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.

2-SUM
Input: array $A[1 \ldots n]$ of numbers and target number $m$
Find: $i, j$ s.t. $A[i] + A[j] = m$ (if they exist)
Algorithm 1
\[
\text{for } i = 1 \text{ to } n \text{ do } \quad \text{for } j = 1 \text{ to } n \text{ do } \quad \text{if } A[i] + A[j] = m \quad \text{SUCCESS}
\]
\[
\text{od} \quad \text{od} \quad \text{od}
\]
FAIL
Algorithm 2 Sort $A$. For each $i$ binary search for $m - A[i]$
\[
O\left(n \log n\right) + O(n \log n) \in O(n \log n)
\]
\[\underbrace{\text{sort}}_{\text{n binary searches}}\]
Algorithm 3 Improve the 2nd phase

Sorted array $A$

| 2 | 3 | 5 | 11 | 12 | 20 | 22 |

Target: $m = 23$

\[
\frac{A[i] + A[j]}{24 - \text{too big} \Rightarrow \text{decrease}}
\]
\[
22 - \text{too small}
\]
\[
23 - \text{just right}
\]

$i, j := 1, n$

while $i \leq j$ do


if $S > m$ then

$j := j - 1$

elseif $S < m$

$i := i + 1$

else SUCCESS

od

FAIL

Correctness invariant:

if there is a solution

\[i^* \leq j^* \text{ then} \]

\[i^* \geq i, j^* \leq j\]

Ex. Give more details

Run-time: $O(n)$

(after sorting)
3-SUM

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


We can reduce 3-SUM to 2-SUM (multiple calls to it)

So run 2-SUM with target $m - A[k]$ for each $k$.
Run-time $O(n \cdot n \log n) = O(n^2 \log n)$

Look more closely:

2-SUM was $O(n \log n) + O(n)$

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, \( T(n) = 1 + \max\{T(\lfloor n/2 \rfloor), T(\lceil n/n \rceil)\} \)
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

\[
T(n) = 2T(n/2) + cn
\]

\( T(n) \in O(n \log n) \)
Solving Recurrence Relations

Two basic approaches

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

Recursion tree method for mergesort

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

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.

\[ T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n - 1, \text{ } T(1) = 0 \]

Sol’n: \( T(n) = n \lfloor \log n \rfloor - 2^\lceil \log n \rceil + 1 \), but not trivial

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

\[ \text{e.g., } T(n) \leq T(n') \quad n' = \text{smallest power of 2 bigger than } n \]

Note: \( n' < 2n \)

For mergesort, this gives \( T(n) \in O(n \log n) \).
Guess and prove by induction for mergesort recurrence

\[ T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + n - 1. \quad T(1) = 0 \]

Prove \( T(n) \leq cn \log n \) by induction \( \forall n \geq 1 \).

- **Base case:** \( n = 1 \quad T(1) = 0 \quad cn \log n = 0 \) for \( n = 1 \).
- **Assume by induction that** \( T(n') \leq cn' \log n' \) for all \( n' < n \), some \( n \geq 2 \).

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

\( n \) even

\[ T(n) = 2T(n/2) + n - 1 \]

\[ \leq 2c \cdot \frac{n}{2} \log \frac{n}{2} + n - 1 \quad \text{by ind.} \]

\[ = cn \log \frac{n}{2} + n - 1 \]

\[ = cn(\log n - 1) + n - 1 \]

\[ = cn \log n - cn + n - 1 \]

\[ \leq cn \log n \quad \text{if} \quad c \geq 1 \]
\( n \text{ odd} \)

\[
T(n) = T\left(\frac{n-1}{2}\right) + T\left(\frac{n+1}{2}\right) + n - 1
\]

\[
\leq c \left(\frac{n-1}{2}\right) \log \left(\frac{n-1}{2}\right) + c \left(\frac{n+1}{2}\right) \log \left(\frac{n+1}{2}\right) + n - 1
\]

Ind.

Use Fact: \( \log\left(\frac{n+1}{2}\right) < \log^{-1} + 1 \)

\[
\forall n \geq 3
\]

\[
\leq c \left(\frac{n-1}{2}\right) \log \left(\frac{n}{2}\right) + c \left(\frac{n+1}{2}\right) \left(\log \left(\frac{n}{2}\right) + 1\right) + n - 1
\]

\[
\leq c n \log n + (1 - \frac{c}{2})(n-1)
\]

\[
\leq c n \log n \quad \forall \quad c \geq 2
\]
CAUTION: What’s wrong with this:

\[ T(n) = 2T(n/2) + n \]

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

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

\[
T(n) = 2T(n/2) + n \\
\leq 2c(n/2) + n \quad \text{by induction} \\
= (c + 1)n \quad \text{so } T(n) \in O(n) \quad \text{false}
\]

growing constant
Example — changing the induction hypothesis

\[ T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + 1 \]
\[ T(1) = 1 \]

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

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

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

So is the guess wrong?

No, e.g., \( n \) a power of 2 gives

\[ T(n) = 2T\left(\frac{n}{2}\right) + 1 = 4T\left(\frac{n}{4}\right) + 2 + 1 \]
\[ \vdots \]
\[ = 2^k T\left(\frac{n}{2^k}\right) + \left(2^{k-1} + \cdots + 2 + 1\right) \quad n = 2^k \]
\[ = 2^k + 2^{k-1} + \cdots + 2 + 1 \]
\[ = 2^{k+1} - 1 \]
\[ = 2n - 1 \]

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., \( k = 1 \)

\[ a = b = 2 \]

\[ T(n) = 2T\left(\frac{n}{2}\right) + cn \quad \text{mergesort} \]

\[ O(n \log n) \]

\[ a = 1 \quad b = 2 \]

\[ T(n) = T(n/2) + cn \]

\[ O(n) \]

\[ a = 4 \quad b = 2 \]

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

\[ O(n^2) \]
Theorem ("Master Theorem")

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

\( a \geq 1, \ b > 1, \ c > 0, \ k \geq 0 \)

Then

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

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.

\[
\frac{h_i}{b^i} = 1 \implies \quad i = \log_b n
\]
Intuition for the Master Theorem via recursion tree

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

\[
= a \left[ aT\left(\frac{n}{b^2}\right) + c \left(\frac{n}{b}\right)^k \right] + cn^k
\]

\[
= a^2T\left(\frac{n}{b^2}\right) + ac \left(\frac{n}{b}\right)^k + cn^k
\]

\[
= a^3T\left(\frac{n}{b^3}\right) + a^2c \left(\frac{n}{b^2}\right)^k + ac \left(\frac{n}{b}\right)^k + cn^k
\]

\[
= \cdots
\]

\[
= a^{\log_b n}T(1) + \sum_{i=0}^{\log_b n - 1} a^i c \left(\frac{n}{b^i}\right)^k
\]

\[
= n^{\log_b a}T(1) + cn^k \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^k}\right)^i
\]
• If $a < b^k$ (i.e., $\log_b a < k$) then $\sum_i \left(\frac{a}{b^k}\right)^i$ is a geometric series and $\frac{a}{b^k} < 1$ so $\sum_i$ 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 $x > 1$

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

$$= \Theta(\log_b n \left( \frac{n^k}{(b \log n)^k} \right) = n^k$$

$$T(n) \leq \Theta \left( n^{\log_b a} \right)$$