Outline

Paradigm

1. Break up given instance into considerably smaller ones.
2. Recursively solve those.
3. Combine their solutions into one for the given instance.

Examples of common pattern

- Sorting (Mergesort)
- Counting inversions
- Finding a closest pair of points in the plane

Lower bound for sorting

Sorting

Problem specification

Input: array $A[1 \dots n]$ of integers with $n \geq 1$

Output: Sort(A), i.e., A sorted from smallest to largest

Mergesort

```
procedure MERGE-SORT(A)
    if n = 1 then
        return A
    else
        m ← ⌊n/2⌋
        L ← A[1, ..., m]
        R ← A[m + 1, ..., n]
        return MERGE(MERGE-SORT(L), MERGE-SORT(R))
```

Sorting Lower Bound

Theorem

Every comparison-based sorting algorithm takes $\Omega(n \log n)$ comparisons on arrays of length n.

Proof

- Every such algorithm for a given n can be modeled as a binary decision tree T.
- Depth d of T is the maximum number of comparisons that A makes on arrays of length n.
- Number ℓ of leaves is at least $n! \doteq 1 \cdot 2 \cdot \ldots \cdot n$.
- $\ell \leq 2^d$
- $d \geq \log(\ell) \geq \log(n!)$
- $(n/2)^{n/2} \leq n! \leq n^n$ so $\log(n!) = \Theta(n \log n)$

Counting Inversions

Definition

An inversion in an array $A[1 \dots n]$ is a pair $(i, j) \in [n] \times [n]$ with $i < j$ and $A[i] > A[j]$.

Example

$A = [3, 5, 4, 7, 3, 1]$

Bounds on Inv(A)

Between 0 (sorted) and $\binom{n}{2}$ (reverse sorted).

Problem specification

Input: array $A[1 \dots n]$ of integers with $n \geq 1$

Output: $\text{Inv}(A) \doteq$ number of inversions in A

Count

D&C approach

Counting cross inversions

- Problem specification

  Input: sorted arrays $L[1 \dots n]$ and $R[1 \dots m]$ with $n, m \geq 1$

  Output: Inv(LR)
- Algorithm running in time $O(n + m)$

Running time of Count: $O(n(\log n)^2)$

Improved Count

Input: $A[1 \cdots n]$, an array of length $n \geq 1$

Output: (Inv(A),Sort(A))

```
procedure COUNT-AND-SORT(A)
    if n = 1 then
        return (0, A)
    else
        m ← ⌊n/2⌋
        (c_L, L) ← COUNT-AND-SORT(A[1, ..., m])
        (c_R, R) ← COUNT-AND-SORT(A[m + 1, ..., n])
        c_cross ← COUNT-CROSS(L, R)
        c ← c_L + c_R + c_cross
        B ← MERGE(L, R)
        return (c_L + c_R + c_cross, B)
```

Running time: $O(n \log n)$

Closest Pair of Points in the Plane

Problem specification

Input: $(x_i, y_i) \in \mathbb{R}^2$ for $i \in [n]$

Output: $\delta \doteq \min\{\delta_{i,j} \text{ for } i,j \in [n] \text{ with } i \neq j\}$ where
$\delta_{i,j} \doteq \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

D&C approach

Closest crossing pair in the plane

Closest Pair of Points in the Plane

Pseudocode for recursive case

1. Find x^*, L, and R
2. Recursively compute δ_L and δ_R
3. $\delta^* \leftarrow \min(\delta_L, \delta_R)$
4. $M \leftarrow \{i \in [n] \text{ s.t. } x_i \in (x^* - \delta^*, x^* + \delta^*)\}$
5. Sort M based on y-coordinate
6. $\delta_M \leftarrow \min\{\delta_{M[i],M[j]} \text{ for } i < j < i + 12\}$
7. Return $\min(\delta^*, \delta_M)$

Correctness

Running time

- Using local sorting: $O(n \log n)$ locally and $O(n(\log n)^2)$ overall
- Using presorting: $O(n)$ locally and $O(n \log n)$ overall

Polynomials

Coefficient representation

$A(x) = A_0 + A_1 x + A_2 x^2 + \cdots + A_{n-1} x^{n-1} = \sum_{i=0}^{n-1} A_i x^i$

Evaluation

- $A(x) = A_0 + x\,(A_1 + x(A_2 + \cdots + x(A_{n-1}) \ldots))$ [Horner]
- $O(n)$ arithmetic operations

Sum

- $A(b) + B(x) = (\sum_{i=0}^{n-1} A_i x^i) + (\sum_{i=0}^{n-1} B_i x^i)$
  $= \sum_{i=0}^{n-1} (A_i + B_i) x^i$
- $O(n)$ arithmetic operations

Product

- $A(x) \cdot B(x) = (\sum_{i=0}^{n-1} A_i x^i) \times (\sum_{j=0}^{n-1} B_j x^j)$
  $= \sum_{k=0}^{2(n-1)} C_k x^k$ where $C_k = \sum_{i=0}^{n-1} A_i B_{k-i}$
- $O(n^2)$ arithmetic operations trivially

Alternate Representation of Polynomials

Point-value representation

- No two polynomials of degree at most $n-1$ can agree on n or more points.
- For any fixed choice of n points $x_0, x_1, \ldots, x_{n-1}$, represent $A(x) = \sum_{i=0}^{n-1} a_i x^i$ as $(y_0, y_1, \ldots, y_{n-1})$ where $y_i = A(x_i)$.

Sum

- $O(n)$ arithmetic operations

Product

- $O(n)$ arithmetic operations

Evaluation

- $A(x) = \sum_{i=0}^{n-1} y_i \cdot \prod_{i \neq j=0}^{n-1} \frac{x - x_j}{x_i - x_j}$ [Lagrange interpolation]
- $O(n^2)$ arithmetic operations trivially

Converting Between Representations

Relationship

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ & & & \vdots & \\ 1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{n-1} \end{bmatrix}$$

Conversions

- From coefficient to evaluations: $O(n^2)$
- From evaluations to coefficients: $O(n^3)$
- Can improve both to $O(n \log n)$ by picking evaluations points cleverly: Fast Fourier Transform (FFT)
- Implies that polynomial multiplication in the coefficient representation can be done in time $O(n \log n)$.

Point Set for Efficient Simultaneous Polynomial Evaluation

Splitting the coefficient array

- $A(x) = \sum_{i=0}^{n-1} A_i x^i$
  $= \sum_{i \text{ even}} A_i x^i + \sum_{i \text{ odd}} A_i x^i$
  $= A_{\text{even}}(x^2) + x \cdot A_{\text{odd}}(x^2)$
- $A(-x) = A_{\text{even}}(x^2) - x \cdot A_{\text{odd}}(x^2)$

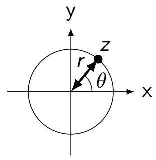
Divide & Conquer approach

- Pick $x_0, \ldots, x_{n-1}$ such that $x_{n/2+i} = -x_i$ for $i \in [n/2] - 1$.
- Evaluation of $A(x)$ at $x_0, \ldots, x_{n-1}$ reduces to evaluation of $A_{\text{even}}(x)$ and $A_{\text{odd}}(x)$ at $x_0^2, \ldots, x_{n/2-1}^2$, with $O(n)$ local work.
- Recursive application requires use of complex numbers. Every complex number $z \neq 0$ has two distinct square roots: $\pm\sqrt{z}$.
- $x_0, \ldots, x_{n-1}$ need to be distinct n-th roots of some $c \in \mathbb{C}$ where n a power of 2. Can pick $c = 1$.

Complex Numbers

Definition
$\mathbb{C} = \mathbb{R}(\sqrt{-1}) = \{x + y\sqrt{-1} \ : \ x, y \in \mathbb{R}\}$

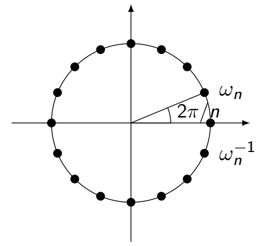
Representation of $z = x + y\sqrt{-1}$



- Cartesian: (x, y)
- Polar: (r, θ) where
  $z = r(\cos(\theta) + \sin(\theta)\sqrt{-1})$

Arithmetic operations
- Sum:
  $(x_1 + y_1\sqrt{-1}) + (x_2 + y_2\sqrt{-1}) = (x_1 + x_2) + (y_1 + y_2)\sqrt{-1}$
- Product:
  $(x_1+y_1\sqrt{-1})\cdot(x_2+y_2\sqrt{-1}) = (x_1x_2-y_1y_2)+(x_1y_2+x_2y_1)\sqrt{-1}$
  $r = r_1 \cdot r_2$ and $\theta = \theta_1 + \theta_2$

Roots of Unity



Primitive n-th root of unity
- Definition: $\omega \in \mathbb{C}$ such that $\langle\omega\rangle \doteq \{\omega^k \ : \ k \in \mathbb{N}\}$ consists of all n-th roots of unity.
- Standard: ω_n
- If ω is primitive n-th root of unity for even n, then ω^2 is primitive $n/2$-th root of unity.

Discrete Fourier Transform

Definition
Input: $n \in \mathbb{N}$; $A_0, \ldots, A_{n-1} \in \mathbb{C}$
Output: $(y_0, \ldots, y_{n-1})$ where $y_i = A(\omega_n^i)$ for
$A(x) = A_0 + a_1x + \cdots + A_{n-1}x^{n-1}$

Algorithm for n a power of 2
- D&C approach runs in time $O(n \log n)$ assuming arithmetic over $\mathbb{C}$ as elementary operations.
- Known as Fast Fourier Transform (FFT).
- ω_n used for specificity in definition of DFT.
- FFT works for arbitrary primitive n-th root.
- Extends to finite fields.

Fast Fourier Transform

Input: n: power of 2; ω: primitive n-th root of 1;
$A_0, \ldots, A_{n-1} \in \mathbb{C}$
Output: $(y_0, \ldots, y_{n-1})$ where $y_i = A(\omega^i)$ for
$A(x) = A_0 + A_1x + \cdots + A_{n-1}x^{n-1}$

Pseudocode

procedure $\mathrm{FFT}(n, \omega, A_0, \ldots A_{n-1})$
  if $n = 1$ **then return** (A_0)
  $(e_0, \ldots, e_{n/2-1}) \leftarrow \mathrm{FFT}(n/2, \omega^2, A_0, A_2, \ldots, A_{n-2})$
  $(f_0, \ldots, f_{n/2-1}) \leftarrow \mathrm{FFT}(n/2, \omega^2, A_1, A_3, \ldots, A_{n-1})$
  $x \leftarrow 1$
  for $k = 0$ to $n/2 - 1$ **do**
    $y_k \leftarrow e_k + x \cdot f_k$
    $y_{n/2+k} \leftarrow e_k - x \cdot f_k$
    $x \leftarrow x \cdot \omega$
  return $(y_0, \ldots, y_{n-1})$

Note: Can be implemented iteratively in-line.

Inverting the Discrete Fourier Transform

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \ldots & \omega_n^{2(n-1)} \\ & & & \vdots & \\ 1 & \omega_n^{n-1} & \omega_n^{(n-1)2} & \ldots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{n-1} \end{bmatrix}$$

- DFT: $\vec{y} = F(\omega_n) \cdot \vec{A}$
- $F(\omega_n^{-1}) \cdot F(\omega_n) = n \cdot I$
  Proof: $(F(\omega_n^{-1}) \cdot F(\omega_n))_{ij} = \sum_{k=0}^{n-1} F(\omega_n^{-1})_{ik} F(\omega_n)_{kj}$
  $= \sum_{k=0}^{n-1} \omega_n^{(j-i)k}$
  $= \begin{cases} n & i = j \\ 0 & i \neq j \end{cases}$
- Inverse DFT: $\vec{A} = F(\omega_n^{-1}) \cdot \vec{y}/n$

Applications of Fast Fourier Transform

Polynomial multiplication in coefficient representation
- Algorithm on input $A(x)$ and $B(x)$ of degree at most d
  1. $\tilde{A} \leftarrow \mathrm{FFT}(n, \omega_n, A)$ and $\tilde{B} \leftarrow \mathrm{FFT}(n, \omega_n, B)$
     where n is smallest power of 2 at least $2d + 1$
  2. $\tilde{C}_i \leftarrow \tilde{A}_i \cdot \tilde{B}_i$ for $i = 0, \ldots, n-1$
  3. Return $C = \mathrm{FFT}(n, \omega_n^{-1}, \tilde{C})/n$
- Running time: $O(n \log n)$ assuming arithmetic on complex numbers as elementary operations

Integer multiplication
- Integer a equals value of polynomial $A(x)$ at $x = 2$.
- Compute $C(x) = A(x) \cdot B(x)$ and output $c = C(2)$.
- Runs in time $O(n \log n)$ assuming complex arithmetic, and time $O(n \cdot \log n \cdot \log\log n)$ using bit operations only.
- Best known: $O(n \log n)$ bit operations.

Integer Multiplication

Problem

Input: nonnegative integers a and b in binary notation

Output: product $a \times b$ in binary notation

Grade school algorithm

$$
\begin{array}{c|cccccccccc}
 & & & & & & 1 & 0 & 0 & 1 & 0 \\
\times & & & & & & 1 & 1 & 1 & 0 & 1 \\
\hline
 & & & & & & 1 & 0 & 0 & 1 & 0 \\
 & & & & 1 & 0 & 0 & 1 & 0 & & \\
 & & & 1 & 0 & 0 & 1 & 0 & & & \\
+ & & 1 & 0 & 0 & 1 & 0 & & & & \\
\hline
 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{array}
$$

Integer Multiplication

Improved D&C approach

- $a \times b = (a_L \times b_L) \cdot 2^n + (a_L \times b_R + a_R \times b_L) \cdot 2^{n/2} + (a_R \times b_R)$
- $a \times b = (a_L \times b_L) \cdot 2^n + (a_R \times b_R) +$
  $((a_L + a_R) \times (b_L + b_R) - (a_L \times b_L) - (a_R \times b_R)) \cdot 2^{n/2}$

Running time

- Recursion tree: $O\left(\left(\sum_{i=0}^{d-1} (\frac{3}{2})^i\right) \cdot n\right)$ where $d = \log_2(n)$
- Geometric sum with ratio r: $\sum_{i=0}^{d-1} r^i = \frac{r^d - 1}{r-1} = \Theta(r^d)$
- For $r = \frac{3}{2}$: $r^d = (\frac{3}{2})^d = \frac{3^d}{2^d}$
- For $d = \log_2(n)$
  - $2^d = n$
  - $3^d = 3^{\log_2(n)} = (2^{\log_2(3)})^{\log_2(n)} = (2^{\log_2(n)})^{\log_2(3)} = n^{\log_2(3)} \doteq n^q$
- Conclusion: $O(\frac{n^q}{n} \cdot n) = O(n^q)$ where $q = \log_2(3) \approx 1.585$

Splitting

- Input: (A, p)
  - array $A[1, \ldots, n]$ of integers
  - integer p
- Output: (L, R)
  - array $L[1, \ldots, |L|]$ consisting of all entries of A less than p
  - array $R[1, \ldots, |R|]$ consisting of all entries of A larger than p
- Algorithm:

procedure SPLIT(A, p)
  $L, R \leftarrow$ empty lists
  for $i = 1$ to n **do**
    if $A[i] < p$ **then**
      append $A[i]$ to L
    else if $A[i] > p$ **then**
      append $A[i]$ to R
  return L and R as arrays

Selection Schema

procedure SELECT(A, k)
  if $n = 1$ **then return** $A[1]$
  pick a pivot p from A
  $(L, R) \leftarrow$ SPLIT(A, p)
  if $k \leq |L|$ **then**
    return SELECT(L,k)
  else if $k > n - |R|$ **then**
    return SELECT(R,$k - (n - |R|)$)
  else
    return p

- Correctness
- Running time assuming linear-time median as pivot: $O(n)$

Approximate Median

Definition

A ρ-approximate median of A is an element p of A such that Split(A, p) returns L and R with $|L| \leq \rho \cdot |A|$ and $|R| \leq \rho \cdot |A|$.

Construction

1. Break up A into consecutive segments of length w.
2. Find the median of each segment.
3. $A' \leftarrow$ subarray consisting of the segment medians.
4. Return the median of A'.

Key claim

The median of A' is a ρ-approximate median of A with $\rho = \frac{3}{4}$.

Linear-Time Selection

Algorithm

procedure FAST-SELECT(A, k)
  if $n = 1$ **then return** $A[1]$
  $A' \leftarrow$ array of medians of the $n' \doteq \lceil \frac{n}{w} \rceil$
      consecutive length-w segments of A
  $p \leftarrow$ FAST-SELECT(A',$\lceil n'/2 \rceil$)
  $(L, R) \leftarrow$ SPLIT(A,p)
  if $k \leq |L|$ **then**
    return FAST-SELECT(L,k)
  else if $k > n - |R|$ **then**
    return FAST-SELECT(R,$k - (n - |R|)$)
  else
    return p

Running time: $O(n)$ for $w \geq 5$