The Design and Analysis of Algorithms

Lecture 11 Divide and Conque IV

Zhenbo Wang

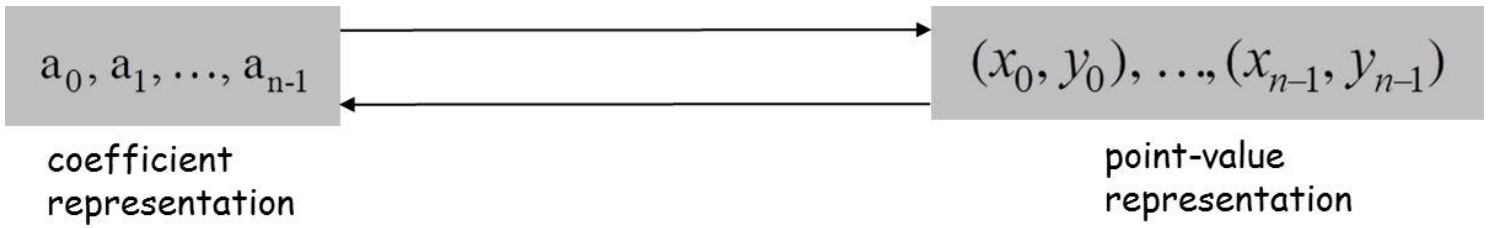
Department of Mathematical Sciences, Tsinghua University

Content

Converting Between Two Polynomial Representations

- Two representations of polynomials: coefficient and point-value.

Goal. Make evaluation or multiplication fast by efficiently converting between two representations.



Coefficient to Point-Value Representation: Intuition

- *Divide*. Break polynomial up into even and odd powers.

  $A(x) = A_{even}(x^2) + xA_{odd}(x^2)$.

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

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

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

  Can evaluate polynomial of degree $\leq n$ at 4 points by evaluating two polynomials of degree $\leq \frac{1}{2}n$ at 2 points.

Discrete Fourier Transform

- *Key idea*: choose $x_k = \omega^k$ where ω is principal n^{th} root of unity.

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \cdots & \omega^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

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: $\omega^0, \omega^1, \cdots, \omega^{n-1}$ where $\omega = e^{2\pi i/n}$.

Fact. The $\frac{1}{2}n^{th}$ roots of unity are: $v^0, v^1, \cdots, v^{n/2-1}$ where $v = e^{4\pi i/n}$.

Fast Fourier Transform

- *Goal*. Evaluate $A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$ at its n^{th} roots of unity: $\omega^0, \omega^1, \cdots, \omega^{n-1}$.
- *Divide*. Break polynomial up into even and odd powers

  $A_{even}(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n/2-2} x^{(n-1)/2}$.

  $A_{odd}(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n/2-1} x^{(n-1)/2}$.

  $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$.
- *Conquer*. Evaluate $A_{even}(x)$ and $A_{odd}(x)$ at the $\frac{1}{2} n^{th}$ roots of unity: $v^0, v^1, \cdots, v^{n/2-1}$, where $v^k = \omega^{2k}$
- *Combine*.

  $A(\omega^k) = A_{even}(v^k) + \omega^k A_{odd}(v^k)$, $0 \le k < n/2$.

  $A(\omega^{k+n/2}) = A_{even}(v^k) - \omega^k A_{odd}(v^k)$, $0 \le k \le n/2$.

FFT: Implementation

$FFT(n, a_0, a_1, \cdots, a_{n-1})$

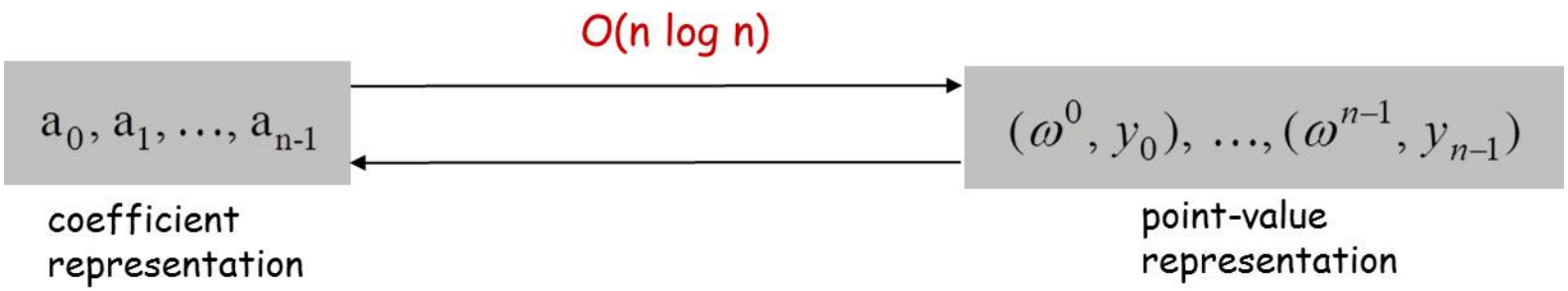
1: **if** $n = 1$ **then**
2: **return** a_0.
3: **end if**
4: $(e_0, e_1, \cdots, e_{n/2-1}) \leftarrow FFT(n/2, a_0, a_2, a_4, \cdots, a_{n-2})$.
5: $(d_0, d_1, \cdots, d_{n/2-1}) \leftarrow FFT(n/2, a_1, a_3, a_5, \cdots, a_{n-1})$.
6: **for** $k = 0$ to $n/2 - 1$ **do**
7: $\omega^k \leftarrow e^{2\pi ik/n}$.
8: $y_k \leftarrow e_k + \omega^k d_k$.
9: $y_{k+n/2} \leftarrow e_k - \omega^k d_k$.
10: **end for**
11: **return** $(y_0, y_1, \cdots, y_{n-1})$.

FFT: Summary

Theorem 1

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

Pf. $T(n) = 2T(n/2) + \theta(n) \Rightarrow T(n) = \theta(n \log n)$. □



Inverse discrete Fourier transform

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

$$\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \cdots & \omega^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Inverse discrete Fourier transform

Claim. Inverse of Fourier matrix F_n is given by following formula:

$$G_n = \frac{1}{n}\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)(n-1)} \end{pmatrix}$$

- *Consequence*. To compute inverse FFT, apply the same algorithm but use $\omega^{-1} = e^{-2\pi i/n}$ as principal n^{th} root of unity (and divide the result by n).

Inverse FFT: Implementation

INVERSEFFT$(n, y_0, y_1, \cdots, y_{n-1})$

```
1: if n = 1 then
2:     return y_0.
3: end if
4: (e_0, e_1, ..., e_{n/2-1}) ← INVERSEFFT(n/2, y_0, y_2, ..., y_{n-2}).
5: (d_0, d_1, ..., d_{n/2-1}) ← INVERSEFFT(n/2, y_1, y_3, ..., y_{n-1}).
6: for k = 0 to n/2 − 1 do
7:     ω^k ← e^{−2πik/n}.
8:     a_k ← e_k + ω^k d_k.
9:     a_{k+n/2} ← e_k − ω^k d_k.
10: end for
11: return (a_0, a_1, ..., a_{n-1})
```

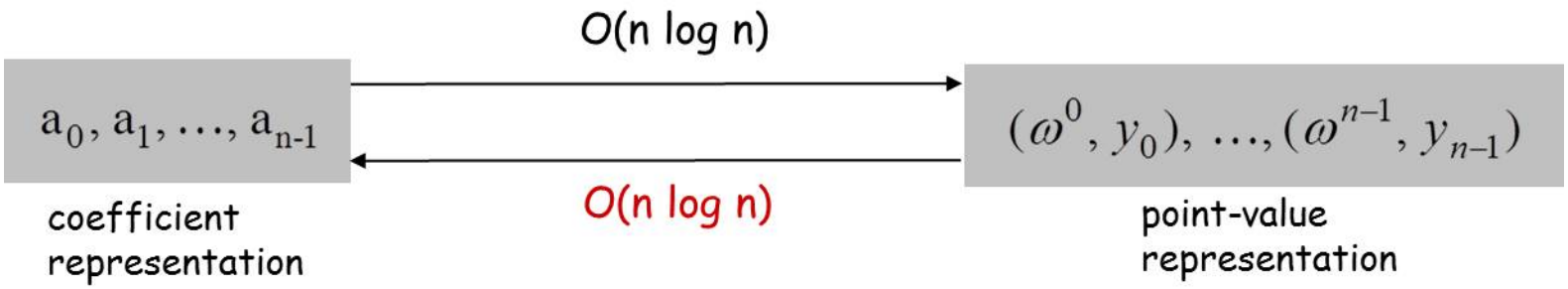
Remark:

Need to divide result by n.

Inverse FFT: Summary

Theorem 2

The inverse FFT algorithm interpolates a degree n – 1 polynomial given values at each of the n^{th} roots of unity in $O(n \log n)$ steps.



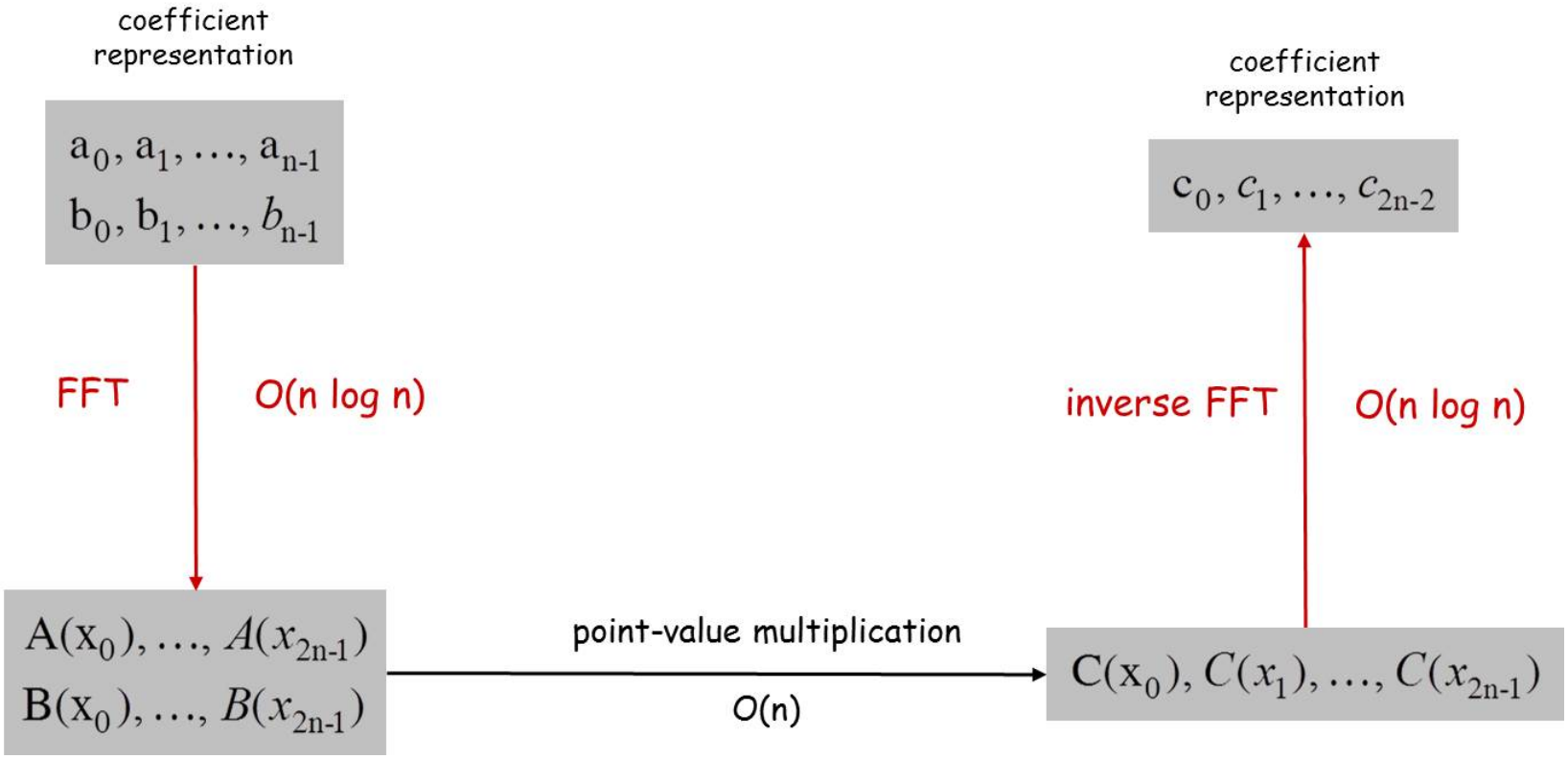
FFT

The FFT is one of the truly great computational developments of this 20*th* 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

Polynomial Multiplication

Theorem 3

Can multiply two degree n – 1 polynomials in $O(n \log n)$ steps.



Integer Multiplication

- *Integer multiplication*. Given two n-bit integers $a = a_{n-1} \cdots a_1 a_0$ and $b = b_{n-1} \cdots b_1 b_0$, compute $a \cdot b$.
- *Convolution algorithm*:

  Form two polynomials: $A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$, $B(x) = b_0 + b_1 x + \cdots + b_{n-1} x^{n-1}$.

  Note: $a = A(2)$, $b = B(2)$.

  Compute $C(x) = A(x) \cdot B(x)$.

  Evaluate $C(2) = a \cdot b$.

  Running time: $O(n \log n)$ complex arithmetic operations.

  $\Theta(n \log n \log \log n)$ bit operations. [Schönhage-Strassen 1971]

  $n \log n$ bit operations. [Harvey-van der Hoeven 2021]

Randomized Quicksort

- 3-way partition array so that:

  Pivot element p is in place.

  Smaller elements in left subarray L.

  Equal elements in middle subarray M.

  Larger elements in right subarray R.

  Recur in both left and right subarrays.

7	6	12	3	11	8	9	1	4	10	2

3	1	4	2	6	7	12	11	8	9	10
L				M	R					

Randomized Quicksort

RANDOMIZED – QUICKSORT(A)

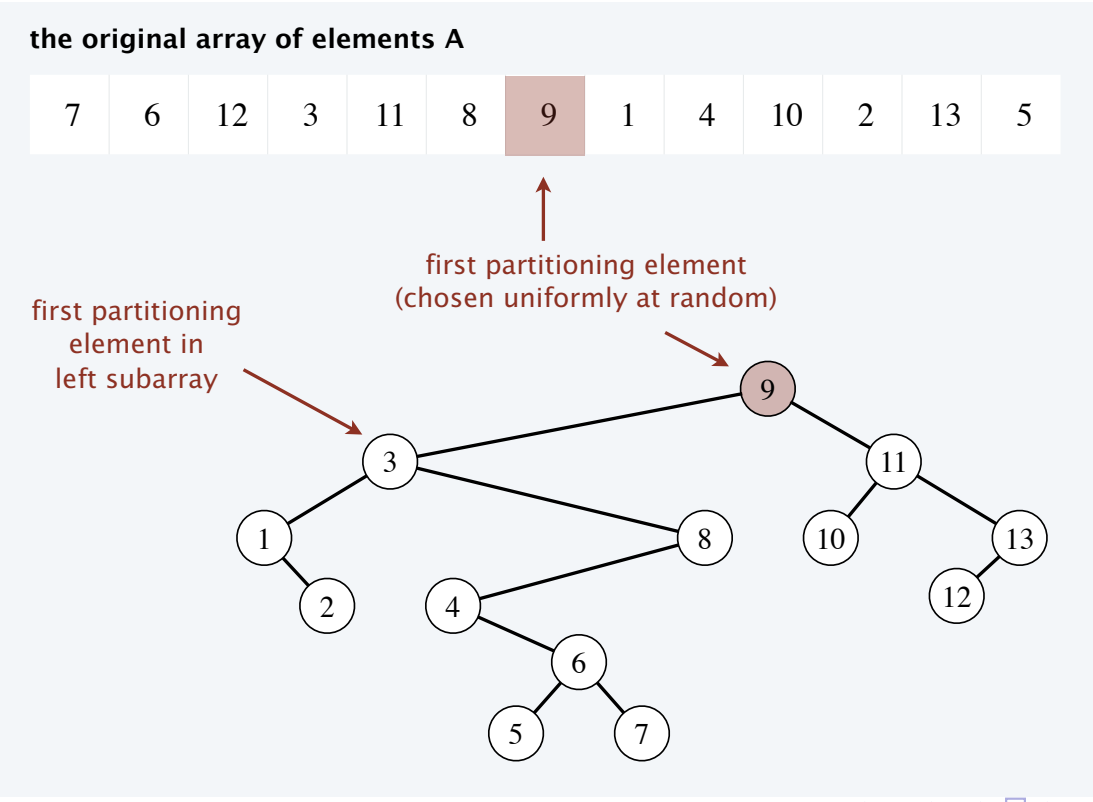
```
1: if list A has zero or one element then
2:     return
3: end if
4: Pick pivot p ∈ A uniformly at random.
5: (L, M, R) ← PARTITION-3-WAY (A, p).
6: RANDOMIZED – QUICKSORT(L).
7: RANDOMIZED – QUICKSORT(R).
```

Analysis of Randomized Quicksort

- *Proposition*. The expected number of compares to quicksort an array of n distinct elements is $O(n \log n)$.

Pf. Consider *BST* representation of partitioning elements.

An element is compared with only its ancestors and descendants.



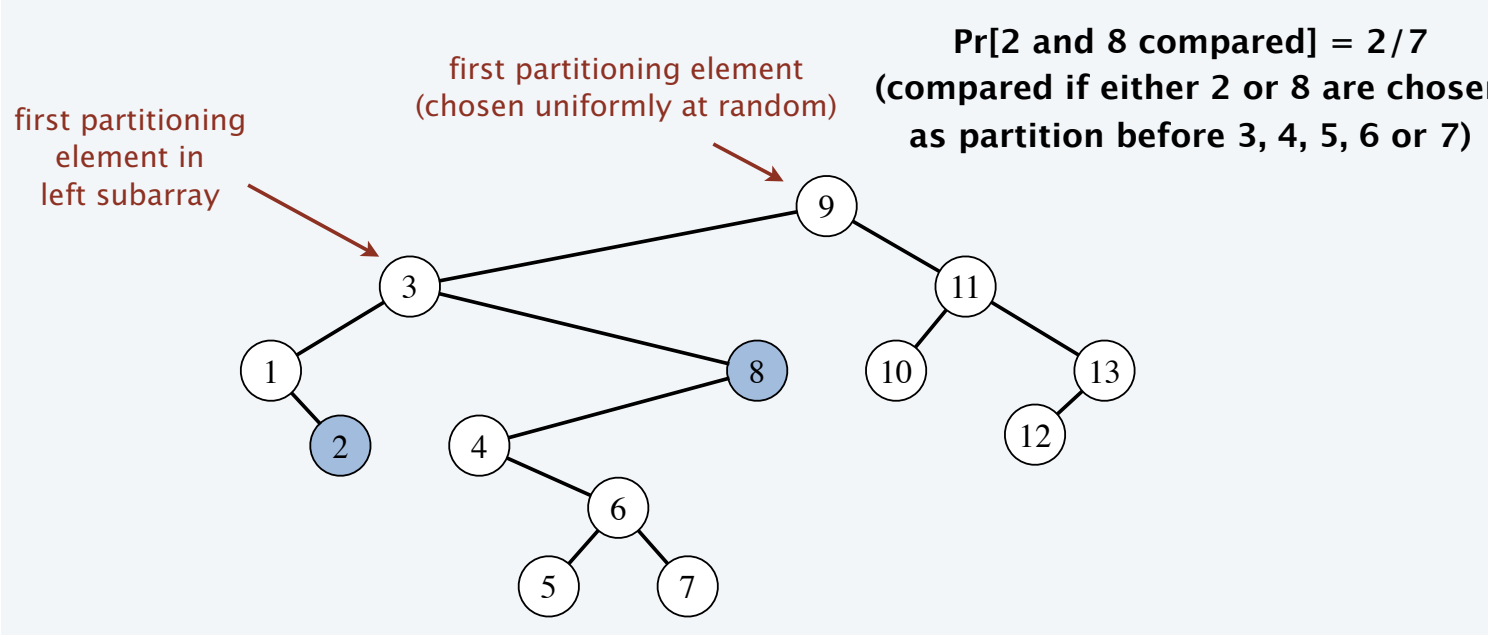
Analysis of Randomized Quicksort

$\Pr[\, a_i \text{ and } a_j \text{ are compared }] = 2/|j - i + 1|.$

Expected number of compares:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = 2 \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{1}{k} \leq 2n \sum_{k=1}^{n} \frac{1}{k} \approx 2n \ln n. \; \square$$

Remark. Number of compares only decreases if equal elements.



Homework

- Read Chapter 5 of the textbook.
- Exercises 5 & 6 in Chapter 5.