NUS | Computing
National University of Singapore

CS3230
eric_han@nus.edu.sg
https://eric-han.com

Computer Science

T05 – Week 6

D&C, Sorting, and Average-Case Analysis

CS3230 – Design and Analysis of Algorithms

Recap from T04

Loop Invariant

- GeeksforGeeks – Loop Invariants
- StackExchange – Tips for Constructing Basic Loop Invariants

Induction

- https://leetcode.com/problem-list/recursion/
- Brilliant – Writing a proof by Induction
- Khan Academy – Verifying an algorithm (also invariant)

D&C

In general, this requires training your thinking processes (which is v hard):

- https://leetcode.com/problem-list/divide-and-conquer/
- T04 Q5: Split by the largest direction (row or column).

T04 Q5: How to achieve $\Theta(n)$

Algorithm

1. Split the matrix always in the larger (width or height)
2. Same algorithm as before.

Proof

Assuming $m > n$, and also vice versa for $n > m$:

$$\begin{aligned} T(m,n) &= T\left(\frac{m}{2}, n\right) + \Theta(n) \\ &= \left[T\left(\frac{m}{2}, \frac{n}{2}\right) + \Theta\left(\frac{m}{2}\right)\right] + \Theta(n) \end{aligned}$$

Since the recurrence reduces by $1/2$ in 2 iterations, we obtain[1] $T(n) = T(n/2) + \Theta(n)$.
Since $a = 1$, $b = 2$, $d = \log_2 1 = 0$, and $f(n) = n \in \Omega(n^{d+\epsilon})$ for any $\epsilon > 0$. Furthermore, the regularity condition is satisfied, as: $1 \cdot f(n/2) = \frac{n}{2} \leq cf(n) \quad$ for $c = \frac{1}{2} < 1$.
Thus, by Case 3 of the Master Theorem: $T(n) = \Theta(n)$.

[1] You may work it out exactly, but... Lazy.

A decision tree consists of:

- **Vertices (Internal):** A comparison
- **Branches:** Outcome of the comparison
- **Leaves:** Output/decision for the input

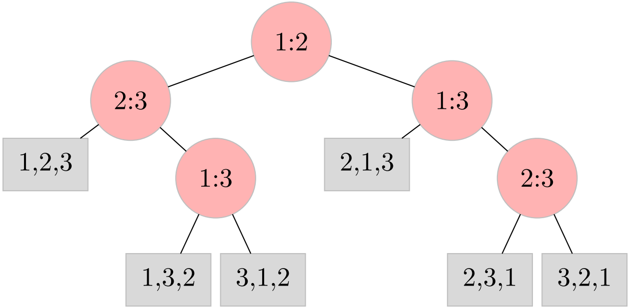

Figure 1: Worst case runtime is the height of the decision tree.

Polynomial Multiplication (Degree n)

Given two polynomials:

$$A(x) = a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0$$

$$B(x) = b_n x^n + \cdots + b_2 x^2 + b_1 x + b_0$$

Their product:

$$C(x) = A(x) \times B(x) = c_{2n} x^{2n} + \cdots + c_2 x^2 + c_1 x + c_0$$

where all coefficients a_i, b_i, c_i are integers.

Brute Force Approach: $O(n^2)$ **Complexity**

$$\forall_i \in [2n..0], \quad c_i = \sum_{j=0}^{n} a_j \cdot b_{i-j}, \quad \text{where } 0 \leq i - j \leq n.$$

Assuming integer addition and multiplication take $O(1)$ time, this approach runs in $O(n^2)$.

Brute Force Approach (code)

```python
def poly_mult_bruteforce(A, B):
    # A: Coeff [a0, a1, ..., a_n] for A(x) = a0 + a1*x + ... + a_n*x^n.
    # B: Coeff [b0, b1, ..., b_n] for B(x) = b0 + b1*x + ... + b_n*x^n.

    n = len(A) - 1  # Degree of the polynomial A or B
    result = [0] * (2 * n + 1) # Result in (2n + 1) coefficients

    # Compute each coefficient c_i for the product polynomial C(x)
    for i in range(2 * n + 1):
        for j in range(max(0, i - n), min(i, n) + 1):
            result[i] += A[j] * B[i - j]
    return result
```

Question 1

Let $x = 10$ to visualize this as base-10 multiplication with $n = 2$.

Given Polynomials

$$A(10) = 352 = 3 \cdot 10^2 + 5 \cdot 10 + 2, \qquad \text{i.e. } a_2 = 3, \quad a_1 = 5, \quad a_0 = 2,$$
$$B(10) = 221 = 2 \cdot 10^2 + 2 \cdot 10 + 1, \qquad \text{i.e. } b_2 = 2, \quad b_1 = 2, \quad b_0 = 1.$$

Compute the coefficients of $C(10) = A(10) \times B(10) = 77,792$ using the $O(n^2)$ algorithm.

Answer

Using the $O(n^2)$ algorithm, we compute:

$$c_4 = a_2 \cdot b_{4-2} = a_2 \cdot b_2 = 3 \cdot 2 = 6.$$

$$\begin{aligned} c_3 &= a_1 \cdot b_{3-1} + a_2 \cdot b_{3-2} = a_1 \cdot b_2 + a_2 \cdot b_1 \\ &= 5 \cdot 2 + 3 \cdot 2 = 10 + 6 = 16. \end{aligned}$$

$$\begin{aligned} c_2 &= a_0 \cdot b_{2-0} + a_1 \cdot b_{2-1} + a_2 \cdot b_{2-2} \\ &= a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0 \\ &= 2 \cdot 2 + 5 \cdot 2 + 3 \cdot 1 = 4 + 10 + 3 = 17. \end{aligned}$$

$$\begin{aligned} c_1 &= a_0 \cdot b_{1-0} + a_1 \cdot b_{1-1} = a_0 \cdot b_1 + a_1 \cdot b_0 \\ &= 2 \cdot 2 + 5 \cdot 1 = 4 + 5 = 9. \end{aligned}$$

$$c_0 = a_0 \cdot b_{0-0} = a_0 \cdot b_0 = 2 \cdot 1 = 2.$$

Hence, $C(10) = 6 \cdot 10^4 + 16 \cdot 10^3 + 17 \cdot 10^2 + 9 \cdot 10 + 2 = 77\,792.$

Question 2

D&C Algorithm

1. Rewrite the polynomials:

$$A(x) = x^{\frac{n}{2}} \cdot A_1(x) + A_2(x), \quad B(x) = x^{\frac{n}{2}} \cdot B_1(x) + B_2(x)$$

   where $A_1(x), A_2(x), B_1(x), B_2(x)$ are polynomials of degree at most $\frac{n}{2}$.

2. Compute four smaller polynomial multiplications:

$$A_1(x) \times B_1(x), \quad A_1(x) \times B_2(x), \quad A_2(x) \times B_1(x), \quad A_2(x) \times B_2(x)$$

3. Compute the final polynomial:

$$C(x) = x^n \cdot [A_1(x) \times B_1(x)] + x^{\frac{n}{2}} \cdot [A_1(x) \times B_2(x) + A_2(x) \times B_1(x)] + A_2(x) \times B_2(x)$$

Use this algorithm to multiply two polynomials of degree $n = 2$.

Answer

Given: $A(10) = 352 = 10 \cdot (3 \cdot 10 + 5) + 2, \qquad B(10) = 221 = 10 \cdot (2 \cdot 10 + 2) + 1$

Computing Partial Products

$$\begin{aligned} A_1(10) \times B_1(10) &= (3 \cdot 10 + 5) \times (2 \cdot 10 + 2) \\ &= 6 \cdot 10^2 + 16 \cdot 10 + 10 \\ &= 600 + 160 + 10 = 770. \end{aligned}$$

$$\begin{aligned} A_1(10) \times B_2(10) &= (3 \cdot 10 + 5) \times 1 \\ &= 3 \cdot 10 + 5 = 35. \end{aligned}$$

$$\begin{aligned} A_2(10) \times B_1(10) &= 2 \times (2 \cdot 10 + 2) \\ &= 4 \cdot 10 + 4 = 44. \end{aligned}$$

$$A_2(10) \times B_2(10) = 2 \times 1 = 2.$$

Compute the final polynomial

$$\begin{aligned}
C(10) &= 10^2 \cdot (A_1(10) \times B_1(10)) \\
&\quad + 10 \cdot (A_1(10) \times B_2(10) + A_2(10) \times B_1(10)) \\
&\quad + A_2(10) \times B_2(10) \\
&= 10^2 \cdot \left(6 \cdot 10^2 + 16 \cdot 10 + 10\right) \\
&\quad + 10 \cdot \left(3 \cdot 10 + 5 + 4 \cdot 10 + 4\right) + 2 \\
&= 6 \cdot 10^4 + 16 \cdot 10^3 + 10 \cdot 10^2 + 7 \cdot 10^2 + 9 \cdot 10 + 2 \\
&= 6 \cdot 10^4 + 16 \cdot 10^3 + 17 \cdot 10^2 + 9 \cdot 10 + 2 \\
&= 60\,000 + 16\,000 + 1\,700 + 90 + 2 \\
&= 77\,992
\end{aligned}$$

Question 3 [P1]

What is the time complexity of that recursive D&C algorithm?

Question 3 [P1]

What is the time complexity of that recursive D&C algorithm?

Answer

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

- There are **4 multiplications** of polynomials of degree $\frac{n}{2}$.
- **Combining results** requires $O(n)$ work.

Since $a = 4$, $b = 2$, and $d = \log_2 4 = 2$, and $f(n) \in O(n^{d-\epsilon})$ for some $\epsilon > 0$, by Case 1 of the Master Theorem, we get:

$$T(n) \in \Theta(n^d) = \Theta(n^2).$$

Thus, this is no better than naive polynomial multiplication.

Karatsuba Algorithm

1. Compute two smaller polynomial multiplications:
$$A_1(x) \times B_1(x), \quad A_2(x) \times B_2(x).$$
2. Compute one multiplication with two additions:
$$[A_1(x) + A_2(x)] \times [B_1(x) + B_2(x)].$$
3. Apply the identity, two subtractions
$$\begin{aligned} A_1(x) \times B_2(x) + A_2(x) \times B_1(x) = {} & [A_1(x) + A_2(x)] \times [B_1(x) + B_2(x)] \\ & - A_1(x) \times B_1(x) - A_2(x) \times B_2(x). \end{aligned}$$
4. Compute $C(x)$

What is the time complexity of Karatsuba's algorithm?

Answer

$$T(n) = 3 \cdot T(n/2) + O(n).$$

- Now, there are **only 3 multiplications** of polynomials of degree $\frac{n}{2}$.
- Additional work still takes $O(n)$.

Since $a = 3$, $b = 2$, and $d = \log_2 3 = 1.58\ldots$, and $f(n) = O(n) = O(n^{d-\epsilon})$ for some $\epsilon > 0$, by Case 1 of the Master Theorem, we get:

$$T(n) \in \Theta(n^d) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots}).$$

Answer

$$T(n) = 3 \cdot T(n/2) + O(n).$$

- Now, there are **only 3 multiplications** of polynomials of degree $\frac{n}{2}$.
- Additional work still takes $O(n)$.

Since $a = 3$, $b = 2$, and $d = \log_2 3 = 1.58\ldots$, and $f(n) = O(n) = O(n^{d-\epsilon})$ for some $\epsilon > 0$, by Case 1 of the Master Theorem, we get:

$$T(n) \in \Theta(n^d) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots}).$$

Remarks

- **Practical Application**: This method is in CPython for multiplying large integers.
- **Beyond Karatsuba**: Can be improved further to $O(n \log n)$ using more advanced techniques.

Question 5 [G]/[P3]

You are given **243** balls, where one is heavier while the rest have the same weight. You (your friend) have a balance scale and must determine the heavier ball while minimizing the worst-case number of weighings.

- The balance scale provides only **comparison results** ($<$, $=$, or $>$).
- Each weighing has a cost.

With these information,

a. What is the minimum number of weighings needed?

b. What is the lower bound for **any** algorithm solving this problem?

Answer

Minimum Number of Weighings

1. Divide the balls into three equal groups: A, B, and C.
2. Weigh group A against group B.
   - If $A = B$, the heavier ball is in group C.
   - If $A > B$, the heavier ball is in group A.
   - If $A < B$, the heavier ball is in group B.

Each weighing reduces the balls by $1/3$, which goes:

$$243 \xrightarrow{\text{1st}} 81 \xrightarrow{\text{2nd}} 27 \xrightarrow{\text{3rd}} 9 \xrightarrow{\text{4th}} 3 \xrightarrow{\text{5th}} 1.$$

After 5 weighings, the last ball must be the heavier one.

Optimal Weighings

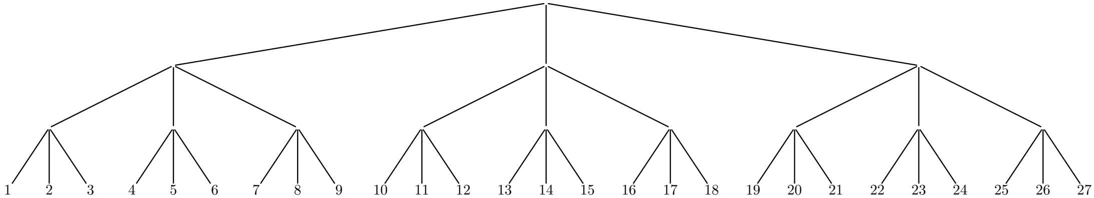

Figure 2: $3^3 = 27$ possible outcomes, with 3 weighings.

- Each weighing divides the balls into at most[2] 3 groups.
- A full **ternary tree** of height h has at most: 3^h leaves.
- Since there are 243 **possible outcomes**, a tree of height 4 is insufficient.
- Thus, at least 5 weighings are necessary.

[2] weighings may not divide the balls into three

Question 6 [G]

You are given an array $A[1..n]$ that is sorted in **non-increasing order**. Your task is to find the largest index i such that $A[i] \geq i$. Design an efficient algorithm to solve this problem.

To guide your approach, consider the following properties of the sorted array:

- If $A[j] \geq j$, then it must hold that (left) $A[j-1] \geq j-1$, unless $j = 0$.
- If $A[j] < j$, then it must follow that (right) $A[j+1] < j+1$, unless $j = n$.

For ease of notation, assume that the array is extended such that $A[0] > 0$ and $A[n+1] < n+1$. Thus, there is a unique i such that $A[i] \geq i$ but $A[i+1] < i+1$.

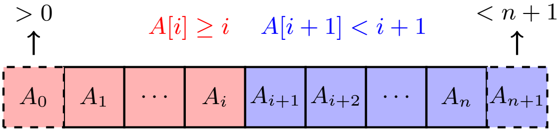

Figure 3: Key observation: Red implies left is red, Blue implies right is blue.

Answer

Method 1: Linear Search

- Perform a linear search to find the largest i such that $A[i] \geq i$.
- This takes $O(n)$ time.

Method 2: Binary Search

- Use binary search, leveraging the given properties of the array.
- This reduces the time complexity to $O(\log n)$.

Method 3: Exponential Search + Binary Search

1. Find the smallest k where $A[2^k] < 2^k$ by testing $k = 0, 1, 2, \ldots$.
   - If $k = 0$, we are already done.
   - Otherwise, this ensures that $A[2^k] < 2^k$ while $A[2^{k-1}] \geq 2^{k-1}$.
2. Apply **binary search** in the range $[2^{k-1}, 2^k]$ to find the largest i such that $A[i] \geq i$.
3. This approach runs in $O(\log i)$ time, where i is the final answer.

Question 7 [G]

Bogosort repeatedly shuffles the array until it happens to be sorted. Analyze its **best-case, worst-case, and average-case** time complexity for an array of length n.

Algorithm 1: Bogosort($A[0..n-1]$)

```
while not IsSorted(A) do
    RandomlyShuffle(A)
return A

Function IsSorted(A):
    for i ← 1 to n − 1 do
        if A[i] < A[i − 1] then
            return false
    return true
```

Note: RandomlyShuffle runs in $O(n)$ using the Fisher-Yates shuffle.

Answer

Best-case

- If the array is already sorted, only one $\mathrm{IsSorted}$ check is needed.
- **Time complexity:** $O(n)$.

Worst-case

- Unbounded; the algorithm may never terminate as shuffles are random.

Average-case

- $n!$ possible permutations, (assume) each equally likely.
- Probability of a correct permutation in one shuffle: $1/n!$
- Expected number of iteration: $n!$,
  - $O(n)$ for $\mathrm{RandomlyShuffle}$ and
  - $O(n)$ for $\mathrm{IsSorted}$.
- **Total expected runtime:** $O(n \cdot n!)$.

Practical [Optional]

Practical repo: To help you further your understanding, not compulsory; Work for Snack!

1. Bruteforce implementation is given, `poly_mult_bruteforce`.
2. Implement the D&C algorithm in code, `poly_mult_dc`.
3. Check that you get this output:

```
Brute force result: [2, 9, 17, 16, 6]
Divide and Conquer result: [2, 9, 17, 16, 6]
```