- It would be nice to have a general solution to the recurrence $T(n) = aT(n/b) + n^c$.
- We do!

$$\boxed{\begin{array}{l} T(n) = aT\left(\frac{n}{b}\right) + n^c \quad a \geq 1, b \geq 1, c > 0 \\ \Downarrow \\ T(n) = \begin{cases} \Theta(n^{\log_b a}) & a > b^c \\ \Theta(n^c \log_b n) & a = b^c \\ \Theta(n^c) & a < b^c \end{cases} \end{array}}$$

Proof (Iteration method)

$$\begin{aligned}
T(n) &= aT\left(\tfrac{n}{b}\right) + n^c \\
&= n^c + a\left(\left(\tfrac{n}{b}\right)^c + aT\left(\tfrac{n}{b^2}\right)\right) \\
&= n^c + \left(\tfrac{a}{b^c}\right) n^c + a^2 T\left(\tfrac{n}{b^2}\right) \\
&= n^c + \left(\tfrac{a}{b^c}\right) n^c + a^2\left(\left(\tfrac{n}{b^2}\right)^c + aT\left(\tfrac{n}{b^3}\right)\right) \\
&= n^c + \left(\tfrac{a}{b^c}\right) n^c + \left(\tfrac{a}{b^c}\right)^2 n^c + a^3 T\left(\tfrac{n}{b^3}\right) \\
&= \ldots \\
&= n^c + \left(\tfrac{a}{b^c}\right) n^c + \left(\tfrac{a}{b^c}\right)^2 n^c + \left(\tfrac{a}{b^c}\right)^3 n^c + \left(\tfrac{a}{b^c}\right)^4 n^c + \ldots + \left(\tfrac{a}{b^c}\right)^{\log_b n - 1} n^c + a^{\log_b n} T(1) \\
&= n^c \textstyle\sum_{k=0}^{\log_b n - 1} \left(\tfrac{a}{b^c}\right)^k + a^{\log_b n} \\
&= n^c \textstyle\sum_{k=0}^{\log_b n - 1} \left(\tfrac{a}{b^c}\right)^k + n^{\log_b a}
\end{aligned}$$

Recall geometric sum $\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1} = \Theta(x^n)$

- $\boxed{a < b^c}$

  $a < b^c \Leftrightarrow \frac{a}{b^c} < 1 \Rightarrow \sum_{k=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^k \leq \sum_{k=0}^{+\infty} \left(\frac{a}{b^c}\right)^k = \frac{1}{1-\left(\frac{a}{b^c}\right)} = \Theta(1)$

  $a < b^c \Leftrightarrow \log_b a < \log_b b^c = c$

$$\begin{aligned} T(n) &= n^c \sum_{k=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^k + n^{\log_b a} \\ &= n^c \cdot \Theta(1) + n^{\log_b a} \\ &= \Theta(n^c) \end{aligned}$$

- $\boxed{a = b^c}$

  $a = b^c \Leftrightarrow \frac{a}{b^c} = 1 \Rightarrow \sum_{k=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^k = \sum_{k=0}^{\log_b n-1} 1 = \Theta(\log_b n)$

  $a = b^c \Leftrightarrow \log_b a = \log_b b^c = c$

$$\begin{aligned} T(n) &= \sum_{k=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^k + n^{\log_b a} \\ &= n^c \Theta(\log_b n) + n^{\log_b a} \\ &= \Theta(n^c \log_b n) \end{aligned}$$

- $\boxed{a > b^c}$

  $a > b^c \Leftrightarrow \frac{a}{b^c} > 1 \Rightarrow \sum_{k=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^k = \Theta\left(\left(\frac{a}{b^c}\right)^{\log_b n}\right) = \Theta\left(\frac{a^{\log_b n}}{(b^c)^{\log_b n}}\right) = \Theta\left(\frac{a^{\log_b n}}{n^c}\right)$

$$\begin{aligned} T(n) &= n^c \cdot \Theta\left(\frac{a^{\log_b n}}{n^c}\right) + n^{\log_b a} \\ &= \Theta(n^{\log_b a}) + n^{\log_b a} \\ &= \Theta(n^{\log_b a}) \end{aligned}$$

- Note: Book states and proves the result slightly differently (don't read it).

7 Changing variables

Sometimes reucurrences can be reduced to simpler ones by *changing variables*

- Example: Solve $T(n) = 2T(\sqrt{n}) + \log n$

  Let $m = \log n \Rightarrow 2^m = n \Rightarrow \sqrt{n} = 2^{m/2}$

  $T(n) = 2T(\sqrt{n}) + \log n \Rightarrow T(2^m) = 2T(2^{m/2}) + m$

  Let $S(m) = T(2^m)$

$$\begin{aligned} T(2^m) = 2T(2^{m/2}) + m &\Rightarrow S(m) = 2S(m/2) + m \\ &\Rightarrow S(m) = O(m \log m) \\ &\Rightarrow T(n) = T(2^m) = S(m) = O(m \log m) = O(\log n \log\log n) \end{aligned}$$

8 Other recurrences

Some important/typical bounds on recurrences not covered by master method:

- Logarithmic: $\Theta(\log n)$
  - Recurrence: $T(n) = 1 + T(n/2)$
  - Typical example: Recurse on half the input (and throw half away)
  - Variations: $T(n) = 1 + T(99n/100)$
- Linear: $\Theta(N)$
  - Recurrence: $T(n) = 1 + T(n-1)$
  - Typical example: Single loop
  - Variations: $T(n) = 1 + 2T(n/2), T(n) = n + T(n/2), T(n) = T(n/5) + T(7n/10 + 6) + n$
- Quadratic: $\Theta(n^2)$
  - Recurrence: $T(n) = n + T(n-1)$
  - Typical example: Nested loops
- Exponential: $\Theta(2^n)$
  - Recurrence: $T(n) = 2T(n-1)$