Algorithmic Paradigms

1. reductions
2. divide and conquer
3. greedy
4. dynamic programming

Reductions

Often, you can use known algorithms to solve new problems. (Don't reinvent the wheel.)

Example: 2-Sum and 3-Sum.

$m = 23$

e.g. $A = \boxed{5 \quad 12 \quad 11 \quad 2 \quad 3 \quad 22 \quad 20}$

2-SUM

Input: array $A[1 \dots n]$ of numbers and target number m

Find: i, j s.t. $A[i] + A[j] = m$ (if they exist)

(we allow $i=j$)

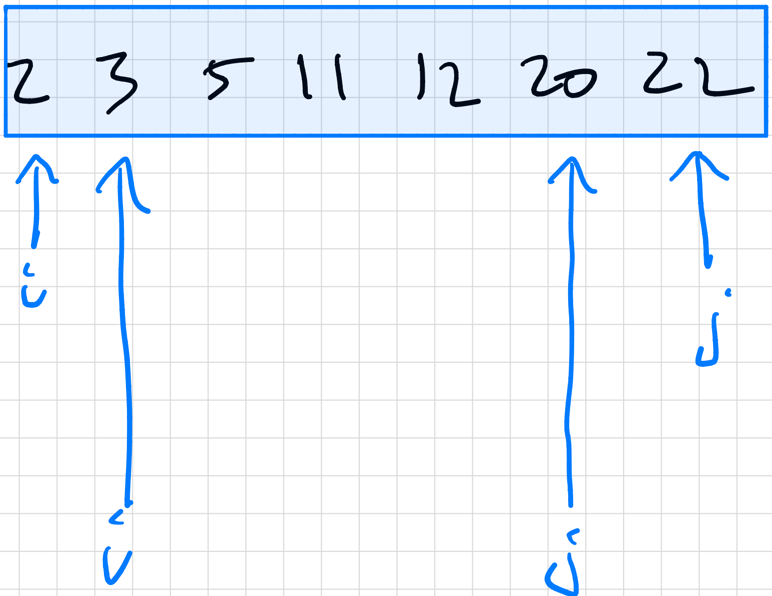
Algorithm 1 **for** $i = 1$ **to** n **do**
  for $j = 1$ **to** n **do**
    if $A[i] + A[j] = m$ SUCCESS
  od
od
FAIL

Algorithm 2 Sort A. For each i binary search for $m - A[i]$

$$\underbrace{O(n \log n)}_{\text{sort}} + \underbrace{O(n \log n)}_{n \text{ binary searches}} \in O(n \log n)$$

Algorithm 3 Improve the 2nd phase

sorted array A target: $m = 23$



$\underline{A[i] + A[j]}$

24 – too big ⇒ decrease j

22 – too small

23 – just right

$i, j := 1, n$
while $i \le j$ **do**
  $S := A[i] + A[j]$
  if $S > m$ **then**
    $j := j - 1$
  elif $S < m$
    $i := i + 1$
  else SUCCESS
od
FAIL

Correctness invariant:
if there is a solution $i^* \le j^*$ then
$i^* \ge i$, $j^* \le j$

Ex. Give more details

Run-time: $O(n)$
(after sorting)

3-SUM

Input: array $A[1\ldots,n]$ of numbers and target number m

Find: i,j,k with $A[i]+A[j]+A[k]=m$

We can reduce 3-SUM to 2-SUM (multiple calls to it)
We want $A[i]+A[j]+A[k]=m$
i.e., $A[i]+A[j]=m-A[k]$

So run 2-SUM with target $m-A[k]$ for each k.
Run-time $O(\underbrace{n}_{\#k\text{'s}}\cdot\underbrace{n\log n}_{\text{2-SUM}})=O(n^2\log n)$

Look more closely:

2-SUM was $O(\underbrace{n\log n}_{\text{sort}})+O(\underbrace{n}_{\text{Algorithm 3}})$

We only need to sort once

This gives $O(n\log n)+O(n^2)=O(n^2)$

Is there a faster algorithms for 3-SUM?
For many years people thought NO, but now there are slightly faster algorithm (2014, 2017).

Divide and Conquer (and solving recurrences)

You've seen (in 1st year & 240) quite a few example of divide and conquer.

divide - break the problem into smaller problems
recurse - solve the smaller subproblems
conquer - combine the solutions to get a sol'n to the whole problem

Examples

- binary search — search in a sorted array for an element e

  - try middle, recurse on first half or second half

  There is only on subproblem and no "conquer" step.
  Let $T(n) =$ max run-time on array of length n.

$$T(n) = 1 + T(n/2)$$

  Actually, $T(n) = 1 + \max\{T(\lfloor \frac{n}{2} \rfloor), T(\lceil \frac{n}{n} \rceil)\}$
  and the solution (as you know) is $T(n) \in O(\log n)$

- sorting

  – mergesort - easy divide, $O(n)$ work to conquer

  – quicksort - $O(n)$ work to divide, easy conquer

  mergesort recurrence

$$\begin{aligned} T(n) &= 2T(n/2) + cn \\ T(n) &\in O(n \log n) \end{aligned}$$

Solving Recurrence Relations

Two basic approaches

- recursion tree method
- guess a solution and prove correct by induction

Recursion tree method for mergesort

if we count comparisons

$$T(n) = 2T(n/2) + cn, \; n \text{ even.} \; T(1) = 0$$

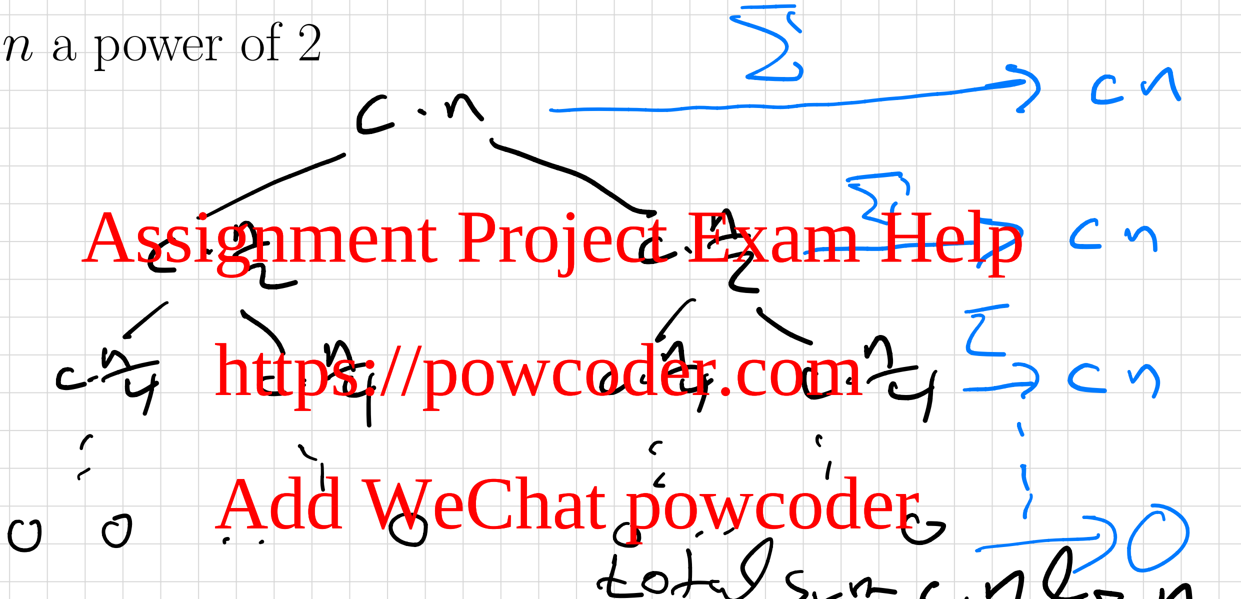
So for n a power of 2

$T(n)$

$T(n/2)$

$T(n/4)$

$T(1)$



CAUTION: Even something this simple gets complicated if we are precise.

\# comparisons

$$T(n) = T(\lfloor \tfrac{n}{2} \rfloor) + T(\lceil \tfrac{n}{2} \rceil) + n - 1, \; T(1) = 0$$

Sol'n: $T(n) = n\lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1$, but not trivial

Luckily we often only want the rate of growth and run-times are usually increasing

e.g., $T(n) \leq T(n')$ $\quad n' =$ smallest power of 2 bigger than n

Note: $n' < 2n$

For mergesort, this gives $T(n) \in O(n \log n)$.

Guess and prove by induction for mergesort recurrence

$$T(n) = T(\lfloor \tfrac{n}{2} \rfloor) + T(\lceil \tfrac{n}{2} \rceil) + n - 1. \; T(1) = 0$$

Prove $T(n) \le cn \log n$ by induction $\forall n \ge 1$.

- Base case: $n = 1$ $\quad T(1) = 0 \qquad cn \log n = 0$ for $n = 1$.
- Assume by induction that $T(n') \le cn' \log n'$ for all $n' < n$, some $n \ge 2$.

Separate into odd and even n — this is one way to be rigorous about floors and ceilings.

n even

$$\begin{aligned} T(n) &= 2T(n/2) + n - 1 \\ &\le 2c \cdot \tfrac{n}{2} \log \tfrac{n}{2} + n - 1 \quad \text{by ind.} \\ &= cn \log \tfrac{n}{2} + n - 1 \\ &= cn(\log n - 1) + n - 1 \\ &= cn \log n - cn + n - 1 \\ &\le cn \log n \quad \text{if } c \ge 1 \end{aligned}$$

<u>n odd</u>

$$
\begin{aligned}
T(n) &= T\left(\frac{n-1}{2}\right) + T\left(\frac{n+1}{2}\right) + n - 1 \\
&\le c\left(\frac{n-1}{2}\right)\log\left(\frac{n-1}{2}\right) + c\left(\frac{n+1}{2}\right)\log\left(\frac{n+1}{2}\right) + n - 1
\end{aligned}
$$

Ind.

use Fact: $\log\left(\frac{n+1}{2}\right) < \log\left(\frac{n}{2}\right) + 1$ $\forall n \ge 3$

$$\le c\left(\frac{n-1}{2}\right)\log\left(\frac{n}{2}\right) + c\left(\frac{n+1}{2}\right)\left(\log\left(\frac{n}{2}\right)+1\right) + n - 1$$

$$\le cn\log n + \left(1 - \frac{c}{2}\right)(n-1)$$

$$\le cn\log n \quad \forall c \ge 2$$

CAUTION: What's wrong with this:

$$T(n) = 2T(n/2) + n$$

Claim: $T(n) \in O(n)$

Proof: Prove $T(n) \leq cn \quad \forall n \geq n_0$
Assume by induction $T(n') \leq cn' \; \forall \; n' < n, n' \geq n_0$.
Then

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\leq 2c(n/2) + n \quad \text{by induction} \\ &= \underbrace{(c+1)}_{\text{growing constant}} n \quad \text{so } T(n) \in O(n) \quad \text{false} \end{aligned}$$

Example — changing the induction hypothesis

$$T(n) = T\left(\lfloor \tfrac{n}{2} \rfloor\right) + T\left(\lceil \tfrac{n}{2} \rceil\right) + 1$$
$$T(1) = 1$$

Guess $T(n) \in O(n)$

Prove by induction $T(n) \leq cn$ for some c

$$T(n) \leq c\left\lfloor \frac{n}{2} \right\rfloor + c\left\lceil \frac{n}{2} \right\rceil + 1 = cn + 1$$

whoops!

So is the guess wrong?

No, e.g., n a power of 2 gives

$$\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + 1 \\
&= 4T\left(\frac{n}{4}\right) + 2 + 1 \\
&\vdots \\
&= 2^k T\left(\frac{n}{2^k}\right) + (2^{k-1} + \cdots + 2 + 1) \qquad n = 2^k \\
&= 2^k + 2^{k-1} + \cdots + 2 + 1 \\
&= 2^{k+1} - 1 \\
&= 2n - 1
\end{aligned}$$

Try to prove by induction $T(n) \leq cn - 1$

$$T(n) \leq c\left\lfloor \frac{n}{2} \right\rfloor - 1 + c\left\lceil \frac{n}{2} \right\rceil - 1 + 1$$

$$= cn - 1$$

Example - changing variables

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \log n$$

Let $m = \log n$ so $n = 2^m$.

$$T(2^m) = 2T(2^{m/2}) + m$$

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

So $S(m/2) = T(2^{m/2})$.

Then $S(m) = 2S(m/2) + m$ Which we know.

$S(m) \in O(m \log m)$ so $T(2^m) = O(m \log m)$

$T(n) = O(\log n(\log\log n))$

We often get recurrences of the form

$$T(n) = aT\left(\frac{n}{b}\right) + cn^k$$

This arises if we divide a problem of size n into a subproblems of size $\frac{n}{b}$ and do cn^k extra work.

e.g., $k = 1$

$a = b = 2$

$T(n) = 2T\left(\frac{n}{2}\right) + cn$ mergesort

$O(n \log n)$

$a = 1 \quad b = 2$

$T(n) = T(n/2) + cn$

$O(n)$

$a = 4 \quad b = 2$

$T(n) = 4T(n/2) + cn$

$O(n^2)$

Theorem ("Master Theorem")

$$T(n) = aT\left(\frac{n}{b}\right) + cn^k$$

$a \geq 1$, $b > 1$, $c > 0$, $k \geq 0$

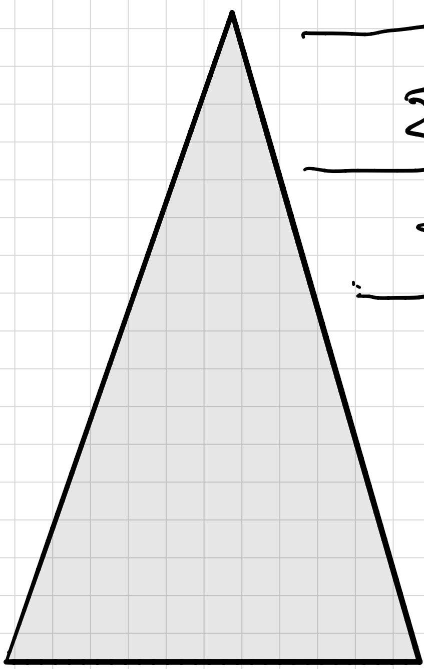
Then

$$T(n) \in \begin{cases} \Theta(n^k) & \text{if } a < b^k \quad \text{i.e., } \log_b a < k \\ \Theta(n^k \log n) & \text{if } a = b^k \\ \Theta(n^{\log_b a}) & \text{if } a > b^k \end{cases}$$

Notes:

- CLRS has a more general version with $f(n)$ in place of cn^k
- you are not responsible for the proof but must know and apply the theorem

A rigorous proof is by induction.
We'll just make senses of it using recursion tree.



$\xrightarrow{\Sigma}$ $cn^k = cn^k\left(\frac{a}{b^k}\right)^0$

$\xrightarrow{\Sigma}$ $ac\left(\frac{n}{b}\right)^k = cn^k\left(\frac{a}{b^k}\right)^1$

$\xrightarrow{\Sigma}$ $a^2c\left(\frac{n}{b^2}\right)^k = cn^k\left(\frac{a}{b^k}\right)^2$

$\vdots$ how far? $\vdots$

$\frac{n}{b^i} = 1 \Rightarrow \boxed{i = \log_b n}$

Intuition for the Master Theorem via recursion tree

$$
\begin{aligned}
T(n) &= aT\left(\frac{n}{b}\right) + cn^k \\
&= a\left[aT\left(\frac{n}{b^2}\right) + c\left(\frac{n}{b}\right)^k\right] + cn^k \\
&= a^2T\left(\frac{n}{b^2}\right) + ac\left(\frac{n}{b}\right)^k + cn^k \\
&= a^3T\left(\frac{n}{b^3}\right) + a^2c\left(\frac{n}{b^2}\right)^k + ac\left(\frac{n}{b}\right)^k + cn^k \\
&\vdots \\
&= a^{\log_b n}T(1) + \sum_{i=0}^{\log_b n - 1} a^i c\left(\frac{n}{b^i}\right)^k \\
&= n^{\log_b a}T(1) + cn^k \sum_{i=0}^{\log_b n - 1}\left(\frac{a}{b^k}\right)^i
\end{aligned}
$$

- If $a < b^k$ (i.e., $\log_b a < k$)

  then $\sum \left(\frac{a}{b^k}\right)^i$ is a geometric series and $\frac{a}{b^k} < 1$

  so $\sum$ is constant and

  $T(n) = n^{\log_b a} T(1) + \Theta(n^k)$

  $T(n) = \Theta(n^k)$

- If $a = b^k$ then

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

So $T(n) = n^{\log_b a} T(1) + cn^k(O(\log n))$

$T(n) = \Theta(n^k \log n)$

- If $a > b^k$ then

  $\sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^k}\right)^i$ is a geometric series with $\frac{a}{b^k} > 1$

so the last term dominates: $\sum_{i=0}^{k-1} x^i = \frac{x^k - 1}{x - 1} \in \Theta(x^k)$ if $x > 1$

$$T(n) = n^{\log_b a}\, T(1) + \underbrace{\Theta\left(n^k \left(\frac{a}{b^k}\right)^{\log_b n}\right)}_{\Theta\left(a^{\log_b n} \frac{n^k}{(b^{\log_b n})^k}\right)}$$

$$(b^{\log_b n})^k = n^k$$

$$= \Theta(a^{\log_b n})$$

$$= \Theta(n^{\log_b a})$$

$$T(n) \in \Theta(n^{\log_b a})$$