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 $A[i..n]$ of numbers and target number $m$
Find $i, j$ s.t. $A[i] + A[j] = m$ (if they exist)
Algorithm 1
for $i = 1..n$
  for $j = i..n$
    if $A[i] + A[j] = m$ SUCCESS
    end
  end
end
FAIL
run time $O(n^2)$

Algorithm 2
Sort $A$.
for each $i$ do binary search for $m-A[i]$
$O(n \log n) + O(n \log n) = O(n \log n)$

n binary searches
Algorithm 3  
**Improve the 2nd phase**

| 3  2 | 1  2 | 2  2 |

- Sorted array $A$
- $i$, $j$, $j-1$
- If $S > m$
  - $j = j - 1$
- Else if $S < m$
  - $i = i + 1$
- Else SUCCESS
- END

**3-Sum**

**Input**: array $A[i..n]$ of numbers and target number $m$.


We will stick to $m = 0$ (allow $i = j$ etc.)

**Correctness invariant**: if there is a solution

- $i^* \leq j^*$
- $i^* \geq i$, $j^* \leq j$

**Example**: Give more details

**Run time**: $O(n)$ (after sorting)
We can reduce 3-SUM to 2-SUM (multiple copies of)
So run 2-SUM with target \(-A[k]\) for each \(k\).
Run-time \( O(n \cdot \log n) = \frac{1}{2} O(n^2 \log n) \)
\(#k's\) 2-SUM

Look more closely:

2-SUM was \( O(n \log n) + O(n) \)
\(\underbrace{\text{sort}}_{\text{Algorithm 2}}\)
We only need to sort once
This gives \( O(n \log n) + O(n^2) = O(n^2) \).

Ex. Solve 3-SUM for general target \(m\)
- modify algorithm
- or (cute reduction): \( A'[i] \leftarrow A[i] - m/3 \)
Solve 3-SUM with target 0 in \( A' \).

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

You've seen (in 1st year CS 240) quite a few examples 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 one subproblem and no "conquer" step.

  Let \( T(n) = \max \) runtime on array of length \( n \)

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

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

  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**

\[ T(n) = 2T\left(\frac{n}{2}\right) + c \cdot n \]

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

Two basic approaches

- recursion tree method.

- guess a solution and prove correct by induction.

Recursion tree method for mergesort recurrence.

\[ T(n) = 2T\left(\frac{n}{2}\right) + c \cdot n \quad n \text{ even} \]

\[ T(1) = C \quad (\text{corrected from class}) \]

So for \( n \) a power of 2

\[ T(n) \]

\[ T\left(\frac{n}{2}\right) \quad c \cdot \frac{n}{2} \]

\[ T\left(\frac{n}{4}\right) \quad c \cdot \frac{n}{4} \]

\[ \vdots \]

\[ c \ldots \ldots c \]

\[ \text{Total Sum } c \cdot \log_2 n + cn \]

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

\[ T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + (n-1) \quad T(1) = 0 \]

Solve: \( T(n) = n \log_2 n - 2 \log_2 n + 1 \) but not trivial.

Luckily we often only want the rate of growth and runtimes are usually increasing.

E.g., \( T(n) \leq T(n') \quad n' \text{ is smallest power of 2 bigger than } n \).

Note: \( n' \leq 2n \)

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

- Prove $T(n) \leq c \cdot n \log n$ by induction for $n \geq 2$ for $(*)$.
- Separating into odd and even $n$ - this is one way to be rigorous about floor and ceiling.

**Basis.** $n=2$: $T(2) = 2T(1)+1 = 2c$ for $n=2$.

So $T(n) \leq c \cdot n \log n$ for $n \geq 2$ if $c \geq \frac{1}{2}$.

- Basis of $n=1$ would suffice: $T(1)=0$; $c \cdot n \log n = 0$.

**Induction step.**

$n$ even: $T(n) = 2T(n/2) + n-1$

- By induction,
  - $2c \cdot \frac{n}{2} \log \frac{n}{2} + n-1$
  - $= c \cdot n \log \frac{n}{2} + n-1 = c \cdot n(\log n - 1) + n-1$
  - $= c \cdot n \log n - c \cdot n + n-1 \leq c \cdot n \log n$ if $c \geq 1$.

$n$ odd: $T(n) = T(n-1) + T(n+1) + n-1$

- By induction,
  - $c \cdot (n-1) \log \frac{n-1}{2} + c \cdot (n+1) \log \frac{n+1}{2} + n-1$

---

**Caution.** What's wrong with this:

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

*Claim??* $T(n) \in \Theta(n)$

*Proof.*

- Prove $T(n) \leq c \cdot n \quad \forall \, n \geq n_0$.
- Assume by induction $T(n') \leq c \cdot n' \quad \forall \, n' < n, \, n' \geq n_0$.
- Then $T(n) = 2T(n/2) + n$
  - $\leq 2 \cdot c \cdot \frac{n}{2} + n$ by induction
  - $(c+1) \cdot n$

So $T(n) \in \Theta(n)$ - false.

"Constant" growing constant.
Example

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

\[ T(1) = 1 \]

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

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

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

So is the guess wrong?

No, e.g. \( n = \text{a power of 2} \) gives

\[ T(n) = 2T\left(\frac{n}{2}\right) + 1 = 4T\left(\frac{n}{4}\right) + 3 + 1 = \ldots \]

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

\[ = 2^k + 2^{k-1} + \ldots + 2 + 1 = 2^{k+1} - 1 = 2n - 1 \]

Try to prove by induction \( T(n) \leq c \cdot n - 1 \)

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

So, curiously, we make the induction work by lowering the bound.