COMPSCI 311: Introduction to Algorithms

Lecture 11: Divide and Conquer

Marius Minea

University of Massachusetts Amherst

Review: Master Theorem for Recurrences

Let $T(n) = aT(n/b) + f(n)$. Then:

1. $T(n) = \Theta(n^{\log_b a})$ when $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$
   $f(n)$ grows polynomially slower than $n^{\log_b a}$
   most work: at bottom of recursion tree
2. $T(n) = \Theta(n^{\log_b a} \log n)$ when $f(n) = \Theta(n^{\log_b a})$
   comparable work at each level
3. $T(n) = \Theta(f(n))$ when $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ *and*
   $af(n/b) < cf(n)$ for some $c < 1$ when n sufficiently large
   $f(n)$ grows polynomially faster than $n^{\log_b a}$
   more work at top level

Clicker Question 1

Consider the recurrence $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$.

Consider three variants of the base case:
1) n = 1; 2) n = 2; 3) n = 1 or n = 2.

The asymptotic complexity of T(n) is:

A) The same in all cases

B) highest for case 1, lowest for case 2, case 3 is intermediate

C) highest for case 1; case 2 and 3 are the same

Counting Inversions: Motivation

n objects, ranked in linear order by different sources

	A	B	C	D	E
RankList1	3	4	2	1	5
RankList2	4	2	1	3	5

How similar are these rankings?

Applications:

- Recommendation systems (collaborative filtering)
- Stability / sensitivity of web ranking functions
- Meta-search tools: compare & aggregate search engines
- Measure "sortedness" of an array

Similarity Metric: Number of Inversions

	A	B	C	D	E
RankList1	3	4	2	1	5
RankList2	4	2	1	3	5

A pair $\{X, Y\} \subseteq \{A, B, C, D, E\}$ has an **inversion** between the two rankings if $rank_1(X) < rank_1(Y)$ but $rank_2(X) > rank_2(Y)$ or vice-versa.

Alternate view: Take Rank1 as reference point and renumber objects based on that rank: D = 1, C = 2, A = 3, B = 4, E = 5.

Rewrite Rank2 as R', ranking each of the new IDs

	1	2	3	4	5
R'	3	1	4	2	5

Say i and j are **inverted** if $i < j$ but $R'(i) > R'(j)$

Is this the same definition? (Has the number of inversions changed?)

Clicker Question 2

	A	B	C	D	E
RankList1	3	4	2	1	5
RankList2	4	2	1	3	5

Rename D = 1, C = 2, A = 3, B = 4, E = 5, rewrite:

	1	2	3	4	5
R'	3	1	4	2	5

Does the number of inversions change?

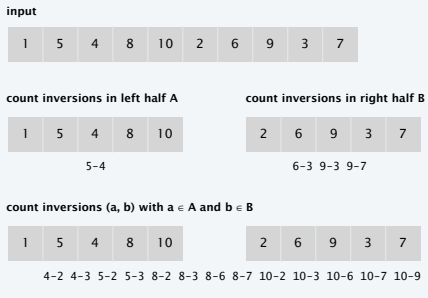
A) Yes, it has changed

B) No, it never changes

C) It has not changed here, but changes in other cases

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with $a \in A$ and $b \in B$.
- Return sum of three counts.



slide credit: Kevin Wayne / Pearson

Clicker Question 3

What do we need to combine the subproblems?

To count inversions between the array halves, we'd need to:

A) Know min and max values in both halves

B) Know min and max values in both halves and their positions

C) Neither of the above is enough

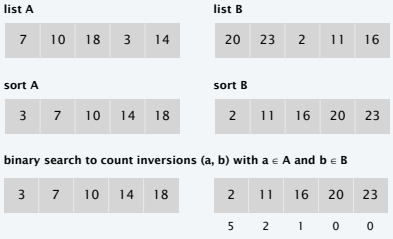
Counting inversions: how to combine two subproblems?

Q. How to count inversions (a, b) with $a \in A$ and $b \in B$?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.
- For each element $b \in B$,
  - binary search in A to find how elements in A are greater than b.

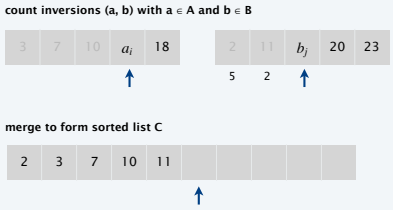


slide credit: Kevin Wayne / Pearson

Counting inversions: how to combine two subproblems?

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i < b_j$, then a_i is not inverted with any element left in B.
- If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.



slide credit: Kevin Wayne / Pearson

Counting inversions: divide-and-conquer algorithm implementation

Input. List L.

Output. Number of inversions in L and L in sorted order.

SORT-AND-COUNT(L)

IF (list L has one element)
  RETURN $(0, L)$.

Divide the list into two halves A and B.

$(r_A, A) \leftarrow$ SORT-AND-COUNT(A). ⟵ $T(n/2)$

$(r_B, B) \leftarrow$ SORT-AND-COUNT(B). ⟵ $T(n/2)$

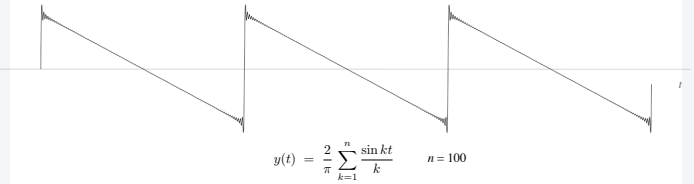
$(r_{AB}, L) \leftarrow$ MERGE-AND-COUNT(A, B). ⟵ $\Theta(n)$

RETURN $(r_A + r_B + r_{AB}, L)$.

slide credit: Kevin Wayne / Pearson

Fourier analysis

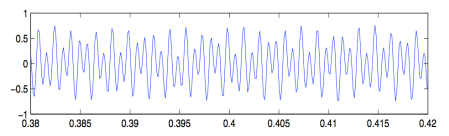
Fourier theorem. [Fourier, Dirichlet, Riemann] Any (sufficiently smooth) periodic function can be expressed as the sum of a series of sinusoids.



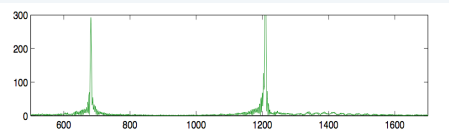
slide credit: Kevin Wayne / Pearson

Time domain vs. frequency domain

Signal. [recording, 8192 samples per second]



Magnitude of discrete Fourier transform.



Reference: Cleve Moler, Numerical Computing with MATLAB

slide credit: Kevin Wayne / Pearson

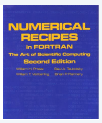
Fast Fourier transform

FFT. Fast way to convert between time domain and frequency domain.

Alternate viewpoint. Fast way to multiply and evaluate polynomials.

we take this approach

> *" If you speed up any nontrivial algorithm by a factor of a million or so the world will beat a path towards finding useful applications for it. "* — *Numerical Recipes*



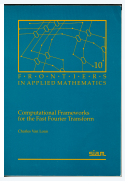
slide credit: Kevin Wayne / Pearson

Fast Fourier transform: applications

Applications.

- Optics, acoustics, quantum physics, telecommunications, radar, control systems, signal processing, speech recognition, data compression, image processing, seismology, mass spectrometry, ...
- Digital media. [DVD, JPEG, MP3, H.264]
- Medical diagnostics. [MRI, CT, PET scans, ultrasound]
- Numerical solutions to Poisson's equation.
- Integer and polynomial multiplication.
- Shor's quantum factoring algorithm.
- ...

> *" The FFT is one of the truly great computational developments of [the 20th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT. "* — *Charles van Loan*



slide credit: Kevin Wayne / Pearson

Representation Tradeoffs

Various data structures implement some operations faster than others

- Arrays vs. Lists
- Union-Find with lists vs. trees

Overall Efficiency: 2 Approaches

- Smart (hybrid) datastructures (best of both worlds)
- Fast conversion between one and the other

Polynomials: coefficient representation

Univariate polynomial. [coefficient representation]

$$A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$$
$$B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_{n-1} x^{n-1}$$

Addition. $O(n)$ arithmetic operations.

$$A(x) + B(x) = (a_0 + b_0) + (a_1 + b_1)x + \ldots + (a_{n-1} + b_{n-1})x^{n-1}$$

Evaluation. $O(n)$ using Horner's method.

$$A(x) = a_0 + (x(a_1 + x(a_2 + \ldots + x(a_{n-2} + x(a_{n-1}))\ldots))$$

```
double val = 0.0;
for (int j = n-1; j >= 0; j--)
    val = a[j] + (x * val);
```

Multiplication (linear convolution). $O(n^2)$ using brute force.

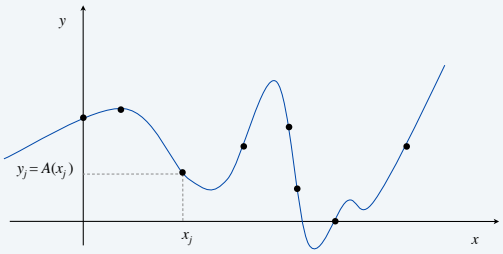
$$A(x) \times B(x) = \sum_{i=0}^{2n-2} c_i x^i \text{ where } c_i = \sum_{j=0}^{i} a_j b_{i-j}$$

slide credit: Kevin Wayne / Pearson

Polynomials: point-value representation

Fundamental theorem of algebra. A degree n univariate polynomial with complex coefficients has exactly n complex roots.

Corollary. A degree $n-1$ univariate polynomial $A(x)$ is uniquely specified by its evaluation at n distinct values of x.



slide credit: Kevin Wayne / Pearson

Polynomials: point-value representation

Univariate polynomial. [point-value representation]

$A(x)$: $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$
$B(x)$: $(x_0, z_0), \ldots, (x_{n-1}, z_{n-1})$

Addition. $O(n)$ arithmetic operations.

$$A(x) + B(x): \ (x_0, y_0 + z_0), \ldots, (x_{n-1}, y_{n-1} + z_{n-1})$$

Multiplication. $O(n)$, but represent $A(x)$ and $B(x)$ using $2n$ points.

$$A(x) \times B(x): \ (x_0, y_0 \times z_0), \ldots, (x_{2n-1}, y_{2n-1} \times z_{2n-1})$$

Evaluation. $O(n^2)$ using Lagrange's formula.

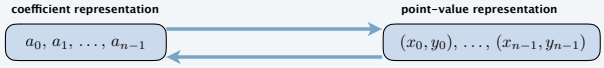
$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$ ← not used

Converting between two representations

Tradeoff. Either fast evaluation or fast multiplication. We want both!

representation	multiply	evaluate
coefficient	$O(n^2)$	$O(n)$
point-value	$O(n)$	$O(n^2)$

Goal. Efficient conversion between two representations ⇒ all ops fast.



Converting between two representations: brute force

Coefficient ⇒ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$.

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

Running time. $O(n^2)$ via matrix–vector multiply (or n Horner's).

Converting between two representations: brute force

Point-value ⇒ coefficient. Given n distinct points $x_0, \ldots, x_{n-1}$ and values $y_0, \ldots, y_{n-1}$, find unique polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, that has given values at given points.

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

Vandermonde matrix is invertible iff x_i distinct

Running time. $O(n^3)$ via Gaussian elimination.

or $O(n^{2.38})$ via fast matrix multiplication

Divide-and-conquer

Decimation in time. Divide into even- and odd- degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$.
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$.
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$.

Cooley-Tukey radix 2 FFT

Decimation in frequency. Divide into low- and high-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
- $A_{low}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$.
- $A_{high}(x) = a_4 + a_5 x + a_6 x^2 + a_7 x^3$.
- $A(x) = A_{low}(x) + x^4 A_{high}(x)$.

Sande-Tukey radix 2 FFT

Coefficient to point-value representation: intuition

Coefficient ⇒ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$. ← we get to choose which ones!

Divide. Break up polynomial into even- and odd-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$.
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$.
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$.
- $A(-x) = A_{even}(x^2) - x A_{odd}(x^2)$.

Intuition. Choose two points to be ± 1.

- $A(1) = A_{even}(1) + 1 A_{odd}(1)$.
- $A(-1) = A_{even}(1) - 1 A_{odd}(1)$.

← Can evaluate polynomial of degree n-1 at 2 points by evaluating two polynomials of degree ½n - 1 at only 1 point.

Coefficient to point-value representation: intuition

Coefficient ⇒ point-value. Given a polynomial $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$, evaluate it at n distinct points $x_0, \ldots, x_{n-1}$. ← we get to choose which ones!

Divide. Break up polynomial into even- and odd-degree terms.

- $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6 + a_7 x^7$.
- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3$.
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3$.
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$.
- $A(-x) = A_{even}(x^2) - x A_{odd}(x^2)$.

Intuition. Choose four complex points to be ± 1, $\pm i$.

- $A(1) = A_{even}(1) + 1 A_{odd}(1)$.
- $A(-1) = A_{even}(1) - 1 A_{odd}(1)$.
- $A(i) = A_{even}(-1) + i A_{odd}(-1)$.
- $A(-i) = A_{even}(-1) - i A_{odd}(-1)$.

← Can evaluate polynomial of degree n–1 at 4 points by evaluating two polynomials of degree ½n – 1 at only 2 points.

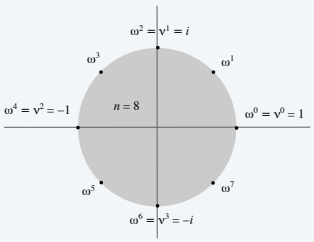
Roots of unity

Def. An n^{th} root of unity is a complex number x such that $x^n = 1$.

Fact. The n^{th} roots of unity are: $\omega^0, \omega^1, \ldots, \omega^{n-1}$ where $\omega = e^{2\pi i / n}$.

Pf. $(\omega^k)^n = (e^{2\pi i k / n})^n = (e^{\pi i})^{2k} = (-1)^{2k} = 1$.

Fact. The ½n^{th} roots of unity are: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$ where $\nu = \omega^2 = e^{4\pi i / n}$.



Fast Fourier transform

Goal. Evaluate a degree $n-1$ polynomial $A(x) = a_0 + \ldots + a_{n-1} x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

Divide. Break up polynomial into even- and odd-degree terms.

- $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{n/2-1}$.
- $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{n/2-1}$.
- $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$.
- $A(-x) = A_{even}(x^2) - x A_{odd}(x^2)$.

Conquer. Evaluate $A_{even}(x)$ and $A_{odd}(x)$ at the ½n^{th} roots of unity: $\nu^0, \nu^1, \ldots, \nu^{n/2-1}$.

Combine. ($\nu^k = (\omega^k)^2$)

- $y_k = A(\omega^k) = A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k)$, $0 \le k < n/2$.
- $y_{k+\frac{1}{2}n} = A(\omega^{k+\frac{1}{2}n}) = A_{even}(\nu^k) - \omega^k A_{odd}(\nu^k)$, $0 \le k < n/2$.

($A(-\omega^k)$)

FFT: implementation

Goal. Evaluate a degree $n-1$ polynomial $A(x) = a_0 + \ldots + a_{n-1} x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \ldots, \omega^{n-1}$.

- $y_k = A(\omega^k) = A_{even}(\nu^k) + \omega^k A_{odd}(\nu^k)$, $0 \le k < n/2$.
- $y_{k+\frac{1}{2}n} = A(\omega^{k+\frac{1}{2}n}) = A_{even}(\nu^k) - \omega^k A_{odd}(\nu^k)$, $0 \le k < n/2$.

```
FFT(n, a_0, a_1, a_2, ..., a_{n-1})

IF (n = 1) RETURN a_0.

(e_0, e_1, ..., e_{n/2-1}) ← FFT(n / 2, a_0, a_2, a_4, ..., a_{n-2}).
(d_0, d_1, ..., d_{n/2-1}) ← FFT(n / 2, a_1, a_3, a_5, ..., a_{n-1}).
                                                          ← 2 T(n / 2)
FOR k = 0 TO n / 2 - 1.
    ω^k ← e^{2πik/n}.
    y_k       ← e_k + ω^k d_k.
    y_{k+n/2} ← e_k - ω^k d_k.
                                                          ← Θ(n)
RETURN (y_0, y_1, y_2, ..., y_{n-1}).
```

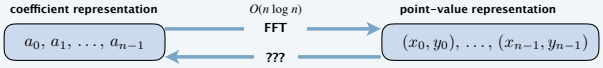

FFT: summary

Theorem. The FFT algorithm evaluates a degree $n-1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ arithmetic operations and $O(n)$ extra space.

assumes n is a power of 2

Pf.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$



Inverse FFT: summary

Theorem. The inverse FFT algorithm interpolates a degree $n-1$ polynomial at each of the n^{th} roots of unity in $O(n \log n)$ arithmetic operations.

assumes n is a power of 2

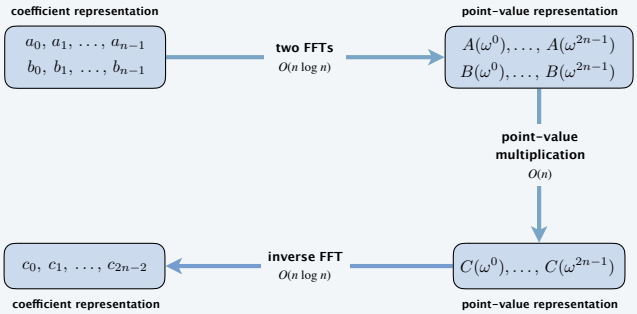
Corollary. Can convert between coefficient and point-value representations in $O(n \log n)$ arithmetic operations.

Polynomial multiplication

Theorem. Given two polynomials $A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ and $B(x) = b_0 + b_1 x + \ldots + b_{n-1} b^{n-1}$ of degree $n - 1$, can multiply them in $O(n \log n)$ arithmetic operations.

pad with 0s to make n a power of 2

Pf.



slide credit: Kevin Wayne / Pearson