L1

Very small changes in the problem statement can have very different complexity

Recall P class of problems solvable in polynomial time

$O(n^k)$ for some constant k

NP class of problem whose solution is verifiable in poly time

eg Hamiltonian cycle. checking whether a cycle is hamiltonian is poly time. Determining whether a graph has a hamiltonian cycle is a hard problem

NP_complete (Defines the level of intractability for NP)

They are the hardest problems

If you can solve one NP_complete problem in poly time, you can solve all problems in NP, in poly time

Interval scheduling

single resource

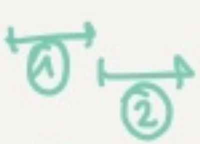
Resources and requests 1... n

S(i) start time $s(i) < f(i)$

F(i) finish time

two requests are compatible if they don't overlap $F(i) \leq s(j)$

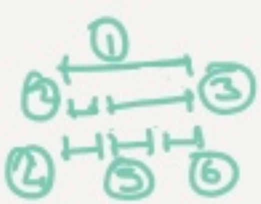
$f(j) \leq s(i)$



COMPATIBLE

Select compatible subset of requests, that is of maximum size

max size!



Claim we can solve this problem by using a greedy algorithm

Definition. A greedy algorithm is a myopic algorithm that

- processes the input one piece at a time but with no apparent lookahead

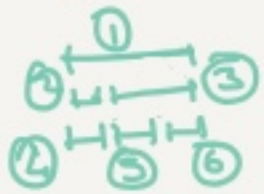
Greedy interval scheduling

1. Use a simple rule to select the request
2. Reject all requests that are incompatible with i
3. Repeat until all requests are processed

problem got smaller

common template for a greedy algo

Possible rules?



Going through in numerical order ✗

Take the one with the shortest time ✗

- for each request, find # of incompatible requests. Select the one with min # of incompatibles ✗

- pick earliest finish time ✓

Claim. Greedy algorithm outputs a list of intervals

$(\langle s(i_1), f(i_1)\rangle \dots \langle s(i_k), f(i_k)\rangle)$

such that $s(i_1) < f(i_1) \leq s(i_2) < f(i_2) \dots \leq s(i_k) < f(i_k)$

Proof If $f(i_j) > s(i_{j+1})$ interval j+1 intersect. contradict step of algorithm

Weighted interval scheduling

Each request has weight wi.

Schedule a subset of requests with maximum weights

Greedy not working here

counter example

Dynamic programming

- Figure out what the subproblems are

$R^x = \{request_j \in R \mid s(j) \geq x\}$

$x = f(i)$ $R^{f(i)}$ requests later $F(i)$

n number of requests # subproblems = n

Solve subproblem once and memoize

subproblems * time to solve each subproblem

$opt(R) = \max_{1 \leq i \leq n} (w_i + opt(R^{f(i)}))$

L2 Divide and conquer

Paradigm

Given a problem of size n, divide it into "a" subproblems of size n/b

$a \geq 1$, $b > 1$

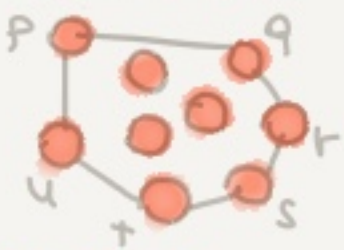
Solve each subproblem recursively

Combine solutions into the overall solution

↑ Not in all divide and conquer algorithms

$T(n) = aT(\frac{n}{b}) + $ [work for combining]

Convex Hull



Given points on a plane

$S = \{(x_i, y_i) \quad i = 1, 2, \dots n\}$

assume not have the same x coordinates, and no three in a line

Convex Hull is the smallest polygon containing all points in S

CH(S)

Sequence of points on the boundary in clockwise order.

As a doubly linked list p ↔ q ↔ r ↔ s ↔ t ↔ u

Greedy strategy

Draw a line between all pairs of points. If the line has no points on one side, then it's part of the hull

n points $O(n^2)$ segments

$O(n)$ test complexity

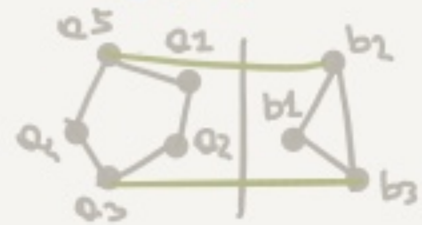
$\Rightarrow O(n^3)$

Sort the point by x coord. (once for all)

For input set S

- Divide into left half A and right half B by x coord
- Compute CH(A) and CH(B)
- Combine

Merging



Starting point for the subproblem is the coord with highest x value.

For the right half is the coord with lowest x value

a5 b2 upper tangent

$y(i,j)$ is maximum

lower tangent

$y(i,j)$ is minimum

Obvious merge algorithm, $O(n^2)$ looking at all pairs of points

Two finger algorithm $\Theta(n)$

```
i = 1
j = 1
while (y(i,j+1) > y(i,j) or y(1-i,j) > y(i,j))
    if y(i,j+1) > y(i,j)   // move left finger clockwise
        j = j + 1 (mod q)
    else
        i = i - 1 (mod p)   // move right finger anti-clockwise
return (ai, bj)
// upper tangent
```

Similar for lower tangent

$T(n) = \underset{\text{subproblems}}{2T(\frac{n}{2})} + \underset{\text{merge}}{\Theta(n)}$

$\Rightarrow \Theta(n \log n)$

Divide the points to get roughly equal problems

Median finding

Find a median in better than $\Theta(n \log n)$ time

sort and go to a n/2

Given a set of n numbers define rank(x) of numbers that are $\leq x$

Select(S, i) ← rank

Find element of rank $\frac{n+1}{2}$

- pick $x \in S$
- compute k (rank of x)

$B = \{y \in S \mid y < x\}$

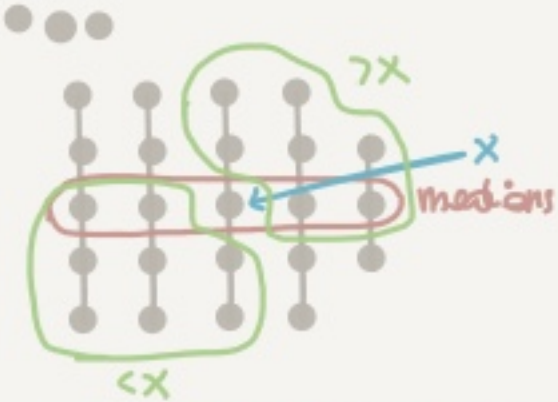
$C = \{y \in S \mid y > x\}$

B	x	C
k−1 elements		n−k elements

```
if k == i : return x
elif k > i : return Select(B, i)
elif k < i : return (C, i-k)
```

Pick x cleverly

- Arrange S into columns of size 5 ($\frac{n}{5}$ columns)
- Sort each columns (big elements on top) linear time
- Find median of medians



Half of the $\frac{n}{5}$ groups contribute at least 3 elements $> x$ except for 1 group with less than 5 elements and 1 group that contains x

At least $3(\lceil \frac{n}{10} \rceil - 2)$ elements are $> x$

// // // $< x$

Recurrence

$$T(n) = \begin{cases} O(1) & n \le 140 \\ T(\lceil \frac{n}{5} \rceil) + T(\frac{7n}{10} + 6) + \Theta(n) \end{cases}$$

Find median of medians

discard at least $\frac{3n}{10} - 6$ elements

sorting, constant for every column (because few elements) and there's $\frac{n}{5}$ columns

R1

Strassen algorithm

for matrix multiplication

$$\underset{A}{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}} \underset{B}{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}} = \underset{C}{\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}}$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21}$
  $C_{12} = A_{11}B_{12} + A_{12}B_{22}$
  $C_{21} = A_{21}B_{11} + A_{22}B_{21}$
  $C_{22} = A_{21}B_{12} + A_{22}B_{22}$

$$T(n) = \underline{8}T(\tfrac{n}{2}) + O(\tfrac{n}{2}^2)$$

$C_{11} = M_1 + M_4 - M_5 + M_7$
$C_{12} = M_3 + M_5$
$C_{21} = M_2 + M_4$
$C_{22} = M_1 - M_2 + M_3 + M_6$

$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
$M_2 = (A_{21} + A_{22})B_{11}$
$M_3 = A_{11}(B_{12} - B_{22})$
$M_4 = A_{22}(B_{21} - B_{11})$
$M_5 = (A_{11} + A_{12})B_{22}$
$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$
$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

$$T(n) = 7T(\tfrac{n}{2}) + O(n^2)$$

MASTER THEOREM $c = 2 < \log_2 7 \Rightarrow O(n^{\log_2 7})$

Master theorem

$f(n) = O(n^c)$ $c < \log_b a$
$T(n) = \Theta(n^{\log_b a})$

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

$f(n) = \Omega(n^c)$ $c > \log_b a$
$\Theta(f(n))$

L3 FFT

Polynomial: $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \dots + a_{n-1}x^{n-1}$

$\rightarrow \langle a_0, a_1, \dots, a_{n-1} \rangle$ vector notation of polynomials

operations on polynomials

① evaluation $A(x), x_0 \rightarrow A(x_0)$

Horner's rule: $O(n)$ time

$A(x) = a_0 + x(a_1 + x(a_2 + \dots x(a_{n-1})\dots))$

② addition $A(x), B(x) \rightarrow C(x) = A(x) + B(x)\ \forall x$

$c_k = a_k + b_k\ \forall k$ $O(n)$

③ multiplication $A(x), B(x) \rightarrow C(x) = A(x) * B(x)$

$c_k = \sum_{j=0}^{k} a_j b_{k-j}$ $O(n^2) \leftarrow$ BAD

convolution of vectors A, reverse(B)
(inner product of all possible shifts)

Reps: Ⓐ Coefficient vector

Ⓑ Roots $r_0, r_1 \dots r_{n-1}$ $c(x-r_0)(x-r_1)\dots(x-r_{n-1})$

Ⓒ Samples (x_k, y_k) for $k = 0, 1, \dots, n-1$

$A(x_k) = y_k$ x_k distinct

Algs

	coeffs	roots	samples
eval.	$O(n)$	$O(n)$	$O(n^2)$
add.	$O(n)$	∞	$O(n)$
mult	$O(n^2)$	$O(n)$	$O(n)$

Convert in $O(n\log n)$ time between coeffs and samples

Matrix view

x_i's and coefficients

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Vandermonde matrix

compute y_i's by multiplying each x_i row with a to get y_i

$V_{jk} = x_j^k$

coeffs $\rightarrow$ samples $= V \cdot A$, $O(n^2)$

samples $\rightarrow$ coeffs $= V^{-1} \cdot A$, $O(n^2)$

Divide and conquer algo

- goal $A(x)$ for $x \in X$

① Divide into even and odd coeffs

$A_{even}(x) = \sum_{k=0}^{n/2-1} a_{2k} x^k = \langle a_0, a_2, a_4, a_6 \dots \rangle$

$A_{odd}(x) = \sum_{k=0}^{n/2} a_{2k+1} x^k = \langle a_1, a_3, a_5, a_7 \dots \rangle$

③ Combine $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$ for $x \in X$

$(x^2)^k$ $\qquad x(x^2)^k$

② Conquer, recursively compute $A_{even}(y)$ and $A_{odd}(y)$ for $y \in X^2 = \{x^2 \mid x \in X\}$

$T(n) = 2T(\frac{n}{2}, |X|) + O(n+x)$

$O(n^2)$

n (n=x at root)

n n

n n n n

n n n ...

I want $|X|$ to get smaller...

Collapsing set X if

$|X^2| = \frac{|X|}{2}$ X^2 is collapsing OR $X = 1$

$|X|=1$: $x = \{1\}$

$|X|=2$: $x = \{-1, 1\}$ $\sqrt{1}$

$|X|=4$: $x = \{-i, i, -1, 1\}$ $\sqrt[4]{1}$

$|X|=8$: $x = \{\pm\frac{\sqrt{2}}{2}(1+i), \pm\frac{\sqrt{2}}{2}(i-1), i, -i, -1, 1\}$ $\sqrt[8]{1}$

nth roots of unity

- uniformly spaced around unit circle

$(\cos\theta, \sin\theta)$

$= \cos\theta + i\sin\theta \rightarrow e^{i\theta}$ Euler's formula

for $\theta = 0, \frac{\tau}{n}, 2\frac{\tau}{n}, \dots, \frac{n-1}{n}\tau$

$(e^{i\theta})^2 = e^{i(2\theta)} = e^{i(2\theta \bmod \tau)}$

$e^{i\tau} = 1$

$N = 2^k \Rightarrow$ nth roots of unity are collapsing

$= x^k = e^{ik\tau/n}$

$O(n\log n)$

Fast Fourier Transform FFT

= divide and conquer algo for DFT

Discrete Fourier Transform DFT

$= V \cdot A$ for $x_k = e^{ik\tau/n}$

$V_{jk} = x_j^k = e^{ijk\tau/n}$ V is the vandermonde matrix

Fast polynomial mult.

$A^* = FFT(A)$ $B^* = FFT(B)$ coeffs

$C_k^* = A_k^* \cdot B_k^*\ \forall k$ samples

$C = IFFT(C^*)$ turn into coeffs

inverse

Claim: $V^{-1} = \overline{V}/n$ $\qquad \overline{a+ib} = a-ib$

R2

B-TREES

It is sorted by doing inorder traversal like BST

All operations are still $O(\log n)$

Caches read entire blocks of data so we set B to be block's size so we have $O(\log_B n)$ block read each operation

$B \leq$ # nodes $< 2B$ except root

$B-1 \leq$ # keys $< 2B-1$

All leaves have same depth balanced

- Search like in BST, for every node look through all values
- Insertion, like in BST but when a node have $2B-1$ we split it

If parent has $2B-1$ as well, you split that as well. You can split the root as well into a new one

- Deletion

- Swap item to delete with inorder successor if item not already leaf
- Redistribute and merge nodes if there is underflowing in order to satisfy invariants