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: 2SUM and 3SUM.

Problem 2SUM

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

Find: $i, j$ such that $A[i] + A[j] = m$ (if they exist)

Example: $A = [5 \ 12 \ 11 \ 2 \ 3 \ 22 \ 20]$ and $m = 23$

Algorithm 1
for $i = 1$ to $n$
do
    for $j = 1$ to $n$
do
        if $A[i] + A[j] = m$ return $(i, j)$
return fail
Run-time: $O(n^2)$

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

$O(n \log n) + n \times O(\log n) \in O(n \log n)$
Algorithm 3: Improve the 2nd phase.

\[
A = [2, 3, 5, 11, 12, 20, 22]
\]

Target \( m = 23 \)

**sorted array**

\[
\]

Exercise: Prove correctness.

Run-time: \( O(n) \) (after sorting)
Problem 3SUM

\( \text{Input: array } A[1 \ldots, n] \text{ of numbers and target number } m \)

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


So run 2SUM with target \( m - A[k] \) for each \( k \).

Run-time: \( O(\frac{n \cdot n \log n}{\# \text{ k's}}) = O(n^2 \log n) \)
\( \text{2SUM} \)

Look more closely: 2SUM was \( O(n \log n) + O(n) \)
\( \text{sort} + \text{alg. 3} \)

We only need to sort once.

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

---

Is there a faster algorithms for 3SUM? 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 solution 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 \( T(n) = \text{max run-time on array of length } n \)
  - then \( T(n) = 1 + T(n/2) \)
  - actually, \( T(n) = 1 + \max \{ T(\lfloor \frac{n}{2} \rfloor), T(\lceil \frac{n}{2} \rceil) \} \)
  - 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 is \( T(n) = 2T(n/2) + cn \)
  - solution is \( T(n) \in O(n \log n) \)
Solving Recurrence Relations

Two basic approaches

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

Recursion tree method for mergesort

Let $T(n)$ be the run-time of mergesort. If $n$ is even then

$$T(n) = \begin{cases} 
2T(n/2) + cn & n > 1 \\
0 & n = 1
\end{cases}$$

Note: $T(1) = 0$ if we are only counting comparisons. (Otherwise $T(1) = c$.)

So for $n$ a power of 2 we have

Total sum $c \cdot n \log_2 n$
CAUTION: Even something this simple gets complicated if we are precise.

Exact recurrence for mergesort for any $n$ is

$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n - 1 & n > 1 \\ 0 & n = 1 \end{cases}$$

The exact closed form solutions is

$$T(n) = n \lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1.$$

Note: this is not trivial to derive! We have only stated the result here.

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

For mergesort, the result on the previous slide gives $T(n') \leq n' \log n'$.

We conclude that

$$T(n) \leq T(n') \leq n' \log n' \leq (2n) \log(2n) \in \Theta(n \log n).$$
Guess and prove by induction for mergesort recurrence

\[
T(n) = \begin{cases} 
T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n - 1 & n > 1 \\
0 & n = 1 
\end{cases}
\]

We will prove that there exists a constant \(c\) such that \(T(n) \leq cn \log n\) for all \(n \geq 1\).

- Base case: \(n = 1\) \(T(1) = 0\) \(cn \log n = 0\) 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

\[
T(n) = 2T(n/2) + n - 1 \\
\leq 2c(n/2) \log(n/2) + n - 1 \quad (\text{by induction hypothesis}) \\
= cn \log(n/2) + n - 1 \\
= cn(\log n - 1) + n - 1 \\
= cn \log n - cn + n - 1 \\
\leq cn \log n \text{ if } c \geq 1
\]
\( n \) odd

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

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

\[
\leq c \left(\frac{n-1}{2}\right) \log \left(\frac{n-1}{2}\right) + c \left(\frac{n+1}{2}\right) \log \left(\frac{n+1}{2}\right) + n - 1 \quad \text{(ind. hyp.)}
\]

now use fact that \( \log \left(\frac{n+1}{2}\right) < \log \left(\frac{n}{2}\right) + 1 \) for all \( n \geq 3 \)

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

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

\[
\leq cn \log n \quad \text{for all} \quad c \geq 2
\]
Lemma: We have
\[ \log \left( \frac{n + 1}{2} \right) < \log \left( \frac{n}{2} \right) + 1 \] (1)
for all \( n \geq 3 \).

Proof: Let
\[ P(n) := \log \left( \frac{n}{2} \right) + 1 - \log \left( \frac{n + 1}{2} \right) \]
be the right hand side minus the left hand side of (1).

Direct numerical evaluation shows that \( P(3) \approx 0.58496 \).

Now note that the derivative of \( P(n) \) with respect to \( n \) is
\[ \frac{1}{n(n + 1) \ln(2)}. \]

It follows that \( P(n) > P(3) \) for all \( n > 3 \).
Maple 2020 (APPLE UNIVERSAL OSX)

Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2020
All rights reserved. Maple is a trademark of Waterloo Maple Inc.
Type ? for help.

> P := log[2](n/2) + 1 - log[2]((n+1)/2);

\[
P := \frac{\ln(n/2)}{\ln(2)} + \frac{\ln(n/2 + 1/2)}{\ln(2)}
\]

# Check that P evaluated at n = 3 is positive.
#
> evalf(subs(n=3,P));

0.5849625007

# Check that the derivative of P w.r.t. n is positive for n >= 3.
#
> diff(P,n);

\[
\frac{1}{n \ln(2)} - \frac{1/2}{(n/2 + 1/2) \ln(2)}
\]

> simplify(%,'symbolic');

\[
\frac{1}{n \ln(2) (n + 1)}
\]

> quit
memory used=4.2MB, alloc=40.3MB, time=0.09
CAUTION: What’s wrong with this?

\[ T(n) = \begin{cases} 
  T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + n & \text{if } n > 1 \\
  0 & \text{if } n = 1
\end{cases} \]

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

Proof:
We will prove that there exists a constant \( c \) such that \( T(n) \leq cn \) for all \( n \geq 1 \).

Base case: \( n = 1 \). \( T(1) = 0 \leq c \) for any constant \( c \geq 0 \).

Assume by induction that \( T(n') \leq cn' \) for all \( n' < n \), some \( n \geq 2 \).

Then

\[
T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + n \\
\leq c \left\lfloor \frac{n}{2} \right\rfloor + c \left\lceil \frac{n}{2} \right\rceil + n \quad \text{(by induction)} \\
= (c + 1)n \quad \text{so } T(n) \in O(n)
\]
CAUTION: What’s wrong with this?

\[ T(n) = \begin{cases} 
T \left( \lfloor \frac{n}{2} \rfloor \right) + T \left( \lceil \frac{n}{2} \rceil \right) + n & \text{if } n > 1 \\
0 & \text{if } n = 1
\end{cases} \]

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

Proof:

We will prove that there exists a constant \( c \) such that \( T(n) \leq cn \) for all \( n \geq 1 \).

Base case: \( n = 1 \). \( T(1) = 0 \leq c \) for any constant \( c \geq 0 \).

Assume by induction that \( T(n') \leq cn' \) for all \( n' < n \), some \( n \geq 2 \).

Then

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

so \( T(n) \not\in O(n) \) false
Example — changing the induction hypothesis

\[ T(n) = \begin{cases} 
T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + 1 & \text{if } n > 1 \\
1 & \text{if } n = 1 
\end{cases} \]

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., \( n = 2^k \) a power gives

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

Prove by induction \( T(n) \leq cn - 1 \).

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

How to complete the proof?

Base case: \( n = 1 \). Any \( c \geq 2 \) works.

Assume by induction that

\[ T(n') \leq cn' - 1 \]

for all \( n' < n \), some \( n \geq 2 \).

Do inductive step above.
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.

Examples when \( b = 2 \) and \( k = 1 \).

- \( a = 1 \)
  - \( T(n) = T\left(\frac{n}{2}\right) + cn \)
  - \( T(n) \in O(n) \)

- \( a = 2 \)
  - \( T(n) = 2T\left(\frac{n}{2}\right) + cn \) (mergesort)
  - \( T(n) \in O(n \log n) \)

- \( a = 4 \)
  - \( T(n) = 4T\left(\frac{n}{2}\right) + cn \)
  - \( O(n^2) \)
**Theorem ("Master Theorem")**

Let \( T(n) = aT\left(\frac{n}{b}\right) + cn^k \) where \( a \geq 1, b > 1, c > 0, k \geq 0 \). Then

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

Note: floors and ceilings are allowed for the argument of \( T\left(\frac{n}{b}\right) \).

**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 the recursion tree.

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

\[
= n^{\log_b a}T(1) + cn^k \sum_{i=0}^{\log_b n-1} \left(\frac{a}{b^k}\right)^i
\]
By considering the recursion tree we have established that

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

- If \( a < b^k \) (that is, \( \log_b a < k \)) then \( \sum \left( \frac{a}{b^k} \right)^i \) is a geometric series and \( \frac{a}{b^k} < 1 \). So \( \sum \) is constant and \( T(n) = n^{\log_b a} T(1) + \Theta(n^k) \in \Theta(n^k) \).

- 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 \( T(n) = n^{\log_b a} T(1) + cn^k(\Theta(\log n)) \in \Theta(n^k \log n) \).

- 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: (Recall that \( \sum_{i=0}^{u-1} x^i = \frac{x^u - 1}{x - 1} \in \Theta(x^u) \) if \( x > 1 \).)

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