Lecture 3: Divide and Conquer II

Rafael Oliveira

University of Waterloo
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

September 14, 2023

Overview

- Polynomial Multiplication
  - Optional I: integer multiplication
  - Optional II: matrix multiplication
- Median Finding & Selection problem
- Acknowledgements

Polynomial Multiplication

- **Input:** two univariate polynomials

$$p(x) = \sum_{i=0}^{n} p_i x^i \text{ and } q(x) = \sum_{i=0}^{n} q_i x^i.$$

- **Output:** the product $a(x) := p(x) \cdot q(x)$. Output given by a list of coefficients $(a_0, \ldots, a_{2n})$
- Assume we are in the unit cost model, or word RAM where coefficients are integers in the range $[-w, w]$

Polynomial Multiplication

- **Input:** two univariate polynomials

$$p(x) = \sum_{i=0}^{n} p_i x^i \text{ and } q(x) = \sum_{i=0}^{n} q_i x^i.$$

- **Output:** the product $a(x) := p(x) \cdot q(x)$. Output given by a list of coefficients $(a_0, \ldots, a_{2n})$
- Naive algorithm:
  1. Note that $a_k = \sum_{i=0}^{k} p_i q_{k-i}$
  2. Hence, can compute all products $p_i q_j$, then compute the above sums

Polynomial Multiplication

- **Input:** two univariate polynomials

$$p(x) = \sum_{i=0}^{n} p_i x^i \text{ and } q(x) = \sum_{i=0}^{n} q_i x^i.$$

- **Output:** the product $a(x) := p(x) \cdot q(x)$. Output given by a list of coefficients $(a_0, \ldots, a_{2n})$
- Naive algorithm:
  1. Note that $a_k = \sum_{i=0}^{k} p_i q_{k-i}$
  2. Hence, can compute all products $p_i q_j$, then compute the above sums
- Runtime analysis: we compute $O(n^2)$ products, and perform $O(n^2)$ additions, hence, total runtime is $O(n^2)$

Polynomial Multiplication

- **Input:** two univariate polynomials

$$p(x) = \sum_{i=0}^{n} p_i x^i \text{ and } q(x) = \sum_{i=0}^{n} q_i x^i.$$

- **Output:** the product $a(x) := p(x) \cdot q(x)$. Output given by a list of coefficients $(a_0, \ldots, a_{2n})$
- Naive algorithm:
  1. Note that $a_k = \sum_{i=0}^{k} p_i q_{k-i}$
  2. Hence, can compute all products $p_i q_j$, then compute the above sums
- Runtime analysis: we compute $O(n^2)$ products, and perform $O(n^2)$ additions, hence, total runtime is $O(n^2)$
- can we do better?

Karatsuba's algorithm

1. write $p(x) = f_1(x) \cdot x^{n/2} + f_2(x)$ and $q(x) = g_1(x) \cdot x^{n/2} + g_2(x)$, where $\deg(f_i), \deg(g_i) \leq n/2$

Karatsuba's algorithm

1. write $p(x) = f_1(x) \cdot x^{n/2} + f_2(x)$ and $q(x) = g_1(x) \cdot x^{n/2} + g_2(x)$, where $\deg(f_i), \deg(g_i) \leq n/2$
2. note that

$$p(x) \cdot q(x) = f_1(x) \cdot g_1(x) \cdot x^n + [f_1(x) \cdot g_2(x) + f_2(x) \cdot g_1(x)] \cdot x^{\frac{n}{2}} + f_2(x) \cdot g_2(x)$$

Karatsuba's algorithm

1. write $p(x) = f_1(x) \cdot x^{n/2} + f_2(x)$ and $q(x) = g_1(x) \cdot x^{n/2} + g_2(x)$, where $\deg(f_i), \deg(g_i) \leq n/2$
2. note that

$$p(x) \cdot q(x) = f_1(x) \cdot g_1(x) \cdot x^n + [f_1(x) \cdot g_2(x) + f_2(x) \cdot g_1(x)] \cdot x^{\frac{n}{2}} + f_2(x) \cdot g_2(x)$$

3. Divide and conquer for the rescue!

$$T(n) = 4 \cdot T(n/2) + \gamma \cdot n$$

Karatsuba's algorithm

1. write $p(x) = f_1(x) \cdot x^{n/2} + f_2(x)$ and $q(x) = g_1(x) \cdot x^{n/2} + g_2(x)$, where $\deg(f_i), \deg(g_i) \leq n/2$
2. note that

$$p(x) \cdot q(x) = f_1(x) \cdot g_1(x) \cdot x^n + [f_1(x) \cdot g_2(x) + f_2(x) \cdot g_1(x)] \cdot x^{\frac{n}{2}} + f_2(x) \cdot g_2(x)$$

3. Divide and conquer for the rescue!

$$T(n) = 4 \cdot T(n/2) + \gamma \cdot n$$

Hmmmmm... this is giving me $O(n^2)$

Karatsuba's algorithm

1. write $p(x) = f_1(x) \cdot x^{n/2} + f_2(x)$ and $q(x) = g_1(x) \cdot x^{n/2} + g_2(x)$, where $\deg(f_i), \deg(g_i) \leq n/2$
2. note that

$$p(x) \cdot q(x) = f_1(x) \cdot g_1(x) \cdot x^n + [f_1(x) \cdot g_2(x) + f_2(x) \cdot g_1(x)] \cdot x^{\frac{n}{2}} + f_2(x) \cdot g_2(x)$$

3. Divide and conquer for the rescue!

$$T(n) = 4 \cdot T(n/2) + \gamma \cdot n$$

4. Can we reduce the number of subproblems?

Need to reduce number of multiplications!

Reducing number of multiplications

- Want to compute

$$p(x)\cdot q(x) = f_1(x)\cdot g_1(x)\cdot x^n + [f_1(x)\cdot g_2(x) + f_2(x)\cdot g_1(x)]\cdot x^{\frac{n}{2}} + f_2(x)\cdot g_2(x)$$

So need to compute the polynomials:

$$f_1(x)\cdot g_1(x), \quad f_1(x)\cdot g_2(x) + f_2(x)\cdot g_1(x), \quad f_2(x)\cdot g_2(x)$$

with less than 4 multiplications.

Reducing number of multiplications

- Want to compute

$$p(x)\cdot q(x) = f_1(x)\cdot g_1(x)\cdot x^n + [f_1(x)\cdot g_2(x) + f_2(x)\cdot g_1(x)]\cdot x^{\frac{n}{2}} + f_2(x)\cdot g_2(x)$$

So need to compute the polynomials:

$$f_1(x)\cdot g_1(x), \quad f_1(x)\cdot g_2(x) + f_2(x)\cdot g_1(x), \quad f_2(x)\cdot g_2(x)$$

with less than 4 multiplications.

- with the product

$$A(x) := (f_1(x) + f_2(x))\cdot(g_1(x) + g_2(x))$$

we are almost there!

Reducing number of multiplications

- Want to compute

$$p(x) \cdot q(x) = f_1(x) \cdot g_1(x) \cdot x^n + [f_1(x) \cdot g_2(x) + f_2(x) \cdot g_1(x)] \cdot x^{\frac{n}{2}} + f_2(x) \cdot g_2(x)$$

  So need to compute the polynomials:

$$f_1(x) \cdot g_1(x), \quad f_1(x) \cdot g_2(x) + f_2(x) \cdot g_1(x), \quad f_2(x) \cdot g_2(x)$$

  with less than 4 multiplications.
- with the product

$$A(x) := (f_1(x) + f_2(x)) \cdot (g_1(x) + g_2(x))$$

  we are almost there!
- Using the products

$$B(x) := f_1(x) \cdot g_1(x), \text{ and } C(x) := f_2(x) \cdot g_2(x)$$

  can compute the 3 above terms!

Recurrence

- Thus, we have the following recurrence:

$$T(n) = 3T(n/2) + \gamma n$$

which yields

$$T(n) = O(n^{\log 3}) = o(n^{1.59}).$$

Recurrence

- Thus, we have the following recurrence:

$$T(n) = 3T(n/2) + \gamma n$$

which yields

$$T(n) = O(n^{\log 3}) = o(n^{1.59}).$$

If you want to learn faster algorithms (and other cool symbolic algorithms), consider taking CS 487.

- Polynomial Multiplication
  - Optional I: integer multiplication
  - Optional II: matrix multiplication
- Median Finding & Selection problem
- Acknowledgements

Integer multiplication

- **Input:** two n-bit numbers $a := a_1 a_2 \cdots a_n$ and $b := b_1 b_2 \cdots b_n$
- **Output:** $a \cdot b$
- Bit complexity model!

Integer multiplication

- **Input:** two n-bit numbers $a := a_1 a_2 \cdots a_n$ and $b := b_1 b_2 \cdots b_n$
- **Output:** $a \cdot b$
- Naive algorithm:

  similar to polynomial multiplication, takes $\Theta(n^2)$ time

Integer multiplication

- **Input:** two n-bit numbers $a := a_1 a_2 \cdots a_n$ and $b := b_1 b_2 \cdots b_n$
- **Output:** $a \cdot b$
- Naive algorithm:

  similar to polynomial multiplication, takes $\Theta(n^2)$ time
- Can we do better?

Integer multiplication

- **Input:** two n-bit numbers $a := a_1 a_2 \cdots a_n$ and $b := b_1 b_2 \cdots b_n$
- **Output:** $a \cdot b$
- Naive algorithm:

  similar to polynomial multiplication, takes $\Theta(n^2)$ time
- Same strategy to Karatsuba's algorithm!
  Write $a = x_1 x_2$ and $b = y_1 y_2$. Note that

$$a \cdot b = x_1 \cdot y_1 \cdot 2^n + (x_1 \cdot y_2 + x_2 \cdot y_1) \cdot 2^{n/2} + x_2 \cdot y_2$$

Integer multiplication

- **Input:** two n-bit numbers $a := a_1 a_2 \cdots a_n$ and $b := b_1 b_2 \cdots b_n$
- **Output:** $a \cdot b$
- Naive algorithm:

  similar to polynomial multiplication, takes $\Theta(n^2)$ time
- Same strategy to Karatsuba's algorithm!
  Write $a = x_1 x_2$ and $b = y_1 y_2$. Note that

  $$a \cdot b = x_1 \cdot y_1 \cdot 2^n + (x_1 \cdot y_2 + x_2 \cdot y_1) \cdot 2^{n/2} + x_2 \cdot y_2$$
- Same recurrence as Karatsuba's!

  Thus $T(n) = O(n^{\log 3})$.

Integer multiplication

- **Input:** two n-bit numbers $a := a_1 a_2 \cdots a_n$ and $b := b_1 b_2 \cdots b_n$
- **Output:** $a \cdot b$
- Naive algorithm:

  similar to polynomial multiplication, takes $\Theta(n^2)$ time
- Same strategy to Karatsuba's algorithm!
  Write $a = x_1 x_2$ and $b = y_1 y_2$. Note that

  $$a \cdot b = x_1 \cdot y_1 \cdot 2^n + (x_1 \cdot y_2 + x_2 \cdot y_1) \cdot 2^{n/2} + x_2 \cdot y_2$$
- Same recurrence as Karatsuba's!

  Thus $T(n) = O(n^{\log 3})$.
- [Harvey, van der Hoeven 2019] algorithm for integer multiplication with $O(n \log n)$ runtime!

- Polynomial Multiplication
  - Optional I: integer multiplication
  - Optional II: matrix multiplication
- Median Finding & Selection problem
- Acknowledgements

Matrix Multiplication

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$

Matrix Multiplication

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$
- Naive algorithm:

Compute n matrix vector multiplications.

Matrix Multiplication

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$
- Naive algorithm:

  Compute n matrix vector multiplications.
- Running time: $O(n^3)$

  Can we do better?

Matrix Multiplication

- **Input:** matrices $A, B \in \mathbb{F}^{n \times n}$
- **Output:** product $C = AB$
- Naive algorithm:

Compute n matrix vector multiplications.

- Running time: $O(n^3)$

Can we do better?

- Strassen 1969: YES!
- Idea: divide matrix into blocks, and *reduce number of multiplications* needed!

Similar in spirit as Karatsuba's algorithm for polynomial multiplication!

Strassen's Algorithm

- Suppose that $n = 2^k$
- Let $A, B, C \in \mathbb{F}^{n \times n}$ such that $C = AB$. Divide them into blocks of size $n/2$:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Strassen's Algorithm

- Suppose that $n = 2^k$
- Let $A, B, C \in \mathbb{F}^{n \times n}$ such that $C = AB$. Divide them into blocks of size $n/2$:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

Strassen's Algorithm

- Suppose that $n = 2^k$
- Let $A, B, C \in \mathbb{F}^{n\times n}$ such that $C = AB$. Divide them into blocks of size $n/2$:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1T_1,\ P_6 = S_2T_2,\ P_7 = S_3T_3$$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1T_1,\ P_6 = S_2T_2,\ P_7 = S_3T_3$$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22}, \; S_2 = S_1 - A_{11}, \; S_3 = A_{11} - A_{21}, \; S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11}, \; T_2 = B_{22} - T_1, \; T_3 = B_{22} - B_{12}, \; T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11}, \; P_2 = A_{12}B_{21}, \; P_3 = S_4 B_{22}, \; P_4 = A_{22}T_4$$

$$P_5 = S_1 T_1, \; P_6 = S_2 T_2, \; P_7 = S_3 T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1T_1,\ P_6 = S_2T_2,\ P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1T_1,\ P_6 = S_2T_2,\ P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$
- $C_{21} = A_{21}B_{11} + A_{22}B_{21} = P_1 - P_4 + P_6 + P_7$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1T_1,\ P_6 = S_2T_2,\ P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$
- $C_{21} = A_{21}B_{11} + A_{22}B_{21} = P_1 - P_4 + P_6 + P_7$
- $C_{22} = A_{21}B_{12} + A_{22}B_{22} = P_1 + P_5 + P_6 + P_7$

Strassen's Algorithm

- Define following matrices:

$$S_1 = A_{21} + A_{22},\ S_2 = S_1 - A_{11},\ S_3 = A_{11} - A_{21},\ S_4 = A_{12} - S_2$$

$$T_1 = B_{12} - B_{11},\ T_2 = B_{22} - T_1,\ T_3 = B_{22} - B_{12},\ T_4 = T_2 - B_{21}$$

- Compute the following 7 products:

$$P_1 = A_{11}B_{11},\ P_2 = A_{12}B_{21},\ P_3 = S_4B_{22},\ P_4 = A_{22}T_4$$

$$P_5 = S_1T_1,\ P_6 = S_2T_2,\ P_7 = S_3T_3$$

- $C_{11} = A_{11}B_{11} + A_{12}B_{21} = P_1 + P_2$
- $C_{12} = A_{11}B_{12} + A_{12}B_{22} = P_1 + P_3 + P_5 + P_6$
- $C_{21} = A_{21}B_{11} + A_{22}B_{21} = P_1 - P_4 + P_6 + P_7$
- $C_{22} = A_{21}B_{12} + A_{22}B_{22} = P_1 + P_5 + P_6 + P_7$
- Correctness follows from the computations

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
  1. 8 additions S_i, T_i's
  2. 7 multiplications P_i's
  3. 10 additions C_{ij}'s

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
  1. 8 additions — S_i, T_i's
  2. 7 multiplications — P_i's
  3. 10 additions — C_{ij}'s
- Recurrence:

$$MM(n) \leq 7 \cdot MM(n/2) + 18 \cdot c \cdot (n/2)^2$$

Analysis of Strassen's Algorithm

- To compute $AB = C$ we used:
  1. 8 additions — S_i, T_i's
  2. 7 multiplications — P_i's
  3. 10 additions — C_{ij}'s
- Recurrence:

$$MM(n) \leq 7 \cdot MM(n/2) + 18 \cdot c \cdot (n/2)^2$$

- Master theorem: $MM(n) = O(n^{\log 7}) \approx O(n^{2.807})$

Can we do better?

- There has been phenomenal progress in this question, spurred by work of Coppersmith and Vinograd.
- By following their approach, the current record for matrix multiplication is roughly $O(n^{2.37})$

Open problem: can you do better?

- Polynomial Multiplication
  - Optional I: integer multiplication
  - Optional II: matrix multiplication

- Median Finding & Selection problem

- Acknowledgements

Median Finding

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$
- **Output:** median of these numbers
- Word RAM model!

Median Finding

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$
- **Output:** median of these numbers
- Naive algorithm: sort the numbers, then output the middle element.

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

Median Finding

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$
- **Output:** median of these numbers
- Naive algorithm: sort the numbers, then output the middle element.

  Running time: $O(n \log n)$.
- Can we do better?

Median Finding

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$
- **Output:** median of these numbers
- Naive algorithm: sort the numbers, then output the middle element.

  Running time: $O(n \log n)$.
- Can we do better?
- Turns out we can solve this problem in $\Theta(n)$ time!

  Divide and conquer!

Median Finding

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$
- **Output:** median of these numbers
- Naive algorithm: sort the numbers, then output the middle element.

  Running time: $O(n \log n)$.
- Can we do better?
- Turns out we can solve this problem in $\Theta(n)$ time!

  Divide and conquer!
- hmmmmm... but how can we divide?

  Subproblem will not be the median problem!

Median Finding

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$
- **Output:** median of these numbers
- Naive algorithm: sort the numbers, then output the middle element.

  Running time: $O(n \log n)$.
- Can we do better?
- Turns out we can solve this problem in $\Theta(n)$ time!

  Divide and conquer!
- hmmmmm... but how can we divide?

  Subproblem will not be the median problem!
- Idea: generalize our problem a little bit, to make it more flexible.

Selection Problem

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$, integer $k \in [n]$
- **Output:** k^{th} smallest element of A
- (Still) Word RAM model!

Selection Problem

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$, integer $k \in [n]$
- **Output:** k^{th} smallest element of A
- To divide-and-conquer, can select an element α of the array (the pivot), and with a linear scan break A into A_L, A_R, where
  $$\begin{cases} a_i \in A_L \text{ iff } a_i < \alpha \\ a_i \in A_R \text{ iff } a_i > \alpha \end{cases}$$

Selection Problem

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$, integer $k \in [n]$
- **Output:** k^{th} smallest element of A
- To divide-and-conquer, can select an element α of the array (the pivot), and with a linear scan break A into A_L, A_R, where
  $$\begin{cases} a_i \in A_L \text{ iff } a_i < \alpha \\ a_i \in A_R \text{ iff } a_i > \alpha \end{cases}$$
- Question: how to find a good pivot?
  If $\text{rank}(\alpha) = r$ (i.e. α is the r^{th} smallest element), then subproblems of size: $r - 1$ and $n - r$

  To make progress on subproblem sizes, need $r = \Theta(n)$.

Selection Problem

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$, integer $k \in [n]$
- **Output:** k^{th} smallest element of A
- To divide-and-conquer, can select an element α of the array (the pivot), and with a linear scan break A into A_L, A_R, where
$$\begin{cases} a_i \in A_L \text{ iff } a_i < \alpha \\ a_i \in A_R \text{ iff } a_i > \alpha \end{cases}$$
- Question: how to find a good pivot?
If $\text{rank}(\alpha) = r$ (i.e. α is the r^{th} smallest element), then subproblems of size: $r - 1$ and $n - r$

To make progress on subproblem sizes, need $r = \Theta(n)$.
- For instance, if $n/4 \leq r \leq 3n/4$, we have:

$$T(n) \leq T(3n/4) + P(n) + \gamma \cdot n$$

where $P(n) =$ time to find a good pivot and $T(n) =$ time to find k^{th} element

Selection Problem

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$, integer $k \in [n]$
- **Output:** k^{th} smallest element of A
- To divide-and-conquer, can select an element α of the array (the pivot), and with a linear scan break A into A_L, A_R, where
$$\begin{cases} a_i \in A_L \text{ iff } a_i < \alpha \\ a_i \in A_R \text{ iff } a_i > \alpha \end{cases}$$
- Question: how to find a good pivot?
If $\text{rank}(\alpha) = r$ (i.e. α is the r^{th} smallest element), then subproblems of size: $r - 1$ and $n - r$

  To make progress on subproblem sizes, need $r = \Theta(n)$.
- For instance, if $n/4 \leq r \leq 3n/4$, we have:
$$T(n) \leq T(3n/4) + P(n) + \gamma \cdot n$$
where $P(n) =$ time to find a good pivot and $T(n) =$ time to find k^{th} element
- So if we could show that $P(n) = O(n)$ we would be done.

Finding good pivot: median of medians

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$
- **Output:** element a_i such that $3n/10 \leq \text{rank}(a_i) \leq 7n/10$

Finding good pivot: median of medians

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$
- **Output:** element a_i such that $3n/10 \leq \text{rank}(a_i) \leq 7n/10$
- Median of medians algorithm:
    1. divide A into $n/5$ arrays $A_1, \ldots, A_{n/5}$ each of size 5
    2. let $\alpha_1, \alpha_2, \ldots, \alpha_{n/5}$ be the medians of $A_1, \ldots, A_{n/5}$, respectively
    3. return $\alpha := \text{median}(\alpha_1, \ldots, \alpha_{n/5})$

Finding good pivot: median of medians

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$
- **Output:** element a_i such that $3n/10 \leq \text{rank}(a_i) \leq 7n/10$
- Median of medians algorithm:
  1. divide A into $n/5$ arrays $A_1, \ldots, A_{n/5}$ each of size 5
  2. let $\alpha_1, \alpha_2, \ldots, \alpha_{n/5}$ be the medians of $A_1, \ldots, A_{n/5}$, respectively
  3. return $\alpha := \text{median}(\alpha_1, \ldots, \alpha_{n/5})$
- Running time based on recurrence:

$$P(n) = T(n/5) + \delta \cdot n$$

Master theorem: $O(n)$

Finding good pivot: median of medians

- **Input:** array with distinct integers $A = [a_1, \ldots, a_n]$
- **Output:** element a_i such that $3n/10 \leq \text{rank}(a_i) \leq 7n/10$
- Median of medians algorithm:
  1. divide A into $n/5$ arrays $A_1, \ldots, A_{n/5}$ each of size 5
  2. let $\alpha_1, \alpha_2, \ldots, \alpha_{n/5}$ be the medians of $A_1, \ldots, A_{n/5}$, respectively
  3. return $\alpha := \text{median}(\alpha_1, \ldots, \alpha_{n/5})$
- Running time based on recurrence:

$$P(n) = T(n/5) + \delta \cdot n$$

  Master theorem: $O(n)$
- Rank of output: note that

$$3 \cdot \frac{n}{10} \leq \text{rank}(\alpha) \leq 7 \cdot \frac{n}{10}$$

  as α larger than median of $n/10$ of the arrays, and smaller than median of $n/10$ of the arrays

Back to selection problem

- Now we can find an element $\alpha \in A$ with $3n/10 \leq \text{rank}(\alpha) \leq 7n/10$ in time $\delta \cdot n$

Back to selection problem

- Now we can find an element $\alpha \in A$ with $3n/10 \leq \text{rank}(\alpha) \leq 7n/10$ in time $\delta \cdot n$
- Our recursion for selection problem is then:

$$T(n) \leq T(7n/10) + P(n) + \gamma \cdot n$$

Back to selection problem

- Now we can find an element $\alpha \in A$ with $3n/10 \leq \text{rank}(\alpha) \leq 7n/10$ in time $\delta \cdot n$
- Our recursion for selection problem is then:

$$T(n) \leq T(7n/10) + P(n) + \gamma \cdot n$$

- But we saw that

$$P(n) \leq T(n/5) + \delta \cdot n$$

Back to selection problem

- Now we can find an element $\alpha \in A$ with $3n/10 \leq \text{rank}(\alpha) \leq 7n/10$ in time $\delta \cdot n$
- Our recursion for selection problem is then:

$$T(n) \leq T(7n/10) + P(n) + \gamma \cdot n$$

- But we saw that

$$P(n) \leq T(n/5) + \delta \cdot n$$

- Thus, we have:

$$T(n) \leq T(7n/10) + T(n/5) + (\gamma + \delta) \cdot n$$

Back to selection problem

- Now we can find an element $\alpha \in A$ with $3n/10 \leq \text{rank}(\alpha) \leq 7n/10$ in time $\delta \cdot n$
- Our recursion for selection problem is then:

$$T(n) \leq T(7n/10) + P(n) + \gamma \cdot n$$

- But we saw that

$$P(n) \leq T(n/5) + \delta \cdot n$$

- Thus, we have:

$$T(n) \leq T(7n/10) + T(n/5) + (\gamma + \delta) \cdot n$$

- Same analysis as recurrence from previous lecture, yields

$$T(n) = \Theta(n).$$

Acknowledgement

- Based on Prof. Lau's lectures 3 and 4

  https://cs.uwaterloo.ca/~lapchi/cs341/notes/L03.pdf
  https://cs.uwaterloo.ca/~lapchi/cs341/notes/L04.pdf

References I

Cormen, Thomas and Leiserson, Charles and Rivest, Ronald and Stein, Clifford. (2009)
Introduction to Algorithms, third edition.
MIT Press

Harvey, David and van der Hoeven, Joris (2019)
Integer multiplication in time $O(n \log n)$
Annals of Mathematics