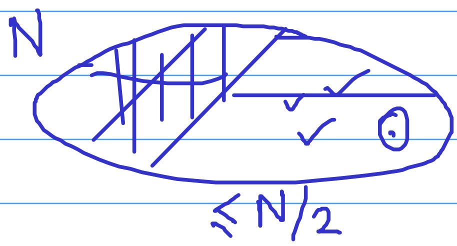
Lecture 1 CS 218 Tue, Jan 3

Binary Search (and variants)

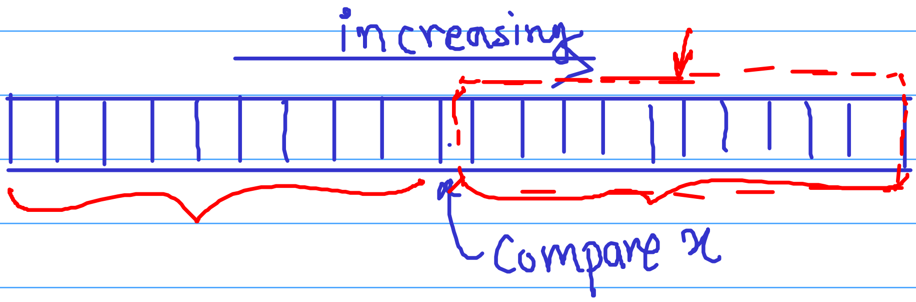
Applicable when with one query, the search space size can be reduced by half. ( )



No. of queries $\log N$

Classic Example:

Given a sorted integer array A and an integer x, find the location of x in A (or say that not present)



Other Examples:

1. Looking for a word in a dictionary
2. Debugging Code
3. Rice cooking

(1) Finding $\lfloor \text{Square root} \rfloor$ of an integer a.

Start with a guess $x \in [1, a]$
Check $\underline{x^2 > a}$

No. of rounds $\underline{\log a}$.

What if we want to output the square root as a real number?

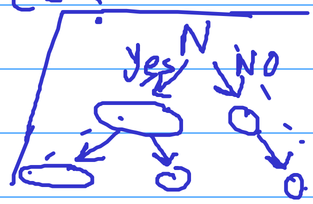
Search space? $a \times 2^k$

No. of queries $\log(a \cdot 2^k) = \log a + k$

k is the no. of precision bits you are asked for.

Better than binary search?

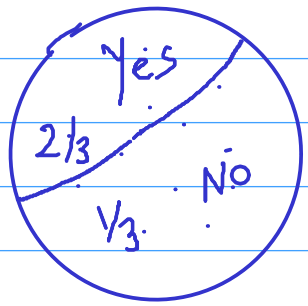
Is there any searching scheme that can work in less than $\log_2 N$ queries?



Ans: NO.

Argument: If the query is Yes/No type then it gives only one bit of information

$\log_4 N = \frac{1}{2} \log_2 N$



In worst case your new search space size $\geq N/2$.

$1/2 \cdot 1/2^2 \cdots 1/2^{\log_2 N}$

Homework ① You have two sorted arrays of Integers. Assume all the entries are distinct in/across the two arrays.

Find the median of the union of two arrays by accessing only $O(\log n)$ entries.

n = size of arrays

HW2 Given an array of integers, and a number S, find a pair of integers in the array whose sum is S.

Trivial : $\binom{n}{2} = O(n^2)$

Another application: suppose for an optimzation problem you can test whether the optimal value is greater than a given number W.

How much time will you take to find the optimal value?

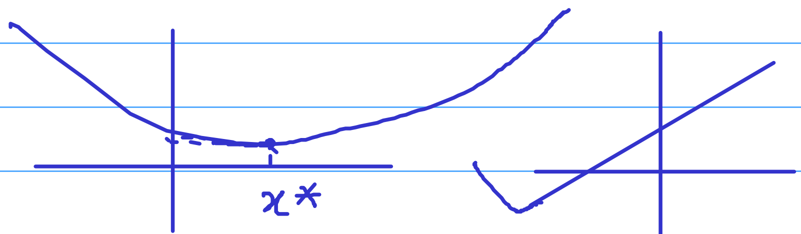
log (initial Range)

What if the range is unknown? (Exponential Search)

Can we find the optimal value in $O(\log(\text{optimal value}))$ queries?

Ans: query with $W = 0, 2, 2^2, 2^3, \ldots$ and stop when optimal value is $\le 2^k$.

HW3 Let $f: \mathbb{R} \to \mathbb{R}$ be a convex function



[equivalently $f'(x)$ is non-decreasing

$f(x)$ is not given explicitly. You can query for $f(x)$ and $f'(x)$ at any point.

Find the point minimizing $f(x)$ (given the promise that x^* exists)

Comment: $f(x) = e^x$ is convex, but has no minimizing point.

Aug 2

Analyzing Algorithms

- Comparing different algorithms

Running time:
Why not implement and see?

- Too many Inputs
- Too many algorithms
- Processor dependent

Will count the number of basic operations.
(addition / comparison)

Asymptotic Analysis

for Input size n running time $f(n)$

$3n-5$ $2n+3$, $5n^2-n+4$.

$\rightarrow O(n)$

Big-O notation.

$O(n)$, $O(n^2)$, $O(n \log n)$, $O(2^n)$

$\Theta(n)$

$f(n) = \sqrt{n} \leftarrow O(n)$

$d \cdot n \leq f(n) \leq c \cdot n$ $\quad \cancel{\leftarrow} \ \Theta(n)$

$A \leftarrow 100 \cdot n \leftarrow O(n)$

$B \leftarrow n^2+9 \leftarrow O(n^2)$

Worst Case Analysis (take the worst bound over all inputs of a fixed size

Why?

① why not average case analysis?

② It's nice to have worst case guarantees and in many cases we can get it.

Describing Algorithms

Pseudocode / Textual description.

(error prone)

implementation details.

Combination of the two

Fri, Jan 6

In an array are there two entries whose sum is S.

Sorting + Binary search vs Hashing

sort the array for each a_i, binary search for $S-a_i$ $O(n\log n)$ comparison $O(n \log n \log N)$	Insert all entries into a Hash table for each a_i search $S-a_i$ in this Hash table. $O(n)$ $h(a_i) = a_i \bmod n$ $O(n(\log N \cdot \log n))$

First Design Idea

Reducing to a subproblem

Same problem on a smaller input (subarray, subgraph)

Assume that you are already given a solution for the subproblem and using that try to build a solution for the original problem.

Solve the subproblem using the same strategy.

Advantage: useful in analyzing the algorithm.

Implementation: recursive or iterative.

Prob 1:

Find minimum value in a given iteger array.

$A = \boxed{a_1 \mid a_2 \mid a_3 \mid \; \mid \; \mid \; \mid \; \mid a_{n-1} \mid a_n}$

subproblem : $\underbrace{\text{min among first } n-1}_{m_{n-1}}$ value

$\rightarrow \quad m_n = \min(m_{n-1}, a_n)$

Recursive	Iterative
$\cdot M(A, i)$:	$m \leftarrow A[1]$
output min value among first i values.	for $i = 2$ to n
if $i = 1 \Rightarrow$ output $A[1]$	$m \leftarrow \min(m, A[i])$
else	
Output $\min(M(A, i-1), A[i])$	

Maximum Subarray Sum problem.

1	2	-5	4	-6	8	7	-3	2	10	3	-7	4	2

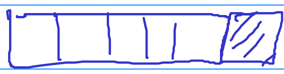
Subarray - contiguous subset.

Given an integer array (possibly with negative entries), find the subarray with maximum sum.

Naive Algorithm:

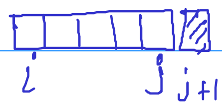
Go over all possible subarrays. and find the one with maximum sum

$O(n^3)$



```
curr-max;
for start = 1 to n.
    s = 0
    for end = start to n
        s = s + A[end];
        curr_max := Max(curr-max, s)
```

going over all subarrays

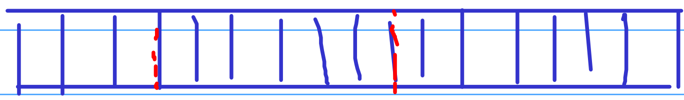


New running time = $O(n^2)$.

Can we improve?

Subproblem

Think about how the subproblem idea can be applied here.



Assume MaxSubarray $(n-1)$ is given.

Can we compute MaxSubarray (n) using it?

- Subarrays of A are of two kinds.
  - which include $A[n]$
  - which are contained in $A[1..n-1]$

$$\text{MaxSubarray}(n) = \underline{\max}\begin{cases} \boxed{\text{Max subarrray}(n-1)} \\ \left.\begin{array}{l} \text{Sum}[1 \ldots n] \\ \text{sum}[2 \ldots n] \\ \vdots \\ \text{Sum}[n-1 \ldots n] \\ \text{Sum}[n] \end{array}\right\} O(n) \end{cases}$$

$$T(n) = T(n-1) + O(n) \Rightarrow T(n) = O(n^2)$$

Improvements?

Ask the subproblem to solve more.

$$\max(\text{sum}[n-1..n], \text{sum}[n-2..n], \ldots, \text{sum}[1..n])$$

$$= \max(A[n]+\text{sum}[n-1], A[n]+\text{sum}[n-2\ldots n-1], \ldots, A[n]+\text{sum}[1..n-1])$$

$$= A[n] + \max(\text{sum}[n-1], \text{sum}[n-2..n-1], \ldots, \text{sum}[1..n-1])$$

Subproblem: maxsubarray$(n-1)$, maxsuffixsum$(n-1)$

$$\text{maxSubarray}(n) = \max\begin{cases}\text{maxsubarray}(n-1)\\ A[n] + \text{maxsuffixsum}(n-1)\\ A[n]\end{cases}$$

$T(n) = T(n-1) + O(1)$

$T(n) = O(n)$

$$\text{maxSuffixsum}(n) = \max(A[n] + \text{maxsuffixsum}(n-1), A[n])$$

maxSuffix = A[1] maxSubarray = max(0, A[1])

```
for (i = 2 to n) {
    maxSubarray = max { maxSubarray
                        maxSuffix + A[i]
                        A[i]
    maxSuffix = max { A[i]
                      maxSuffix + A[i]
}
```

Principle Used:

When designing recursive / inductive idea, sometimes it is useful to solve a more general or harder problem.

Ex

Given share prices for n days $P_1, P_2, \dots P_n$.
Want to buy it on one of the days and sell it on later day. Maximize Profit

$7, 9, 3, 4, 6, 4, \boxed{2}, 4 \dots$ $\qquad O(n)$

Ex Celebrity

Party one celebrity

Queries: ask i^{th} person, whether "You know the j^{th} person.

$O(n)$ queries.

Exponentiation.

Given a, n compute a^n.

Multiplication unit cost.

$$Exp(a, n) = Exp(a, n-1) \times a$$

$$a^n = a^{n-1} \times a$$

no. of multiplication $= n-1$

Repeated squaring

if n is even

$$a^n = \left(a^{n/2}\right)^2 \qquad (1 \text{ mult}$$

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

if n is odd

$T(n) = 2 \log n$

$$\underbrace{a^n}_{n\log a} = \left(a^{\frac{n-1}{2}}\right)^2 \cdot a \qquad (2 \text{ mult}$$

$a^7 =$

$$\begin{tikzcd}
a \arrow[d] \arrow[dr] & \\
a^2 \arrow[r] \arrow[d] & a^3 \arrow[d] \\
a^4 \arrow[r] & a^7
\end{tikzcd}$$

4 multiplications.

$a^{15} = (a^7)^2 \cdot a$ 2 mult.

$a^7 = (a^3)^2 \cdot a$ 2 mult.

$a^3 = a^2 \cdot a$ 2 mult.

6 multiplications

a^{15} in 5 multiplications?

$a^5 = (a^2)^2 \cdot a$

$a^{15} = a^5 \cdot a^5 \cdot a^5$

5 multiplications.

Think

Given n, what is the smallest number of multiplications needed to compute a^n?

Can you design an efficient algorithm to find the smallest number of multiplications required?

Matrix Exponentiation.

$F_n = F_{n-1} + F_{n-2}$

matrix multiplication

Can you compute the n^{th} Fibonacci number in $O(\log n)$ operations?

0, 1, 1, 2, 3

$F_n = \varphi^n + 1/\varphi^n$

Hint

Design Idea 2

Divide and Conquer

- Divide the problem into multiple subproblems of size $n/2$

- Combine the solutions of the subproblems and build a solution for the original problem.

Example Merge sort.

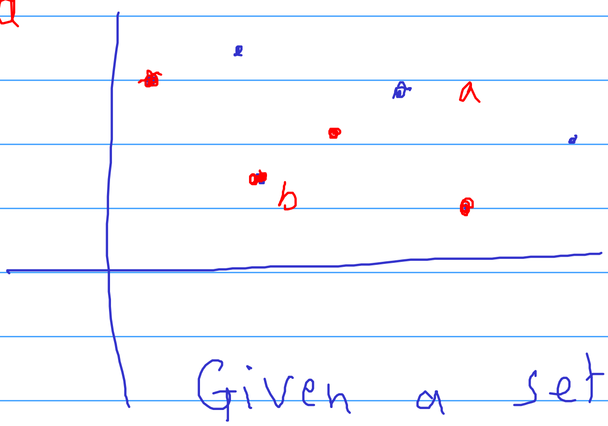
$$T(n) = \underbrace{a}_{\text{no. of subprolems.}} \cdot T(n/2) + \underbrace{f(n)}_{\text{merge}}$$

Divide and Conquer might improve the running time for example

$$O(n^2) \longrightarrow O(n \log n)$$

~~Dominating~~ set problem

Non-dominated



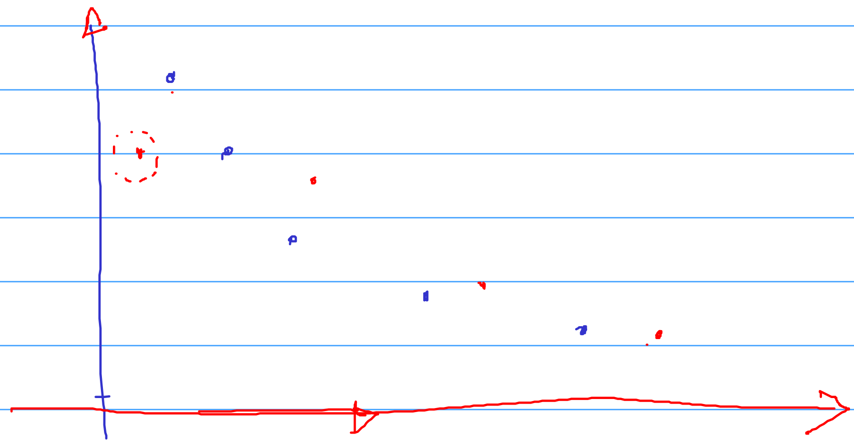
(x, y) dominates (α, β) if $x \geq \alpha$ and $y \geq \beta$

Given a set of points, find those points which are not dominated by any other point.

$T(n) = T(n-1) + O(n)$

$O(n \log n)$

Divide and Conquer



Obs Non-dominated points: from left to right the y-coordinate decreases

Suppose we have two lists of non-dominated points. Assume both the lists are sorted in increasing order of x-coordinates. And thus, they will be decreasing order of y-coordinates.

Merge Procedure:

The merge procedure will take two such lists and will output the set of non-dominated points in the union of the two lists.

We have one pointer for each list, which is initially at the start of the list.

While both lists are non-empty {

Let (a_i, b_i) and (c_j, d_j) be the current points in the two lists.

Find which of the two has the smaller x-coordinate, let it be (a_i, b_i)

if $d_j \geq b_i$ then (a_i, b_i) is dominated. Discard (a_i, b_i) and move the pointer ahead in this list.

otherwise (a_i, b_i) is non-dominated. Insert (a_i, b_i) in the output list and move the pointer ahead in this list.

}

If one of lists is non-empty then insert the remaining points in the output list.

Very similar to merge step in merge sort.

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

$$\underline{T(n) = O(n \log n).}$$

Integer Multiplication.

Bit complexity

Adding two n-bit numbers $\longrightarrow O(n)$

Multiplying two n-bit numbers

$a \times b \rightarrow$ add a, b times $\left(b \uparrow 2^n \right.$

school method

$$\begin{array}{r} 101 \\ 110 \\ \hline 000 \\ 1010 \\ 10100 \\ \hline 11110 \end{array}$$

$000 \leftarrow n$ bits

$1010 \leftarrow n+1$

$10100 \leftarrow 2n-1$ bits

n

$O(n^2)$

$5+4$

$\cancel{0}$

$\cancel{0}$ $\cancel{0}$

$\cancel{0}$ $\cancel{0}$

$\cancel{0}$ $\cancel{0}$

$\cancel{0}$ 0

Can we do better?

Karatsuba [1960] $O(n^{1.58})$

Let's first talk about squaring.

$a \leftarrow n$ bit integer. Find a^2.

let's try to reduce it to squaring of an $n-1$ bit integer

$a = 2a' + \varepsilon$

$\xleftarrow{\quad a' \quad}$ $\varepsilon = 0$ or 1

$\boxed{n-1 \mid \varepsilon}$

$$a^2 = (2a'+\varepsilon)^2 = \underbrace{4 \cdot a'^2}_{2n+3 \text{ bits}} + \varepsilon^2 + 4a'\varepsilon$$

2 left shifts

$n+2$ bits

2 left shifts

$T(n) = O(n^2) \Leftarrow T(n) = T(n-1) + O(n)$

How about divide and conquer?

Reducing to squaring $n/2$ bit integers.

$$a = a_1 \cdot 2^{n/2} + a_0$$

a_1	a_0
$n/2$	$n/2$

$$a^2 = \left(2^{n/2} a_1 + a_0\right)^2$$

$$= \underbrace{2^n}_{\text{left shift by } n \text{ bits}} \cdot \underbrace{a_1^2}_{T(n/2)} + \underbrace{a_0^2}_{T(n/2)} + 2 \cdot 2^{n/2} \cdot \underbrace{a_1 \cdot a_0}_{?}$$

Can we compute $a_1 \cdot a_0$ via squaring?

$$\rightarrow \; 2 \cdot a_1 \cdot a_0 = (a_1 + a_0)^2 - a_1^2 - a_0^2$$

$$a^2 = 2^n \cdot a_1^2 + a_0^2 + 2^{n/2}\left((a_1+a_0)^2 - a_1^2 - a_0^2\right)$$

$O(n)$

squaring in $T(n/2)$?

$$\boxed{T(n) = 3\,T(n/2) + O(n).}$$

$$\Rightarrow \quad T(n) = O\left(n^{\log_2 3}\right) \simeq O\left(n^{1.58}\right)$$

$a_1 + a_0 = 2 \cdot b + \varepsilon$

$(2b+\varepsilon)^2$

$= 4b^2 + 4b\varepsilon + \varepsilon^2$

solving the recurrence

$$T(n) \le c \cdot n + 3T(n/2)$$

$$T(n) \le c \cdot n + 3 \cdot cn/2 + 3^2 \cdot T(n/4)$$

$$\le cn + 3 \cdot cn/2 + 3^2 \cdot cn/2^2 + 3^3 \cdot T(n/8)$$

$$\le \underbrace{cn + \frac{3cn}{2} + \frac{3^2}{2^2} cn + \cdots \frac{3^{\log n - 1}}{2^{\log n - 1}} cn}_{} + 3^{\log n}\, T(1)$$

$$T(n) \le cn \underbrace{\frac{(3/2)^{\log n} - 1}{3/2 - 1}} + \underline{3^{\log n}}$$

$$\boxed{\begin{array}{l} x^{\log n} \\ (2^{\log x})^{\log n} \\ 2^{\log x \cdot \log n} \\ (2^{\log n})^{\log x} \\ n^{\log x} \end{array}}$$

$$= 2cn \cdot n^{\log 3/2} + n^{\log 3}$$

$$= 2c\, n^{1+\log 3/2} + n^{\log 3}$$

$$= O(n^{\log_2 3}) \approx O(n^{1.585})$$

$$\boxed{\begin{array}{l} \log 3/2 \\ = \log 3 - \log 2 \end{array}}$$

What about multiplication?

$$a \cdot b = \frac{(a+b)^2 - a^2 - b^2}{2}$$

$$a \cdot b = \frac{(a+b)^2 - (a-b)^2}{4}$$

Multiplication directly (without going via squaring)

$a \times b$?

$$a = a_1 \cdot 2^{n/2} + a_0$$

$$b = b_1 \cdot 2^{n/2} + b_0$$

$$ab = a_1 \cdot b_1 \cdot 2^n + a_0 b_1 \cdot 2^{n/2} + a_1 b_0 \cdot 2^{n/2} + a_0 b_0$$

$$= a_1 \cdot b_1 \cdot 2^n + (a_0 b_1 + a_1 b_0)\, 2^{n/2} + a_0 b_0$$

HW: Can these three terms $a_1 b_1$, $a_0 b_1 + a_1 b_0$, $a_0 b_0$ be computed somehow with three multiplications?

Hint: first compute $(a_1 + a_0)(b_1 + b_0)$

$n/2$ bits

two more multiplications allowed.

Can we instead divide into three parts?

squaring $a = \underline{a_2} \cdot 2^{2n/3} + \underline{a_1} \cdot 2^{n/3} + \underline{a_0}$

$$a^2 = \underbrace{a_2^2} \cdot 2^{4n/3} + \underbrace{2a_2a_1} \cdot 2^{n} + \left(\underbrace{a_1^2 + 2a_0a_2}\right) 2^{2n/3}$$
$$+ \underbrace{2a_1a_0} \cdot 2^{n/3} + \underbrace{a_0^2} \cdot 2^0$$

$$T(n) = \alpha\, T(n/3) + O(n)$$
$$T(n) = O\left(n^{\log_3 \alpha}\right)$$

current best $n^{1.585}$

$\log_3 6 = 1.63$

$\log_3 5 = \underline{1.46} \implies O(n^{1.46})$

Can we compute the desired terms with 5/6 squarings? + $O(n)$ operations.

easy to do with 6 squarings.

$a_0^2,\ a_1^2,\ a_2^2,\ (a_0+a_1)^2,\ (a_0+a_2)^2,\ (a_1+a_2)^2$

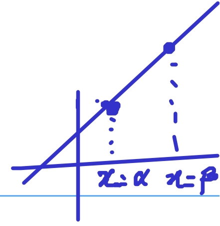
Idea:

To improve, we need to see it as a squaring of a polynomial.

$$A(x) = a_2 x^2 + a_1 x + a_0$$

$$(A(x))^2 = \underbrace{a_2^2}\, x^4 + \underbrace{2a_1a_2}\, x^3 + \underbrace{(a_1^2 + 2a_0a_2)}\, x^2$$
$$+ \underbrace{2a_0a_1}\, x + \underbrace{a_0^2}$$

Polynomial Representations: for degree d



$\rightarrow$ coefficients $d+1$

$\rightarrow$ Roots d

$\rightarrow$ Evaluations $d+1$

$x = \alpha_1$
$x = \alpha_2$
$x = \alpha_3$
$x = \alpha_{d+1}$

How easy/difficult it is to square a Polynomial in evaluation representation?

Given evaluations of $A(x)$, computing evaluations of $A^2(x)$?

$$A^2(\alpha) = (A(\alpha))^2$$

$\rightarrow$ Each evaluation of $A^2(x)$ needs one squaring

$\rightarrow$ How many evaluations of $A^2(x)$ needed?

Five evaluations $\rightarrow$ five squarings.

But we are really interested in coeff of $A^2(x)$.

Plan:

$$\underset{a_0, a_1, a_2}{\text{Coeffs of } A(x)} \xrightarrow[\text{① } O(n)]{?} \text{Evaluations of } A(x) \text{ (five)}$$

$$\text{② squaring} \downarrow$$

$$\text{coeffs of } A^2(x) \xleftarrow[\text{③ } \boxed{O(n)}]{?} \text{Evaluations of } A^2(x) \text{ (five)}$$

①

$$A(x) = a_2 x^2 + a_1 x + a_0$$

$$A(x=\alpha) = a_2 \alpha^2 + a_1 \cdot \alpha + a_0 \qquad \boxed{O(n)}$$

$a_0, a_1, a_2 \rightarrow n/3$ bits

$x = 0, 1, -1, 2, -2$

$A(0) = a_0 \qquad \underbrace{A(1)}_{} = a_0 + a_1 + a_2 \qquad A(-1) = a_0 - a_1 + a_2$

$n/3+4$ bits $\leftarrow A(2) = a_0 + 2a_1 + 4a_2$

② $A(0)$, $A(1)$, $A(-1)$, $A(2)$, $A(-2)$

$$\Big\downarrow \text{square}$$

$A^2(0)$, $A^2(1)$, $A^2(-1)$, $A(2)$, $A(-2)$

$$\boxed{5T(n/3) + O(n)}$$

③ Define $S(x) := A^2(x)$

How are coeffs and Evaluations of $S(x)$ are related?

$$S(0) = a_0^2$$
$$S(1) = a_0^2 + 2a_0a_1 + a_2^2 + 2a_0a_2 + 2a_1a_2 + a_2^2$$
$$S(2) = a_0^2 + 2 \cdot 2a_0a_1 + 4\,(a_2^2 + 2a_0a_2) + 8 \cdot 2a_1a_2 + 16\,a_2^2$$

$$\begin{bmatrix} S(0) \\ S(1) \\ S(2) \\ S(-1) \\ S(-2) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & -2 & 4 & -8 & 16 \end{bmatrix} \begin{bmatrix} a_0^2 \\ 2a_0a_1 \\ a_1^2 + 2a_0a_2 \\ 2a_1a_2 \\ a_2^2 \end{bmatrix}$$

$$\text{eval-vector} = \Big[\; M \;\Big]_{5\times 5} \quad \text{coeff-vector}$$

$$M^{-1} \cdot \text{eval-vector} = \underbrace{\text{Coeff-vector}}$$

can we do this in $O(n)$?

We can pre-compute M^{-1} and store it.

Once computed, M^{-1} can be used to square any integer.

M^{-1} .. eval-vector

5×5

$$\left(\frac{n}{3}+4\right) \times 2 = O(n)$$

All entries of M^{-1} are constants.

HW Multiplication/division of an n bit integer with a constant can be done in $O(n)$ bit operations.

need 25 multiplications and 20 additions.
Overall $O(n)$ time.

$$T(n) = 5T(n/3) + O(n)$$

$$T(n) \approx O(n^{1.46}) \quad \text{Toom-Cook}$$

Integer Multiplication History

1960 Karatsuba $O(n^{1.585})$

Toom Cook $O(n^{1.46})$

can be further generalized by dividing the integers into more parts and get better and better time complexity.
But, the time complexity will remain something like $O(n^{1+\varepsilon})$ for $\varepsilon > 0$.

1971 Schönhage Strassen $O(n \log n \log\log n)$

2005 Fürer $O(n \log n \; 2^{\log^* n})$

2019 Harvey, Van der Hoeven $O(n \log n)$

Ideas: polynomial evaluation (also known as discrete fourier Transform) divide and conquer and other ideas.

Quiz? notes.

Lecture 4

Jan 13

Last class

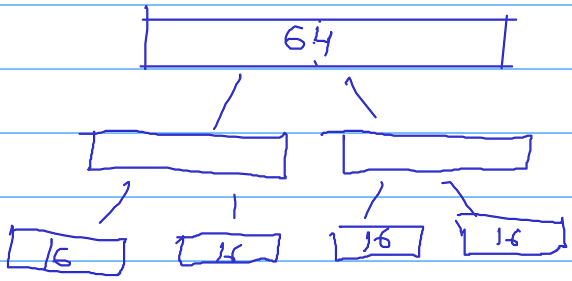
divide and conquer for integer multiplication

Karatsuba $O(n^{\log 3}) \approx O(n^{1.58})$

In practice:

may be slower than the school method say for 64 bit int

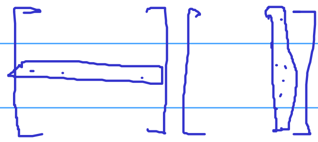
combination of Karatsuba and school method may be better.



Similar ideas can be applied to

- Matrix multiplication
- Po

- Matrix multiplication

A and B are $n \times n$ matrices 

find $A \cdot B$

Naive algorithm $O(n^2 \times n) = O(n^3)$

Divide and conquer

verify

$$\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix} = \begin{bmatrix} A_1B_1 + A_2B_3 & A_1B_2 + A_2B_4 \\ A_3B_1 + A_4B_3 & A_3B_2 + A_4B_4 \end{bmatrix}$$

Assume a subroutine for $n/2 \times n/2$ matrices

no. of multiplications = 8

no. of additions = 4

Recurrence $T(n) = \boxed{8}\,T(n/2) + O(n^2)$ 7

$T(n) = \boxed{O(n^3)}$ $O(n^{\log 7}) \approx O(n^{2.81})$ Strassen

current $O(n^{2.37\ldots})$ want: $O(n^2 \log^c n)$

Puzzle (Secret sharing)

A resource shared ownership — n people

Should be accessible only if - at least k of them together.

Polynomial Multiplication

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d$$

$$b(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_d x^d$$

$$a \times b = a_0 b_0 + (a_1 b_0 + a_0 b_1) x + (a_2 b_0 + a_1 b_1 + a_0 b_2) x^2 + \cdots\cdots + a_d b_d x^{2d}$$

$$= \sum_{j=0}^{2d} x^j \left(\sum_i a_i b_{j-i} \right)$$

Naive algorithm $O(d^2)$

(unit cost arithmetic operations)

Karatsuba? Verify.

Convolution (discrete)

$a = (a_0, a_1, a_2, \ldots, a_m) \in \mathbb{R}^{m+1}$

$b = (b_0, b_1, b_2, \ldots, b_n) \in \mathbb{R}^{n+1}$

$O(mn)$

$n+m+1$

$a * b =$

$(a_0 b_0,\ a_1 b_0 + a_0 b_1,\ a_2 b_0 + a_1 b_1 + a_0 b_2, \ldots\ldots,\ \left(\sum_i a_i b_{j-i}\right),\ \ldots,\ a_m b_n)$

$a_0 b_0$	$a_1 b_0$	$a_2 b_0$	$a_3 b_0$	$\ldots$	$a_m b_0$
$a_0 b_1$	$a_1 b_1$	$a_2 b_1$	$a_3 b_1$	$\ldots$	$a_m b_1$
$a_0 b_2$	$a_1 b_2$	$a_2 b_2$	$a_3 b_2$	$\ldots$	$a_m b_2$
		$\vdots$			
$a_0 b_n$	$a_1 b_n$	$a_2 b_n$	$a_3 b_n$	$\ldots$	$a_m b_n$

Sliding window

$a_0 \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6$

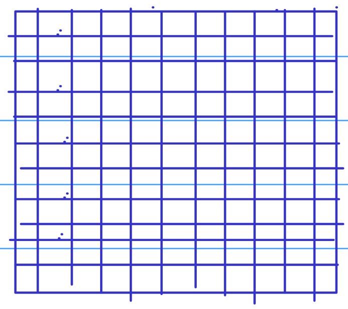
b_2	b_1	b_0

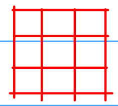
Applications

- Signal processing
  - Smoothening of noisy data
    e.g. 7-day averages of covid cases

- Image processing

2d convolution.





polynomial in 2 variables

- Probability

A dice

Outcome	1	2	3	4	5	6
prob	0.2	0.1	0.05	0.3	0.15	0.2

Roll two dice and take sum of the two

compute probabilities of all outcomes.

2, 3, 4, --- , 12

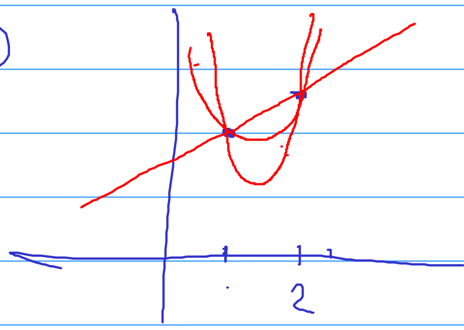
0.04 0.06 . . . . -

Convolution?

Polynomial multiplication / convolution

- Faster algorithms.

Representation of polynomials $a(x)$



- Coefficients
- Roots
- Evaluations. $x_1, x_2, \dots, x_{d+1}$

$a(x_1), a(x_2), \dots, a(x_{d+1})$

Claim Given d+1 evaluations, there is a
HW unique degree d polynomial satisfying those.

Computation in evaluation representation

	Coeff	Roots	Evaluations.
Multiplication	$O(d^2)$	$O(d)$	$O(d)$
addition	$O(d)$	?	$O(d)$

How efficiently can we compute evalutions from coefficients?

d+1 coefficients $\longrightarrow$ d+1 evaluations $O(d^2)$

$a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 \dots + a_d x_1^d$ $O(d)$

$a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3 \dots + a_d x_2^d$ $x_0 = 0$

May be we can choose evaluation points cleverly and find correlations among evaluations?

$$a(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{d-1} x^{d-1}$$

$\boxed{a(1)} = a_0 + a_1 + a_2 + \dots + a_{d-1}$ } $d-1$ additions (d is even)

$\boxed{a(-1)} = a_0 - a_1 + a_2 - a_3 \quad - a_{d-1}$ } $d-1$ additions

$a_0 + a_2 + a_4 \dots a_{d-2}$ } $d/2 - 1$ additions

$a_1 + a_3 + \dots + a_{d-1}$ } $d/2 - 1$ additions

$2d-2$ additions $\longrightarrow$ d additions.

work reduced by half.

$$a(\alpha) = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 + \dots$$

$$a(-\alpha) = a_0 - a_1\alpha + a_2\alpha^2 - a_3\alpha^3 + \dots$$

$a_{even}(x) = a_0 + a_2 x + a_4 x^2 + \dots$ degree $\frac{d-1}{2}$

$a_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \dots$ degree $\frac{d-1}{2}$

$$a(\alpha) = a_{even}(\alpha^2) + \alpha \cdot a_{odd}(\alpha^2)$$

$$a_0 + a_2\alpha^2 + a_4\alpha^4 + \alpha(a_1 + a_3\alpha^2 + a_5\alpha^4)$$

$$a(-\alpha) = a_{even}(\alpha^2) - \alpha \cdot a_{odd}(\alpha^2)$$

One Degree $d-1$ $\longrightarrow$ 2 degree $\frac{d-1}{2}$ polynomials

2 evaluation | 1 evalution.

Two points — a polynomial of deg d

$$a_{even} = a_0 + a_2 x + a_4 x^2 \cdots + a_{d-1} x^{\frac{d-1}{2}}$$

$$a_{odd} = a_1 + a_3 x + a_5 x^2 \cdots + a_d x^{\frac{d-1}{2}}$$

$$a(x) = a_{even}(x^2) + x\, a_{odd}(x^2)$$

$$a(\alpha), \quad a(-\alpha)$$

$$\boxed{\begin{aligned} a(\alpha) &= a_{even}(\alpha^2) + \alpha\, a_{odd}(\alpha^2) \\ a(-\alpha) &= a_{even}(\alpha^2) - \alpha\, a_{odd}(\alpha^2) \end{aligned}}$$

deg d, two evaluations, one poly

$$\boxed{2c \cdot d}$$

deg $\frac{d-1}{2}$, one evaluation, two polys

$$\boxed{2 \cdot c \cdot \frac{d-1}{2}}$$

$$\alpha_1, \alpha_2, \cdots, \alpha_{2d+1}$$

$$\alpha_1, -\alpha_1, \alpha_2, -\alpha_2, \cdots$$

deg d poly at k points

two polynomials, degree $\frac{d-1}{2}$, $\frac{k}{2}$ points

Old set $\alpha_1, -\alpha_1, \alpha_2, -\alpha_2, \alpha_3, -\alpha_3, \ldots$

New set of points $\underbrace{\alpha_1^2, \alpha_2^2}, \underbrace{\alpha_3^2, \alpha_4^2} \ldots$

Need $\boxed{\alpha_1^2 = -\alpha_2^2}$ for applying same trick again

4 points $i, -1, -i, 1$ $\quad i = \sqrt{-1}$

4 points $\underbrace{i, -i}, \underbrace{-1, 1}$

2 points $\underbrace{-1, 1}$

1 point 1

$i \leftarrow$ 4^{th} root of unity

8^{th} root of unity

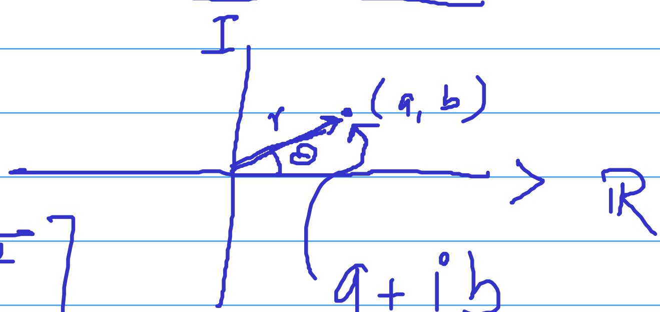
apply this k times $\quad 2^k$ th root of unity

$a + ib$

$= re^{i\theta}$

$r = \sqrt{a^2 + b^2}$

$\tan\theta = \frac{b}{a}$

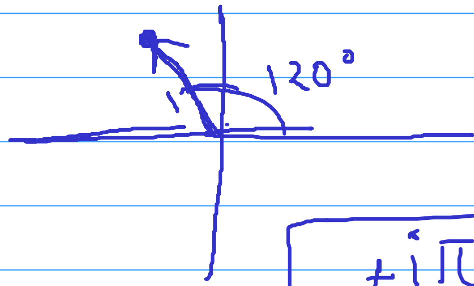


w is k^{th} root of unity

$$w^k = 1$$

$k=3$

$$\boxed{e^{2\pi i} = 1}$$



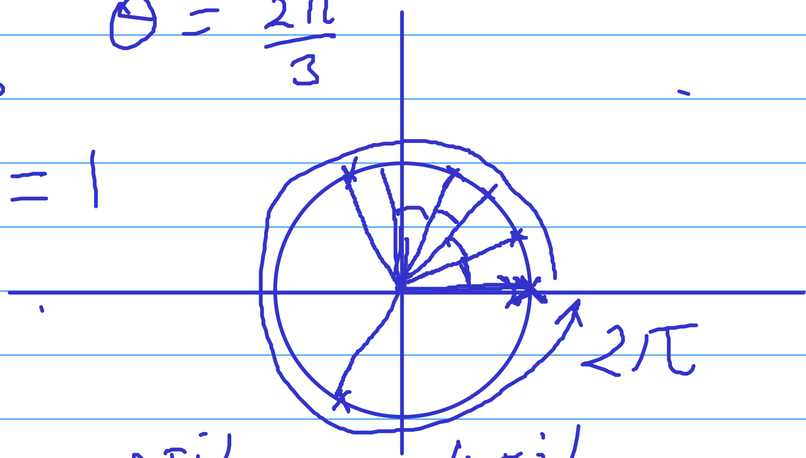
$$\left(e^{2\pi i/3}\right)^3 = 1$$

$$\boxed{e^{+i\pi} = -1}$$

$$r e^{i\theta} \qquad r = 1$$

$$\theta = \frac{2\pi}{3}$$

$$\left(e^{4\pi i/3}\right)^3 = 1$$



$$e^0, \quad e^{2\pi i/3}, \quad e^{4\pi i/3}$$

k^{th} roots

$$e^0, \; e^{2\pi i/k}, \; e^{4\pi i/k}, \; \dots, \; e^{\frac{k-1}{k}\cdot 2\pi i}$$

Properties of k^{th} roots of unity

$k \to$ even ① $\quad w = e^{2\pi i/k} \Rightarrow \boxed{w^{k/2} = e^{i\pi} = -1}$

$$w^0, w^1, w^2, w^3 \; \dots, \; w^{k-1}$$

$$-w^j = e^{i\pi} w^j = w^{k/2} w^j = w^{j+\frac{k}{2}}$$

$$\omega^{k/2} = -1$$

$$-\omega^{j} = \omega^{j+k/2}$$

(2) $\omega^0, \omega^1, \omega^2, \omega^{k/2}, \dots, \omega^{k-1}$] k^{th} roots of unity

$$\underbrace{\omega^0, \omega^2, \omega^4, \omega^k, \omega^{k+2}, \dots, \omega^{2k-2}}_{k/2 \text{ distinct numbers}}$$

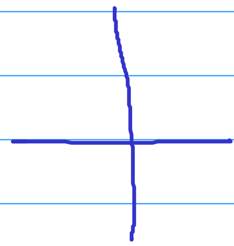
$\frac{k}{2}$th roots.

$$e^0, \; e^{2\cdot 2\pi i/k}, \; e^{2\cdot 4\pi i/k}, \; e^{2\cdot 6\pi i/k}, \;$$

---

(3) $\omega^0 + \omega + \omega^2 + \dots + \omega^{k-1} = 0$

$$\omega^j = -\omega^{j+k/2}$$



Polynomial Evaluation

(discrete Fourier Transform)

$a(x)$ deg $d-1$

d is a power of 2

Evaluate $a(x)$ over d points

$O(d^2)$ arithmetic operations

the d^{th} roots of unity.

$O(d \log d)$ arithmetic operations.

$$\omega \leftarrow e^{2\pi i/d}$$

$$\omega^0, \omega^1, \omega^2, \omega^3, \ldots, \omega^{d-1}$$

$$a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{d-1} x^{d-1}$$

$$a_{odd} = a_1 + a_3 x + a_5 x^2 + \cdots + a_{d-1} x^{\frac{d}{2}-1}$$

$$a_{even} = a_0 + a_2 x + a_4 x^2 + \cdots + a_{d-2} x^{\frac{d}{2}-1}$$

$$a(x) = \underline{a_{even}(x^2)} + x \cdot \underline{a_{odd}(x^2)}$$

Evaluate $a(x)$ over $\omega^0, \omega^1, \omega^2, \omega^{d/2}, \ldots, \omega^{d-1}$

$\downarrow$

Evaluate $a_{even}(x)$ $\omega^0, \omega^2, \omega^4, \ldots, \omega^{d-2}$

$a_{odd}(x)$ $= \gamma^0, \gamma^1, \gamma^2, \ldots \gamma^{d/2-1}$

One polynomial of degree $d-1$
evaluate at d^{th} roots of unity
(d points)

$\downarrow$

two polynomials of degree $\frac{d}{2}-1$
evaluate at $\frac{d}{2}$ th roots of unity
($d/2$ points)

$T(d)$ = no. of arithmetic operations in evaluating a degree $d-1$ polynomial at d th roots of 1.

$$T(d) = 2T(d/2) + O(d)$$

[d additions
d multiplications]

$$T(d) = O(d \log d)$$

Fast Fourier Transform (FFT)

$O(d \log d)$ $\boxed{\text{Polynomial Multiplication / Convolution}}$

$O(d \log d)$ 1. Evaluate
$O(d)$ 2. Multiply these Evaluations
$O(d \log d)$ 3. compute coeffs from Evaluations

$$a(x) = a_0 + a_1 x + \cdots \quad a_{d-1} x^{d-1}$$

Evaluate over d^{th} roots of unity

$$\begin{bmatrix} a(\omega^0) \\ a(\omega^1) \\ a(\omega^2) \\ \vdots \\ a(\omega^{d-1}) \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{d-1} \\ 1 & \omega^2 & \omega^4 & & \cdots & \omega^{2d-2} \\ & & & & & \\ 1 & \omega^{d-1} & & & \cdots & \omega^{(d-1)^2} \end{bmatrix}}_{M} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \\ a_{d-1} \end{bmatrix}$$

M^{-1} $\qquad$ $O(d \log d)$

$$M^{-1} \cdot \text{Eval} = \text{coeff}$$

Claim $\quad M' = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & & \omega^{-(d-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \\ & & & & \\ 1 & \omega^{-(d-1)} & & \cdots & \omega^{-(d-1)^2} \end{bmatrix}$

HW $\quad MM' = dI$

$M^{-1} \cdot \text{eval.}$] $O(d \log d)$ operations