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 $A[1 \cdots n]$ 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..n
    for j = i..n
        if A[i] + A[j] = m  SUCCESS
    end
end
FAIL
```

run time $O(n^2)$

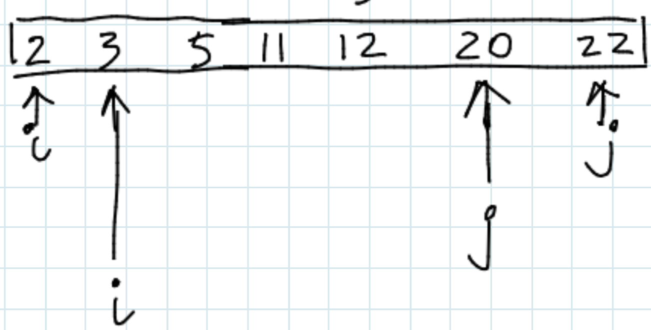
Algorithm 2 Sort A.

for each i do binary search for $m - A[i]$

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

Algorithm 3 Improve the 2nd phase

sorted array A



target: $m = 23$

$A[i]+A[j]$
24 – too big. Decrease j.
22 – too small. Increase i.
23 – just right!

```
i ← 1; j ← n
while i ≤ j
    S ← A[i] + A[j]
    if S > m
        j ← j-1
    else if S < m
        i ← i+1
    else SUCCESS
end
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

<u>input</u> array $A[1..n]$ of numbers and target number m.

Find i, j, k with $A[i]+A[j]+A[k] = m$. (allow $i=j$ etc.)

We will stick to $m = 0$

We can reduce 3-SUM to 2-SUM (multiple copies of)

we want $A[i] + A[j] + A[k] = 0$

i.e. $A[i] + A[j] = -A[k]$

So run 2-SUM with target $-A[k]$ 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 $\underbrace{O(n \log n)}_{\text{sort}} + \underbrace{O(n)}_{\text{Algorithm 2}}$

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 & 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 soln to 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 run time on array of length n

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

  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

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

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

Solving Recurrence Relations

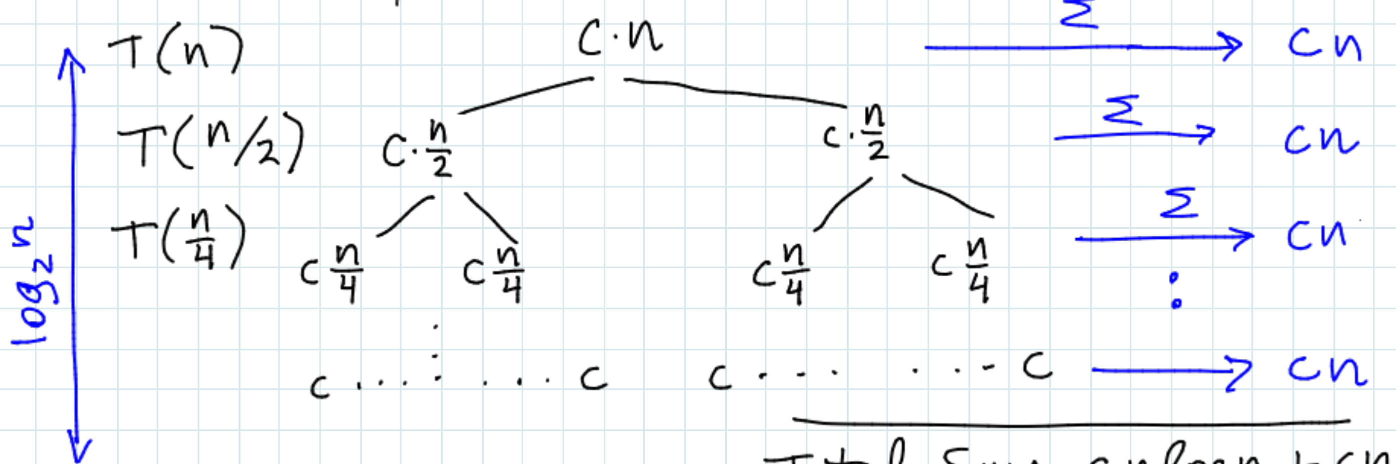
Two basic approaches

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

Recursion tree method for mergesort recurrence.

$T(n) = 2T(\frac{n}{2}) + c \cdot n$, n even. $T(1) = c$ (corrected from class)

So for n a power of 2



Total Sum $c \cdot n \log n + cn$

CAUTION Even something this simple gets complicated if we are precise

\# comparisons

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

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

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

e.g. $T(n) \leq T(n')$ n' = 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 $T(n) \le c \cdot n \log n$ by induction $\forall n \ge 2$ for (*)

separating into odd and even n — this is one way to be rigorous about floor and ceiling.

basis. $n=2$ $T(2) = 2T(1)+1 = 1$ $c \cdot n \log n = 2c$ for $n=2$

so $T(n) \le c \cdot n \log n$ for $n=2$ if $c \ge \frac{1}{2}$

basis of $n=1$ would suffice $T(1)=0$ $c \cdot n \log n = 0$

induction step

n even $T(n) = 2T(\frac{n}{2}) + n - 1$

$$\underset{\text{by induction}}{\le} 2c\frac{n}{2}\log\frac{n}{2} + n - 1$$

$$= cn\log\frac{n}{2} + n - 1 = c \cdot n(\log n - 1) + n - 1$$

$$= c \cdot n \log n - c \cdot n + n - 1 \le c \cdot n \log n \text{ if } c \ge 1.$$

n odd $T(n) = T(\frac{n-1}{2}) + T(\frac{n+1}{2}) + n - 1$

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

...

CAUTION. What's wrong with this:

$$T(n) = 2T(\tfrac{n}{2}) + n$$

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

Pf Prove $T(n) \le c \cdot n$ $\forall n \ge n_0$

Assume by induction $T(n') \le c \cdot n'$ $\forall n' < n$, $n' \ge n_0$

Then $T(n) = 2T(\frac{n}{2}) + n$

$$\le 2 \cdot c \cdot \frac{n}{2} + n \quad \text{by induction}$$

$$= \underbrace{(c+1)}_{\text{constant}} n \qquad \text{so } T(n) \in O(n) \text{ — false}$$

— growing constant

Example

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

$$T(1) = 1$$

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

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

$$T(n) \le c \lfloor \tfrac{n}{2} \rfloor + c \lceil \tfrac{n}{2} \rceil + 1 = cn + 1 \quad \text{whoops!}$$

So is the guess wrong?

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

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

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

$$T(n) \le c \cdot \lfloor \tfrac{n}{2} \rfloor - 1 + c \cdot \lceil \tfrac{n}{2} \rceil - 1 + 1 = c \cdot n - 1$$

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