Binary Search (and variants)

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

\[ \text{No. of queries} \leq \log N \]

Classic Example:
Given a sorted integer array \( A \) and an integer \( x \), find the location of \( x \) in \( A \) (or say that \( x \) is not present).

Other Examples:
1. Looking for a word in a dictionary
2. Debugging code
3. Rice cooking

Finding \( \lfloor \sqrt{a} \rfloor \) of an integer \( a \).
Start with a guess \( x \in [1, a] \)
Check \( x^2 > a \)
No. of rounds \( \log a \).
What if we want to output the square root as a real number?

Search space: \( a \times 2^k \)

No. of queries: \( \log(a \times 2^k) = \log a + k \)

\( k \) 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 \( \log N \) 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 \( \geq \frac{N}{2} \).

\[
\frac{1}{2} \cdot \frac{1}{2} \cdot \ldots \cdot \frac{1}{2^{\log_2 N}}
\]

Homework: 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.

\( N = \text{size of arrays} \)
HW2. Given an array of integers, and a number $S$, find a pair of integers in the array whose sum is $S$.

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

Another application: suppose for an optimization 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?

$\log(\text{initial Range})$

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 $\leq 2^k$.

HW3. Let $f: \mathbb{R} \to \mathbb{R}$ be a convex function. $f(x)$ 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.
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 \rightarrow 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} \leq O(n)

d:n \leq f(n) \leq c \cdot n \rightarrow \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)

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 \( S - a_i \)
  \( \Theta(n \log n) \) comparison
  \( \Theta(n \log n \log N) \)

- Insert all entries into a Hash table
- For each \( a_i \),
  Search \( S - a_i \) in this Hash table
  \( \Theta(n) \)
  \( h(a_i) = a_i \mod n \)
  \( \Theta(n(\log N \cdot \log n)) \)
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 integer array.

\[ A = [a_1, a_2, a_3, \ldots, a_{n-1}, a_n] \]

Subproblem: min among first \( n-1 \) value

\[ m_n = \min(m_{n-1}, a_n) \]

---

Recursive

\[ M(A, i) : \]
output min value among first \( i \) values.
if \( i = 1 \) => output \( A[i] \)
else output \( \min(M(A, i-1), A[i]) \)

Iterative

\[ m \leftarrow A[i] \]
for \( i = 2 \) to \( n \)
\[ m \leftarrow \min(m, A[i]) \]
Maximum Subarray Sum problem.

\[ 1 \ 2 \ -5 \ -4 \ 6 \ 8 \ 7 \ -3 \ 2 \ 0 \ 3 \ -7 \ 4 \ 2 \]

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 { going over all subarrays
    s = 0
    for end = start to n
      s = s + A[end];
      curr_max := Max(curr_max, s)
  }

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.

\(\text{which are contained in } A[1..n-1]\)

MaxSubarray\((n)\) = \(\max\left\{\begin{array}{c}
\text{Max subarray \((n-1)\)}
\sum_{i=1}^{n} a_i
\sum_{i=2}^{n} a_i
\sum_{i=n-1}^{n} a_i
\sum_{i=n}^{n} a_i
\end{array}\right\} + O(n)\)

\(T(n) = T(n-1) + O(n) \Rightarrow T(n) = O(n^2)\)
Ask the subproblem to solve more.

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

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

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

Subproblem: \( \max \text{subarray}(n-1), \max \text{suffix sum}(n-1) \)

\[
\max \text{Subarray}(n) = \max \left\{ \max \text{subarray}(n-1), A[n] + \max \text{suffix sum}(n-1) \right\}
\]

\[
T(n) = T(n-1) + O(1)
\]

\[
T(n) = O(n^2)
\]

\[
\max \text{Suffix sum}(n) = \max \left( A[n] + \max \text{suffix sum}(n-1), A[n] \right)
\]

\[
\max \text{Suffix} = A[1], \quad \max \text{Subarray} = \max \left( 0, A[1] \right)
\]

for \( i = 2 \) to \( n \):

\[
\max \text{Subarray} = \max \left\{ \max \text{Subarray}, \max \text{Suffix} + A[i], A[i] \right\}
\]

\[
\max \text{Suffix} = \max \left\{ A[i], \max \text{Suffix} + A[i] \right\}
\]

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 \( p_1, p_2, \ldots, p_n \). Want to buy it on one of the days and sell it on later day. Maximize Profit

\[
7, 3, 4, 6, 4, 3, 4, \ldots \quad O(n)
\]

Ex
Celebrity Party: one celebrity you know the

Queries: ask \( i^{th} \) person, whether \( j^{th} \) person. \( O(n) \) queries.
Exponentiation.

Given $a, n$ compute $a^n$.

\[ \exp(a, n) = \exp(a, n-1) \times a \]

\[ a^n = a^{n-1} \times a \]

no. of multiplication = $n-1$

Repeated squaring

if $n$ is even

\[ a^n = (a^{n/2})^2 \]

(1 mult)

\[ T(n) = T(n/2) + 2 \]

if $n$ is odd

\[ a^n = \left( a^{ \frac{n-1}{2} } \right) \cdot a \]

\[ n \log_2 a \]

(2 mult)

\[ T(n) = 2 \log_2 n \]

$a^7 = \left( a^4 \rightarrow a^8 \rightarrow a^7 \right)$ by multiplications.
\[
\begin{align*}
a^{15} &= (a^7)^2 \cdot a & \text{2 mult.} \\
a^7 &= (a^3)^2 \cdot a & \text{2 mult.} \\
a^3 &= a^2 \cdot a & \text{2 mult.} \\
\end{align*}
\]

6 multiplications

\[
\begin{align*}
a^{15} \text{ in 5 multiplications?} \\
\quad a^5 &= (a^2)^2 \cdot a & \text{5 multiplications} \\
\quad a^{15} &= a^5 \cdot a^5 \cdot a^5
\end{align*}
\]

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**

\[
F_n = F_{n-1} + F_{n-2}
\]

Can you compute the \( n \)th Fibonacci number in \( \Theta(\log n) \) operations?

\[
\begin{align*}
0, 1, 1, 2, 3 \\
F_n = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}} \quad \text{(where } \varphi = \frac{1 + \sqrt{5}}{2} \text{)}
\end{align*}
\]

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: Mergesort:

\[
T(n) = a \cdot T(n/2) + f(n)
\]

\( a \) is the number of subproblems.

Divide and conquer might improve the running time for example:

\( O(n^2) \rightarrow O(n \log n) \)

---

Dominating set problem

Non-dominated point \((x, y)\) dominates another point \((a, b)\) if \( x \geq a \) and \( y \geq b \)

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

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

\[ T(n) = O(n\log n). \]
Integer Multiplication

Bit complexity

Adding two n-bit numbers \( \rightarrow O(n) \)

Multiplying two n-bit numbers

\[ a \times b \rightarrow \text{add } a, b \text{ times} \]

School method

\[
\begin{array}{c}
\phantom{10} \\
110 \\
000 \leftarrow \text{n bits} \\
1010 \leftarrow \text{n+1 bits} \\
10100 \leftarrow 2\text{n-1 bits} \\
11110
\end{array}
\]

Can we do better?

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

Let's first talk about squaring.

\( a \leftarrow n \text{ bit integer. Find } a^2 \)

Let's try to reduce it to squaring of an \( n-1 \) bit integer

\( a = 2a' + \varepsilon \)

\( n \) left shifts

\( a^2 = (2a' + \varepsilon)^2 = 4a'^2 + \varepsilon^2 + 4a'\varepsilon \)

\( 2n+3 \) bits

\( T(n) = O(n^2) \iff T(n) = T(n-1) + O(n) \)
How about divide and conquer?
Reducing to squaring \( n/2 \) bit integers.

\[
\begin{align*}
  a &= a_1 \cdot 2^{n/2} + a_0 \\
  a^2 &= (2^{n/2} a_1 + a_0)^2 \\
  &= 2^n \left( \frac{a_1^2 + a_0^2}{2} + 2 \cdot 2^{n/2} \cdot a_1 \cdot a_0 \right) \\
  &= 2^n a_1^2 + a_0^2 + 2 \cdot 2^{n/2} \cdot a_1 \cdot a_0
\end{align*}
\]

\[T(n/2) \leq T(n/2) \]

Can we compute \( a_1 \cdot a_0 \) via squaring?

\[
2^2 a_1 \cdot a_0 = (a_1 + a_0)^2 - a_1^2 - a_0^2
\]

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

\[
T(n) = \begin{cases} 
3 \cdot T(n/2) + O(n) 
\end{cases}
\]

\[
T(n) = \begin{cases} 
O(n^{\log_2 3}) 
= O(n^{1.58}) 
\end{cases}
\]

solving the recurrence

\[
T(n) \leq c \cdot n + 3T(n/2)
\]

\[
T(n) \leq c \cdot n + 3 \cdot c \cdot n/2 + 3^2 \cdot T(n/4)
\]

\[
\leq c \cdot n + 3 \cdot c \cdot n/2 + 3^2 \cdot c \cdot n/2^2 + 3^3 \cdot T(n/8)
\]

\[
\leq \left( n + \frac{3c}{2} n + \frac{3^2c}{2^2} n + \ldots + \frac{3^{\log_2 n} n}{2^{\log_2 n - 1}} \right) + 3^{\log_2 n} T(1)
\]
\[ T(n) = \left( \frac{\log n}{\frac{3}{2} - 1} \right) \log \frac{3}{2} - 1 + 3 \log n \]

\[
\begin{align*}
&= 2(n \log \log n) + n \log \frac{3}{2} + n \log 3
\end{align*}
\]

\[= 0(n \log n) = o(n^{1.585})
\]

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)

\[ a \times b ?
\]

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

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

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

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

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 \((a_1 + a_0)(b_1 + b_0)\) with \(n/2\) bits, two more multiplications allowed.
Can we instead divide into three parts?

\[ a^2 = a_2^2 \cdot 2^{4n/3} + 2a_2a_1 \cdot 2^n + (a_1^2 + 2a_0a_2) \cdot 2^{2n/3} + 2a_1a_0 \cdot 2^{n/3} + a_0^2 \cdot 2^0 \]

\[ T(n) = \alpha T(n/3) + O(n) \]

\[ T(n) = \mathcal{O}(n^{1.585}) \quad \text{current best} \]

\[ \log_3 6 = 1.63 \]
\[ \log_3 5 = 1.46 \implies \mathcal{O}(n^{1.46}) \]

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.

\[ A(x) = a_2 x^2 + a_1 x + a_0 \]

\[ (A(x))^2 = a_2^2 x^4 + 2a_1a_2 x^3 + (a_1^2 + 2a_0a_2) x^2 + 2a_0a_1 x + a_0^2 \]
Polynomial Representations: for degree $d$

$\rightarrow$ coefficients $d+1$

$\rightarrow$ Roots $d$

$\rightarrow$ Evaluations $d+1$

$\lambda = \alpha_1$

$\lambda = \alpha_2$

$\lambda = \alpha_3$

$\lambda = \alpha_{d+1}$

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

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

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

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

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

Five evaluations $\rightarrow$ five squarings

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

**Plan:**

coeffs of $A(\lambda)$ $\xrightarrow{\text{evaluation}}$ Evaluations of $A(\lambda)$

$\rightarrow_1 O(n)$

$\rightarrow_2$ squaring

coeffs of $A^2(\lambda)$ $\xleftarrow{\text{evaluation}}$ Evaluations of $A^2(\lambda)$

$\rightarrow_3 O(n)$

$\rightarrow_5$ (five)

$A(\lambda) = \alpha_2 \lambda^2 + \alpha_1 \lambda + \alpha_0$

$A(\lambda = \alpha_k) = \alpha_k^2 \lambda_k^2 + \alpha_k \lambda_k + \alpha_0$

$a_0, a_1, a_2 \rightarrow \sqrt{3}$ bits

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

$O(n)$

$A(0) = a_0$

$A(1) = a_0 + a_1 + a_2$

$A(-1) = a_0 - a_1 + a_2$

$n/3 + 4$ bits $\xleftarrow{\text{evaluation}} A(2) = a_0 + 2a_1 + 4a_2$
2. \( A(0), A(1), A(-1), A(2), A(-2) \) 
\( \downarrow \) Square 
\( A(0), A^2(1), A^2(-1), A(2), A(-2) \)

3. Define \( S(x) := A^2(x) \)

How are coefficients and evaluations of \( S(x) \) are related?

\[
\begin{align*}
S(0) &= a_0^2 \\
S(1) &= a_0^2 + 2a_0a_1 + a_1^2 + 2a_0a_2 + 2a_1a_2 + a_2^2 \\
S(2) &= a_0^2 + 2 \cdot 2a_0a_1 + 4 (a_1^2 + 2a_0a_2) + 8 \cdot 2a_1a_2 + 16a_2^2
\end{align*}
\]

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

eval-vector = \( M \) \( \begin{bmatrix}
\text{Coeff-vector}
\end{bmatrix} \)

\( M^{-1}, \text{eval-vector} = \frac{\text{Coeff-vector}}{\text{can we do this in } O(n)} \)

We can precompute \( M^{-1} \) and store it.

Once computed, \( M^{-1} \) can be used to square any integer.
\( M^{-1} \): eval-vector
\[ (n/3 + 4) \times 2 = O(n) \]

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

\textbf{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.

\[
T(n) = 5T(n/3) + O(n)
\]

\[
T(n) = O(n^{1.46}) \quad \text{Toom-Cook}
\]

\textbf{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, Vunder Hoeven \( O(n \log n) \)

Ideas: polynomial evaluation (also known as discrete Fourier Transform), divide and conquer and other ideas.
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
- \( \mathbb{P} \)
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

\[
T(n) = \begin{cases}
8T(n/2) + O(n^2) & \text{Strassen} \\
O(n^3) & \text{current} \quad 0(n^{2.373}) \quad \text{want: } 0(n^{2 \log^c n})
\end{cases}
\]

$T(n) = O(n^{3 \log 2})$
Puzzle (Secret sharing)

A resource shared ownership — n people

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

Polynomial Multiplication

\[ a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_d x^d \]

\[ b(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_d x^d \]

\[ a \cdot b = a_0 b_0 + (a_1 b_0 + a_0 b_1) x + (a_2 b_0 + a_1 b_1 + a_0 b_2) x^2 + \ldots + a_d b_d x^{2d} \]

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

Naive algorithm \( O(d^2) \)

(unit cost arithmetic operations)

Karatsuba? Verify.
Convolution (discrete)

\[ a = (a_0, a_1, a_2, \ldots, a_m) \in \mathbb{R}^{m+1} \]
\[ b = (b_0, b_1, b_2, \ldots, b_n) \in \mathbb{R}^{n+1} \]
\[ a \ast b = 0^{n+m+1} \]
\[ (a_0b_0, a_1b_0 + a_0b_1, a_2b_0 + a_1b_1 + a_0b_2, \ldots, \sum a_i b_{j-i}, \ldots, a_mb_n) \]

Sliding window

\[ a_0 \quad a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5 \quad a_6 \]
\[ b_0 \quad b_1 \quad b_2 \]
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 $x_1, x_2, \ldots, x_{d+1}$

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

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

Computation in evaluation representation

<table>
<thead>
<tr>
<th>Multiplication</th>
<th>Coeff</th>
<th>Roots</th>
<th>Evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(d^2)$</td>
<td>$O(d)$</td>
<td>$O(d)$</td>
<td>$O(d)$</td>
</tr>
</tbody>
</table>

addition $O(d)$ $O(d)$ $O(d)$

How efficiently can we compute evaluations from coefficients?

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

\[ a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 + \cdots + a_d x_1^d \]

\[ a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3 + \cdots + a_d x_2^d \]

\[ x_0 = 0 \]
May be we can choose evaluation points cleverly and find correlations among evaluations?

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

\[ a(1) = a_0 + a_1 + a_2 + \ldots + a_{d-1} \quad \text{3 \, d-1 additions (d is even)} \]

\[ a(-1) = a_0 - a_1 + a_2 - a_3 - \ldots - a_{d-1} \quad \text{3 \, d-1 additions} \]

\[ a_0 + a_2 + a_4 + \ldots + a_{d-2} \quad \text{3 \, d/2-1 additions} \]

\[ a_1 + a_3 + \ldots + a_{d-1} \quad \text{3 \, d/2-1 additions} \]

2d-2 additions → d additions.

work reduced by half.

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

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

\[ a_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + \ldots \quad \text{degree } \frac{d-1}{2} \]

\[ a_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + \ldots \quad \text{degree } \frac{d-1}{2} \]

\[ a(x) = a_{\text{even}}(x^2) + x \cdot a_{\text{odd}}(x^2) \]

\[ a_0 + a_2 x^2 + a_4 x^4 + x \left( a_1 + a_3 x^2 + a_5 x^4 \right) \]

\[ a(-x) = a_{\text{even}}(x^2) - x \cdot a_{\text{odd}}(x^2) \]

One \text{Degree } d-1 \rightarrow 2 \text{ degree } \frac{d-1}{2} \text{ polynomials, 2 evaluations, 1 evaluation.}
Two points - a polynomial of deg $d$

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

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

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

$a(x)$, $a(-x)$

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

$$a(-x) = a_{\text{even}}(x^2) - x a_{\text{odd}}(x^2)$$

---

deg $d$, two evaluations, one poly

$$\sqrt{2 \leq d}$$

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

$$2 \leq \frac{d-1}{2}$$

$a_1, a_2, \ldots, a_{2d+1}$

$a_1, -a_1, a_2, -a_2, \ldots$

deg $d$ poly at $k$ points

$$\text{two poly nominals, degree } \frac{d-1}{2}, \frac{k}{2} \text{ points}$$
Old set $\alpha_1, -\alpha_1, \alpha_2, -\alpha_2, \alpha_3, -\alpha_3, \ldots$

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

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

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

4 points $i, -i, i, -i$

2 points $-1, 1$

1 point $1$

$i = 4^{th \text{ root of unity}}$

$8^{th \text{ root of unity}}$

Apply this $2^k$ times $2^{th \text{ root of unity}}$

$a + ib$

$-re^{i\theta}$

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

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

$(a, b) \rightarrow \mathbb{R}$
$w$ is $k^{th}$ root of unity

\[ w^k = 1 \]

\[ \begin{array}{c}
2\pi i \\
\frac{2\pi i}{3}
\end{array} e^{\frac{2\pi i}{3}} = 1 \quad \text{K} = 3 \]

\[ e^{i\pi} = -1 \]

\[ (e^{\frac{2\pi i}{3}})^3 = 1 \]

\[ \begin{array}{c}
2\pi \\
\frac{2\pi}{3}
\end{array} e^{\frac{2\pi}{3}} = 1 \]

\[ e^{\frac{4\pi i}{3}} = 1 \]

\[ e^0, e^{\frac{2\pi i}{3}}, e^{\frac{4\pi i}{3}}, \ldots, e^{\frac{k-1}{3} \cdot 2\pi i} \]

Properties of $k^{th}$ roots of unity

$k$ even \( \therefore w = e^{\frac{2\pi i}{k}} \Rightarrow w^{k/2} = e^{i\pi} = -1 \)

\[ w^0, w^1, w^2, w^3, \ldots, w^{k-1} \]

\[ -w^j = e^{i\pi} w^j = w^{k/2} w^j = w^{j + k/2} \]
\[ w^{K/2} = -1 \]
\[ -w^{j} = w^{j + K/2} \]

\[ \omega, \omega^2, \omega^4, \ldots, \omega^{k-1} \] \( k \) th roots of unity

\[ \omega^0, \omega^1, \omega^2, \ldots, \omega^{2k-2} \]

\( k/2 \) distinct numbers

\[ \frac{k}{2} \text{th roots} \]

\[ e^0, e^{2\pi i/k}, e^{4\pi i/k}, \ldots, e^{2(k-1)\pi i/k} \]

\[ w^0 + w^1 + w^2 + \ldots + w^{k-1} = 0 \]

\[ w^j = -w^{j + K/2} \]
Polynomial Evaluation
(discrete Fourier Transform)

\[ a(x) \text{ deg } d-1 \]
\[ \text{Evaluate } a(x) \text{ over } d \text{ points} \]
\[ O(d^2) \text{ arithmetic operations} \]
\[ \text{the } d^{th} \text{ roots of unity} \]
\[ O(d \log d) \text{ arithmetic operations} \]

\[ w \leftarrow e^{2\pi i/d} \]

\[ w^0, w^1, w^2, w^3, \ldots, w^{d-1} \]

\[ a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{d-1} x^{d-1} \]
\[ a_{\text{odd}} = a_1 + a_3 x + a_5 x^2 + \ldots + a_{d-1} x^{d-1} \]
\[ a_{\text{even}} = a_0 + a_2 x + a_4 x^2 + \ldots + a_{d-2} x^{d-2} \]

\[ a(x) = a_{\text{even}}(x^2) + x \cdot a_{\text{odd}}(x^2) \]

Evaluate \( a(x) \) over \( w^0, w^1, w^2, w^3, \ldots, w^{d-1} \)

Evaluate \( a_{\text{even}}(x) \) at \( w^0, w^1, w^2, w^3, \ldots, w^{d-2} \)

\[ a_{\text{odd}}(x) = x^0, x^1, x^2, \ldots, x^{d-1} \]
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 (\( \frac{d}{2} \) points)

\[ T(d) = \text{no. of arithmetic operations in evaluating a degree } d-1 \text{ polynomial at } d^{th} \text{ roots of unity} \]

\[ T(d) = 2T\left(\frac{d}{2}\right) + O(d) \]

\[ \begin{array}{l}
\text{[ } d \text{ additions} \\
\text{[ } d \text{ multiplications} \\
\end{array} \]

\[ T(d) = O(d \log d) \]

Fast Fourier Transform (FFT)

\[ O(d \log d) \]

\[ \text{Polynomial Multiplication/Convolution} \]

\[ \begin{array}{l}
O(d \log d) \circ \text{ Evaluate} \\
O(d) \circ \text{ Multiply these Evaluations} \\
O(d \log d) \circ \text{ compute coeffs from Evaluations} \\
\end{array} \]
\[ a(x) = a_0 + a_1 x + \ldots + a_{d-1} x^{d-1} \]

Evaluate over \(d\)th roots of unity:

\[
\begin{bmatrix}
    a(w^0) \\
    a(w) \\
    a(w^2) \\
    \vdots \\
    a(w^{d-1})
\end{bmatrix}
= \begin{bmatrix}
    1 & 1 & 1 & \ldots & 1 \\
    1 & w & w^2 & \ldots & w^{d-1} \\
    1 & w^2 & w^4 & \ldots & w^{2d-2} \\
    \vdots \\
    1 & w^{d-1} & \ldots & & w^{d-1}
\end{bmatrix}
\begin{bmatrix}
    a_0 \\
    a_1 \\
    a_2 \\
    \vdots \\
    a_{d-1}
\end{bmatrix}
\]

Let \(M = \begin{bmatrix}
    1 & 1 & 1 & \ldots & 1 \\
    1 & w & w^2 & \ldots & w^{d-1} \\
    1 & w^2 & w^4 & \ldots & w^{2d-2} \\
    \vdots \\
    1 & w^{d-1} & \ldots & & w^{d-1}
\end{bmatrix}\) and \(M^{-1}.\text{Eval} = \text{coeff} \Rightarrow O(d \log d)\)

Claim \(M' = \begin{bmatrix}
    1 & 1 & 1 & \ldots & 1 \\
    1 & w^{-1} & w^{-2} & \ldots & w^{-(d-1)} \\
    1 & w^{-2} & w^{-4} & \ldots & w^{-(2d-2)} \\
    \vdots \\
    1 & w^{-(d-1)} & \ldots & & w^{-(d-1)^2}
\end{bmatrix}\)

\[\text{HW: } MM' = dI\]

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