DIVIDE & CONQUER (D&C)

- from Greek/Latin "divide & rule"
  - divide et impera
  - διαίπτε καὶ βασίλευε

Applications of D&C

- this lecture: median finding, integer multiplication, fast Fibonacci numbers
- recitation: fast matrix multiply
- lecture 3: Fast Fourier Transform
- lecture 4: Applications to data structures

MAIN IDEA

Given problem of size n:

- Divide it into subproblems of size \( \frac{n}{b} \)
- Solve each subproblem recursively (conquer)
- Combine soln’s of subproblems to get solution to the whole problem (rule)
\[ T(n) = T\left(\frac{n}{b}\right) \times \text{(#subproblems)} \]
\[ + \text{(time needed to combine solutions)} \]

Usually analyzed using the Master theorem (reviewed in tomorrow’s recitation) or ad-hoc technique, usually amounting to just unraveling the above recursion.

**APPLICATION 1: MEDIAN FINDING**

- e.g. \( \text{median}([5, 7, -1, -8, 9, 2, 13]) = 5 \)
- Formally: given set \( S \) of \( n \) numbers, define

\[ \text{rank}(x) = \text{how many elements of } S \text{ are } \leq x \]

in example above \( \text{rank}(-1) = 2 \)
upper median \( \hat{\text{A}} \) element of rank \( \left\lceil \frac{n+1}{2} \right\rceil \)
lower median \( \hat{\text{A}} \) element of rank \( \left\lfloor \frac{n+1}{2} \right\rfloor \)

if \( n \) is odd, these are the same

\underline{Algorithmic Challenge} (median finding is a special case)

\textbf{INPUT:} set \( S \) of \( n \) elements, \( i \)

\textbf{GOAL:} find element of rank \( i \)

- naive algorithm:
  - sort the elements of \( S \)
  - return \( i \)-th element

running time \( O(n \cdot \log n) \),
(\text{using e.g. mergesort or heapsort})

- challenge: can we improve this to \( O(n) \)?

- [Blum, Floyd, Pratt, Rivest, Tarjan 1973]:
  \( O(n) \), using D&C
- Idea:

1. Pick some $x \in S$ cleverly

2. Find $B = \{ y \in S \mid y < x \}$
   
   $T = \{ y \in S \mid y > x \}$

   $S = \leftarrow B \rightarrow X \leftarrow T \rightarrow$

   $\text{rank}(x) = |B| + 1$

3. If $\text{rank}(x) = i$, return (lucky)
   
   If $\text{rank}(x) > i$, find element of rank $i$ in subset $B$
   
   If $\text{rank}(x) < i$, find element of rank $i - \text{rank}(x)$ in $T$

*Why do we need to pick $x$ cleverly?*

E.g. execution:

- $S = \{6, 10, 13, 5, 8, 3, 2, 11\}$
- Choose $x = 2$
\[ B = \emptyset \]
\[ T = S \setminus \{2\} \]

- Choose \( x = 3 \)

\[ B = \emptyset \]
\[ T = S \setminus \{2, 3\} \]

\[ \vdots \]

you get the idea:
\[ n \text{ steps, each takes } O(n) \]

runtime \( O(n^2) \)

- **how to pick \( x \) cleverly?**

need to pick \( x \) so \( \text{rank}(x) \) is not extreme

\[ \left\{ \begin{array}{l}
\text{take } O(n) \\
1. \text{Divide the } n \text{ elements into } \left\lfloor \frac{n}{5} \right\rfloor \\
\text{groups of 5 elements (fewer than 5 elements are left ungrouped).}
2. \text{Find the median of each group.}
\end{array} \right. \]
3. Recursively find the median \( x \) of the \( \lceil \frac{n}{5} \rceil \) group medians.

4. Find \( B = \{ y \in S \mid y < x \} \)
   \( T = \{ y \in S \mid y > x \} \)

\[ S = \begin{array}{c}
B \rightarrow x \leftarrow T
\end{array} \]

\[ \text{rank}(x) = |B| + 1 \]

5. If \( \text{rank}(x) = i \), return \( x \)
   If \( \text{rank}(x) > i \), find element of rank \( i \) in subset \( B \)
   If \( \text{rank}(x) < i \), find element of rank \( i - \text{rank}(x) \) in \( T \)

\[ \text{Pictorially} \]

\[ \text{(arrows point to larger values)} \]
Analysis:

- at least \( \left\lfloor \frac{n/5}{2} \right\rfloor = \left\lfloor \frac{n}{10} \right\rfloor \) group medians ≤ \( x \)

⇒ at least \( 3 \left\lfloor \frac{n}{10} \right\rfloor \) elements ≤ \( x \)

- similarly at least \( 3 \left\lfloor \frac{n}{10} \right\rfloor \) elements ≥ \( x \)

- simplification: for \( n > 50 \), \( 3 \left\lfloor \frac{n}{10} \right\rfloor > \frac{n}{4} + 1 \)

⇒ guaranteed \( |B| < \frac{3}{4} n \) & \( |T| < \frac{3}{4} n \)

so 1. steps 1, 2: divide the medians of groups \( \Theta(n) \)

- step 3: recursively find medians \( T(\frac{n}{5}) \)

- step 4: find \( B, T \)

- step 5: recurse on \( B \) or \( T \)

\( T(n) = T(\frac{n}{5}) + T(\frac{3}{4} n) + \Theta(n) \)

Claim: \( T(n) ≤ c \cdot n \), for some constant \( c \).

Proof: Induction: suppose true for \( < n \)

Inductive step:

\( T(n) = T(\frac{n}{5}) + T(\frac{3}{4} n) + \Theta(n) \)

\[ \leq \frac{1}{5} c n + \frac{3}{4} c \cdot n + \Theta(n) \]
\[ \frac{19}{20} cn + \Theta(n) \]

\[ = c \cdot n - \left( \frac{1}{20} cn - \Theta(n) \right) \]

\[ \leq c \cdot n, \text{ if } c \text{ sufficiently large} \]

\[ \text{to kill the constant hiding in } \Theta(c \cdot n). \]

\[ (c \text{ should also be chosen sufficiently large so that } T(n) \leq c \cdot n \text{ holds for base case: } n \leq 50) \]

Why linear time? Because \( \frac{1}{5} + \frac{3}{4} < 1 \)

\[ \Rightarrow \text{substantial problem reduction in every step} \]

\[ \Rightarrow \text{geometric series} \]

\[ \Rightarrow \text{overall runtime is of same order as first step of recursion.} \]
APPLICATION 2: INTEGER MULTIPLICATION

INPUT: two $n$ bit numbers $a, b$
GOAL: compute $a \cdot b$

- grade school algorithm

\[
\begin{array}{c}
  \text{e.g.} \quad 1010 \\
  \times \quad 1101 \\
  \hline
  \text{01010} \\
  \text{00000} \\
  \text{1010} \\
  \hline
  \text{100000010}
\end{array}
\]

Running time $O(n^2)$ (as I need to add $n$ integers of $n$ bits)

- better algorithm?

idea 1: use divide and conquer

OK, subproblems?
view \[ a = 2^{n/2} \cdot X + Y \rightarrow \text{\( n/2 \)-bit numbers} \]

\[ b = 2^{n/2} \cdot Z + W \]

then \[ a \cdot b = (2^{n/2}X + Y) \cdot (2^{n/2}Z + W) \]

\[ = 2^n X \cdot Z + 2^{n/2} (X \cdot W + YZ) + Y \cdot W \]

products of \( n/2 \)-bit numbers

\[ T(n) = 4 \cdot T(n/2) + \Theta(n) \]

\[ = \Theta(n^{\log_2 4}) = \Theta(n^2) \]

(master theorem)

idea 2: Anatolii Karatsuba 1962

- same way to partition \( a, b \) in most significant and least significant part

- compute \( X \cdot Z, Y \cdot W \)

- but \underline{don't} compute \( X \cdot W, YZ \)
- instead compute
  \[(X+Y) \cdot (Z+W) = (X \cdot W + Y \cdot Z) + XZ + YW\]

- since we already have \(X \cdot Z, Y \cdot W\), we get \(X - W + Y \cdot Z\)

- now \(T(n) = 3 \cdot T(\frac{n}{2}) + \Theta(n)\)
  \[= \Theta(n \log^2 3) = \Theta(n^{1.58})\]

Is this best possible?

[Schönhage & Strassen 1971]: \(\Theta(n \log n \log \log n)\)

[Füredi 2007]: \(n \log n \cdot 2\)

---

**APPLICATION 3: POWER OF A MATRIX**

**INPUT**: \(n \times n\) matrix \(A\), integer \(m\)

**Goal**: compute \(A^m\)

**Naive algorithm**: compute

\[A \times A = A^2\]
\[A^2 \times A = A^3\]
\[\vdots\]
\[A^{m-1} \times A = A^m\]
Better algorithm using D&C:

**top-down approach**

\[
A^m = \begin{cases} 
A^{m/2} \times A^{m/2}, & \text{if } m \text{ even} \\
A^{m-1} \times A^{m-1}, & \text{if } m \text{ odd}
\end{cases}
\]

so \( T(n,m) \leq T(n, \frac{m}{2}) + \Theta(n^{2.37}) \)

\[
\leq T(n, \frac{m}{4}) + 2 \cdot \Theta(n^{2.37})
\]

\[
\leq \ldots \leq T(n,2) + \Theta(n^{2.37} \log m)
\]

\[
= \Theta(n^{2.37} \log m)
\]

- so turned an \( m \) into \( \log m \)
- important if \( n \) small, \( m \) big
Application of better Algorithm: Fibonacci numbers

\[ F_0 = 1, \ F_1 = 1, \ F_m = F_{m-1} + F_{m-2} \text{ for } m \geq 2 \]

0, 1, 1, 2, 3, 5, 8, 13, \ldots

\[ F_{30} \text{? Bigger than 1 million} \]

\[ F_m = \frac{1}{\sqrt{5}} \cdot \left( \left( \frac{1 + \sqrt{5}}{2} \right)^m - \left( \frac{1 - \sqrt{5}}{2} \right)^m \right), \text{ for all } m \]

but these are irrational so can't compute \( F_m \) w/ this formula

How compute \( F_m \)?

- Naive algorithm:

\[
F(m) = \begin{cases} 
1 & \text{if } m \leq 1 \\
F(m-1) + F(m-2) & \text{else}
\end{cases}
\]

Running time: \( T(m) = T(m-1) + T(m-2) \)

same recursion as \( F_m \) => \( T(m) = 2^{O(m)} \)
- Dynamic Programming from 6.006

\[ \Theta(m^2) \]

- Even Better?

\[
take \quad A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}
\]

let \[ F(m) = \begin{pmatrix} F_{m+1} & F_m \\ F_m & F_{m-1} \end{pmatrix} \]

**Theorem:** \[ F(m) = A^{m+1} \quad \text{for all } m \geq 1 \]

**Proof:** By induction

- **Base case:** \[ F(1) = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} = A^2 \]

- Assume equality is true for \( m-1 \)

- **Inductive step:**

\[
A^m \times A = \begin{pmatrix} F_m & F_{m-1} \\ F_{m-1} & F_{m-2} \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} F_m + F_{m-1} & F_m \\ F_{m-1} + F_{m-2} & F_{m-1} \end{pmatrix}
\]
\[ \begin{pmatrix} F_{m+1} \\ F_m \end{pmatrix} = \begin{pmatrix} F_m & F_{m-1} \\ F_{m-1} & F_{m-2} \end{pmatrix} \]

\[ = F(m) \]

\[ \Rightarrow \text{Using our fast matrix powering algorithm, computing } A^m = F(m-1) = \begin{pmatrix} F_m & F_{m-1} \\ F_{m-1} & F_{m-2} \end{pmatrix} \]

can be done in time:

\[ \Theta(\log m) \cdot \text{time to multiply } \Theta(\log m) \cdot \text{bit integers} \]

\[ = \Theta(\log m \cdot m^{1.58}) \text{ (with Karatsuba)} \]

\[ = \Theta(\log m \cdot \log\log m \cdot m) \text{ (with Schönhage-Strassen)} \]

important not to forget that our numbers are growing