Mergesort and Recurrences

(CLRS 2.3, 4.4)

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

Outline:

1. Introduce the divide-and-conquer algorithm technique.
2. Discuss a sorting algorithm obtained using divide-and-conquer (mergesort).
3. Introduce reccurences as a means to express the running time of recursive algorithms.
4. Discuss iteration (recursion tree) as a way to solve a reccurrence.

1 Divide-and-conquer

Let's say we want to solve a problem P. For e.g. P could be the problem of sorting an array, or finding the smallest element in an array. Divide-and-conquer is an approach that can be applied to any P and goes like this:

Divide-and-Conquer

To Solve P:

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

The simplest way is to divide into *two* subproblems. Can be extended to divide into k subproblems.

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

2 MergeSort

A divide-and-conquer solution for sorting an array gives an algorithm known as mergesort:

- Mergesort:
  - Divide: Divide an array of n elements into two arrays of $n/2$ elements each.
  - Conquer: Sort the two arrays recursively.
  - Combine: Merge the two sorted arrays.
- 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=0 and r=n-1):

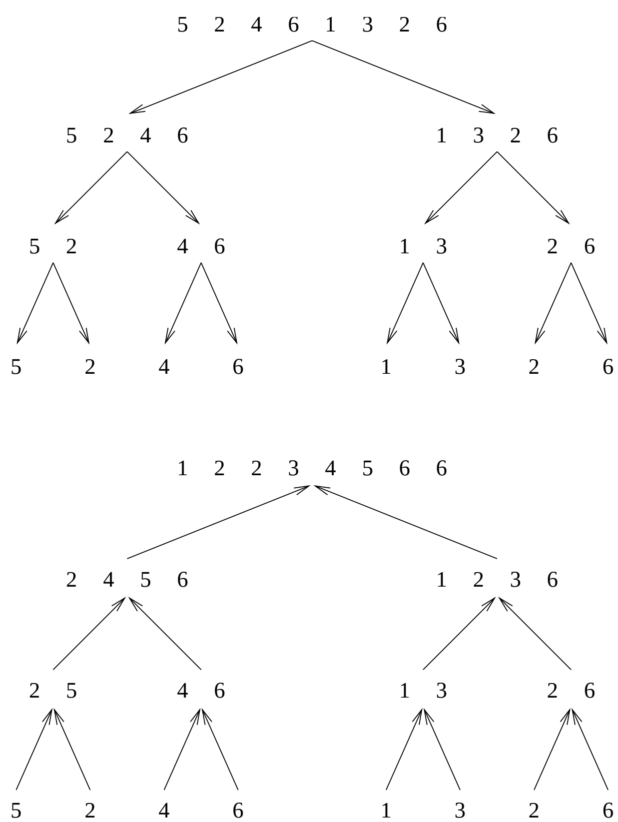
```
Merge Sort(A,p,r)
    If p < r then
        q = ⌊(p + r)/2⌋
        MergeSort(A,p,q)
        MergeSort(A,q+1,r)
        Merge(A,p,q,r)
```

- 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: $\Theta(r - p)$
  - Implementation is a bit messier.

2.1 Mergesort Correctness

- Merge: Why is merge correct? As you look at the next item to put into the merged output, what has to be true?
- Assuming that Merge is correct, prove that Mergesort() is correct.
  - Visualize the recursion tree of mergesort, sorting comes to down to a bunch of merges. If merge works correctly, then mergesort works correctly.
  - Formally, we need 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 relation**. Each "divide" step yields two sub-problems of size $n/2$.
- Let $T(n)$ denote the worst-case running time of mergesort on an array of n elements. We have:

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

- Simplified, $T(n) = 2T(n/2) + \Theta(n)$
- We can see that the recurrence has solution $\Theta(n \log_2 n)$ 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 (more below).
- 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 to solve the original recursion by induction. This step is necessary for a complete proof, but it is rather mechanical, so it is usually skipped.

  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

The steps for solving a recurrence relation are the following:

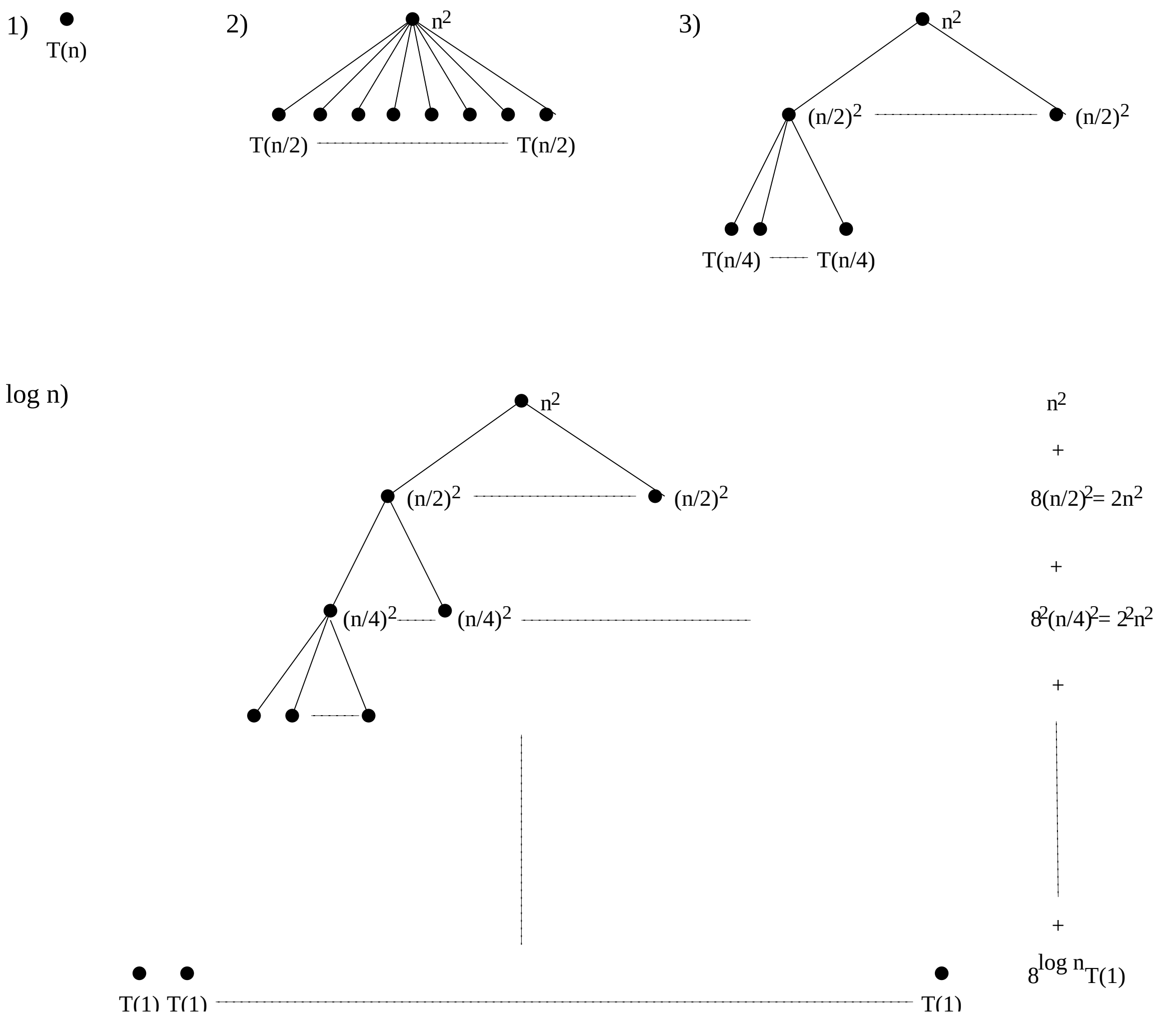
1. Draw the recursion tree to get a feel for how the recursion goes. Sometimes, for easy recurrences, it is sufficient to see the bound. This step can be skipped.
2. Iterate and solve the summations to get the final bound.
3. Use induction to prove this bound formally (substitution method).

In this incarnation of the class we will skip the induction step —- generally speaking this step is pretty mechanical.

For us solving a recurrence will mean finding a theta-bound for $T(n)$ by iteration.

3.1 Solving Recurrences via Recursion tree

- 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 gives you the solution of the recurrence
- Example: $T(n) = 8T(n/2) + n^2 \quad (T(1) = 1)$



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

4 Solving Recurrences by iteration

- Example: Solve $T(n) = 8T(n/2) + n^2$ (with $T(1) = 1$)

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

  - 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 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: $T(n) = 2T(n-1)$

6 Optional

6.1 Master Method

- It is possible to come up with a formula for recurrences of the form $T(n) = aT(n/b) + n^c$ ($T(1) = 1$). This is called the *master method.*
  - Merge-sort $\Rightarrow T(n) = 2T(n/2) + n$ ($a = 2, b = 2$, and $c = 1$).

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

Proof (by iteration)

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

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.

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

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