MIT6.046J Design and Analysis of Algorithms

Xiyuan Yang
2025.10.13

Lecture Notes for advanced algorithms for Open lecture MIT 6.046J

Contents

§1. Introduction

§1.1. Course Overview

1. Divide and Conquer - FFT, Randomized algorithms

2. Optimization - greedy and dynamic programming
3. Network Flow
4. Intractibility (and dealing with it)
5. Linear programming
6. Sublinear algorithms, approximation algorithms
7. Advanced topics

§1.2. Complexity Recall

- **P**: class of problems **solvable** in polynomial time. $O(n^k)$ for some constant k.
  - P 类问题可以使用确定性图灵机在多项式时间内解决的问题集合
- **NP**: class od problems **verifiable** in polynomial time.

> **Example** (Hamiltonian Cycle).
> Find a simple cycle to contain each vertex in V.
> - Easy to evaluate, but hard to calculate!
>
> We have P $\subset$ NP.
> - 在多项式时间内找到正确答案,那么肯定可以在多项式时间内验证答案是否正确。(默认比较两个答案是否相同是可以在多项式实现内实现的)

- **NPC**: NP Complete
  - 问题本身属于 NP 复杂度类
  - 为 NP 困难问题:
    - 所有的 NP 问题都可以在多项式时间内归约到问题 C 上。

§1.3. Interval Scheduling

Requests $1, 2, ..., n$: single resource.

- $s(i)$: the start time
- $f(i)$: the finish time
- $s(i) < f(i)$
- two requests are compatible: $[s(i), f(i)] \cap [s(j), f(j)] \neq \emptyset$

Goal: select a compatible subset of requests with the maximum size.

Solving: Greedy Search!

- Use a simple rule to select a request i.
- Reject all requests incompatible with i.
- Repeat until all requests are processed.

§2. Divide and Conquer

§2.1. Paradigm

Intuition: Splitting bigger problems into smaller problems.

- Solve the sub-problems recursively
- Combine solutions of sub-problems to get overall solutions.

$$T(n) = aT\left(\frac{n}{b}\right) + [\text{work for merge}]$$

- a: The number of sub-problems during recursion.
- b: The size of each sub-problems

For example, for the merge sort:

$$T(n) = aT\left(\frac{n}{2}\right) + O(n)$$

§2.2. Convex Hall

§2.2.1. Brute force for Convex Hull

C_n^2 segments, testing each segment:

- All other points are on the single side: correct
- Else: false

Time Complexity: $O(n^3)$

§2.2.2. Gift Wrapping Algorithms

Given n points in the plane, the goal is to find the smallest polygon containing all points in $S = \{(x_i, y_i) | i = 1, 2, ..., n\}$. We ensure no two points share the same x coordinates or the y coordinates, no three points are in the same line.

Intuition: Gift wrapping algorithms.

Recordings (Simple Gift Wrapping Algorithms).
- Select the initial point
  - Find the point which has the smallest x coordinates or the smallest y coordinates.
- 找到旋转角度最大的点，作为凸包上的点选入
  - 这也可以看做是一种橡皮筋手搓生成凸包的过程
- Time Complexity: $O(n \cdot h)$

§2.2.3. Divide and Conquer for Convex Hall

Recordings (When to use Divide and Conquer).
- 分治最关键的是两个步骤：
  - 分解成若干个子问题（递）
  - 把子问题的结果合并起来（归）
- 如果使用分治法，那务必重视的一点是递归的终点（最简单的情况）必须是简单可解的。($O(1)$ Time Complexity)

For simple condition: when $n \leq 3$, the convex hall is quite simple! All the points are the vertices of the convex hall.

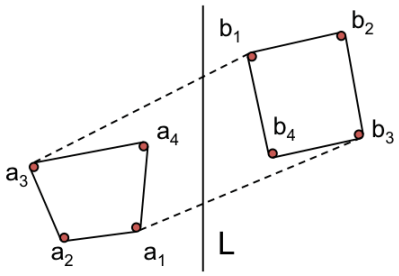
Now, we need to solve two things:

- When to divide
  - With x coordinates
  - More like half splitting!

- When to conquer
  - The most critical step!
  - We need to finding the bridges (Upper Bridge and Lower Bridge) to form the bigger convex hall.
    - **Two Finger Algorithms**

Recordings (Two Finger Algorithms).
- 基本思路类似于双指针法实现线性扫描
- 基本思想还是不断旋转找到最外部的切线

Example



a_3, b_1 is upper tangent. $a_4 > a_3$, $b_2 > b_1$ in terms of Y coordinates.
a_1, b_3 is lower tangent, $a_2 < a_1$, $b_4 < b_3$ in terms of Y coordinates.

a_i, b_j is an upper tangent. Does not mean that a_i or b_j is the highest point. Similarly, for lower tangent.

Figure 1: Convex Hall Conquering Steps

Time Complexity:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

Thus total time complexity: $\Theta(n \log n)$

For the compute of this time complexity, we can use Master Theorem.

§2.3. Master Theorem

For simple cases:

$$T(n) = aT\left(\frac{n}{b}\right)$$

To compute this complexity, we use the recursive tree to solve this:

$$T(n) = a^k T\left(\frac{n}{b^k}\right)$$

For the recursion endpoint, $\frac{n}{b^k} = 1$, we can compute:

$$T(n) = a^{\log_b n} T(1) = n^{\log_b a} T(1) = O\left(n^{\log_b a}\right)$$

For general cases:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

We need to compare $f(n)$ and $n^{\log_b a}$.

§2.3.1. The Work at the Leaves Dominates

$$f(n) = O\left(n^{\log_b(a-\varepsilon)}\right)$$

Then it means that recursion part dominates! ($T(n) = aT\left(\frac{n}{b}\right)$). Thus, the time complexity is:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) = aT\left(\frac{n}{b}\right) + O\left(n^{\log_b(a-\varepsilon)}\right) = \Theta\left(n^{\log_b a}\right)$$

§2.3.2. The Work is Balanced

$$f(n) = \Theta\left(n^{\log_b a} \cdot \log^k n\right)$$

Then it means the two parts are both the dominant parts! The total time complexity remains the same.

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) = aT\left(\frac{n}{b}\right) + \Theta\left(n^{\log_b a} \cdot \log^k n\right) = \Theta\left(n^{\log_b a} \cdot \log^{k+1} n\right)$$

§2.3.3. The Work at the Root Dominates

$$f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right)$$

and:

$$\exists c \in R, \exists N_0 \in \mathbb{N}, \forall n > N_0 : af\left(\frac{n}{b}\right) \leq cf(n)$$

Recordings (Regular Condition).

- 这个条件说明划归到子问题的时候时间复杂度可能很大，但是对于大问题“分而治之”的复杂度是非常昂贵的。
- 总复杂度有递归的最高层（根节点）的代价决定，这也保证该情况下时间复杂度的量级为 $\Theta(f(n))$

Then the total time complexity:

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

Example (Example for the work at the root dominates).
例如如果递归的时间复杂度为：

$$T(n) = 3T\left(\frac{n}{4}\right) + n^2$$

$a = 3, b = 4$, $T(n) = \Theta(n^2)$

§2.4. Median Finding

Problem 2.4.1 (Median Finding).
Given set of n numbers, define rank(x) as number of numbers in the set that are $\leq x$. Find element of rank $\lfloor \frac{n+1}{2} \rfloor$ (lower median) and $\lceil \frac{n+1}{2} \rceil$ (upper median).

Obviously, we can use **sorting algorithms** to solve this! The time complexity is $\Theta(n \log n)$.

Simple Algorithms: Define problem as `Select(S,i)` to find the `i` th element value in the set S.

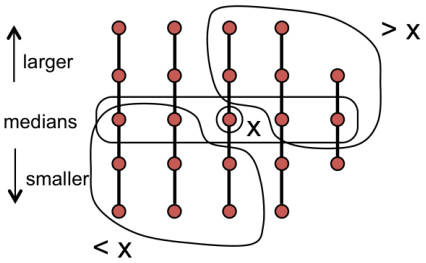
- Pick $x \in S$
  - We just pick it cleverly
- Compute $k = \text{rank}(x)$
- $B = \{y \in S | y < x\}$
- $C = \{y \in S | y > x\}$
- algorithms:
  - If $k = i$: `return x`
  - If $k < i$: `return Select(C,i-k)`
  - If $k > i$: `return Select(B,i)`

For dummy choices for selecting $x \in S$, for the worse case, the time complexity is $\Theta(n^2)$.

§2.4.1. Picking x cleverly

- Arrange S into columns of size 5 ($\lceil \frac{n}{5} \rceil$ cols).
- Sort each columns in linear time.
- Find **medians of medians** as the selected x.

Recordings (Why selecting this?).
- 对于简单的取常数或者中间值的方法在极端情况下会退化到平方时间复杂度，因为我们难以知道全局数据的分布特征，因此我们很难选择一个好的 splitting
- 和快速排序很类似！我们希望选择一个好的 splitting ，这样让递归算法变成对数级别的。
- 而下面的选择可以保证 splitting 的效率，即至少有 $3(\lceil \frac{n}{10} \rceil - 2)$ 的点被分到左边并且至少有 $3(\lceil \frac{n}{10} \rceil - 2)$ 的点被分到右边。



How many elements are guaranteed to be $> x$?
Half of the $\lceil \frac{n}{5} \rceil$ groups contribute at least 3 elements $> x$ except for 1 group with less than 5 elements and 1 group that contains x.
At lease $3(\lceil \frac{n}{10} \rceil - 2)$ elements are $> x$, and at least $3(\lceil \frac{n}{10} \rceil - 2)$ elements are $< x$

Figure 2: SELECT for medians of medians

Recurrence:

$$T(n) = T\left(\lceil \frac{n}{5} \rceil\right) + T\left(\frac{7n}{10} + 6\right) + \Theta(n)$$

- $T\left(\lceil \frac{n}{5} \rceil\right)$ 是找到中位数的中位数的算法时间
- $\Theta(n)$ 是分组线性扫描需要的时间复杂度
- $\frac{7n}{10} + 6$ 代表子问题的规模，因为我们保证$3\left(\lceil \frac{n}{10} \rceil - 2\right)$ 会被分到对应的组，因此最坏情况就是 $\frac{7n}{10} + 6$

Solving this recurrence.

Proof by induction:

We need to solve: $\exists \alpha > 0, n_0 \geq 1, \forall n \geq n_0, T(n) \leq \alpha n$.

We select $n_0 = 140$.

for $0 \leq n \leq 140$, obvious. we select $\alpha = \max(T_{max}(1 \leq n \leq 140), 20c)$.

for $n > 140$.

We propose $\forall k < n, T(k) \leq \alpha(k)$

we need to prove by induction $T(n) \leq \alpha n$.

$$T(n) = T\left(\lceil \frac{n}{5} \rceil\right) + T\left(\frac{7}{10}n + 6\right) + \textcircled{H}(n)$$

$$\leq T\left(\lceil \frac{n}{5} \rceil\right) + T\left(\frac{7}{10}n + 6\right) + cn.$$

$$\leq \alpha \lceil \frac{n}{5} \rceil + \alpha\left(\frac{7}{10}n + 6\right) + cn \leq \alpha\left(\frac{n}{5} + 1\right) + \alpha\left(\frac{7}{10}n + 6\right) + cn$$

$$= \frac{9}{10}\alpha n + 7\alpha + cn.$$

we need to prove $\frac{9}{10}\alpha n + 7\alpha + cn \leq \alpha n$, for some case we select α

$$cn + 7\alpha \leq \frac{1}{10}\alpha n.$$

select $\alpha = 20c$, then obviously $cn \geq 7\alpha = 140c$ ☆

thus we can prove the induction!

Figure 3: Induction proof for median finding algorithms

Example (Medians of Medians).
给定一个数组和 k 值，尝试求解这个数组中的第 k 大的元素。要求保证时间复杂度为 $O(n)$.

```python
from typing import List

class Solution:
    def find_median_of_small_array(self, arr: List[int]) -> int:
        # O(1) for constant length arrays
        arr.sort()
        return arr[len(arr) // 2]

    def select_pivot(self, arr: List[int]) -> int:
```

```
        n = len(arr)
        if n <= 5:
            return self.find_median_of_small_array(arr)

        # splitting into sub-lists
        sublists = [arr[i : i + 5] for i in range(0, n, 5)]
        medians = [self.find_median_of_small_array(sublist) for sublist in sublists]

        # ! very important recursion!
        # That is the T(n/5) part
        return self.findKthSmallest(medians, len(medians) // 2 + 1)

    def findKthSmallest(self, arr: List[int], k: int) -> int:
        n = len(arr)
        if n == 1:
            return arr[0]

        pivot = self.select_pivot(arr)

        # do partition based on selected pivot
        less = [x for x in arr if x < pivot]
        equal = [x for x in arr if x == pivot]
        greater = [x for x in arr if x > pivot]

        len_less = len(less)
        len_equal = len(equal)

        # the core recursion part remains unchanged
        if k <= len_less:
            return self.findKthSmallest(less, k)
        elif k <= len_less + len_equal:
            return pivot
        else:
            new_k = k - len_less - len_equal
            return self.findKthSmallest(greater, new_k)

    def findKthLargest(self, nums: List[int], k: int) -> int:
        n = len(nums)
        k_smallest = n - k + 1
        return self.findKthSmallest(nums, k_smallest)
```

§2.5. Matrix Multiplication

For simple matrix multiplication, the time complexity is $O(n^3)$.

$$c_{i,j} = \sum_{k=1}^{p} a_{i,k} b_{k,j}$$

- n^3 times multiplication.
- $n^3 - n^2$ times addiction.

§2.5.1. Strassen Algorithms

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

For simple divide and conquer algorithms:

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

这个是基本的分治算法，对于子矩阵，需要进行 8 次 子矩阵的乘法和 4 次子矩阵的加法。

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Based on Master theorem, the time complexity is $O(n^3)$, remains unchanged!

The break through for strassen algorithms are reducing matrix multiplication from 8 times into 7 times by reducing repeated computation!

$$M_1 = (A_{11} + B_{22})(B_{11} + B_{22})$$
$$M_2 = (A_{21} + A_{22})B_{11}$$
$$M_3 = A_{11}(B_{12} - B_{21})$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12})B_{22}$$
$$M_6 = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$

7 times matrix multiplication ($\frac{n}{2} \times \frac{n}{2}$), and 18 times addiction.

$$C_{11} = M_1 + M_4 - M_5 + M_7$$
$$C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4$$
$$C_{22} = M_1 - M_2 + M_3 + M_6$$

Thus the time complexity:

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2) = \Theta\left(n^{\log_2 7}\right) \approx \Theta(n^{2.807})$$

§2.6. FFT

§2.6.1. Polynomials

All about polynomial:

$$A(x) = \sum_{k=1}^{n-1} a_k x^k = [a_0, a_k, ..., a_{n-1}]$$

§2.6.2. Operations for $A(x)$

§2.6.2.1. Evaluation

Definition 2.6.2.1.1 (Evaluation)**.**
Given x, calculate $A(x)$.

Theorem 2.6.2.1.1 (Horner's Rule).
我们希望更少次数的乘法和加法

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

- Before: $\Theta(n^2)$ times multiplication and $\Theta(n)$ times addition.
- After: $\Theta(n)$ times multiplication and $\Theta(n)$ tines addition.
- Thus the time complexity: $O(n^2) \to O(n)$

§2.6.2.2. Addition

Definition 2.6.2.2.1 (Addition).

$$C(x) = A(x) + B(x)$$

Obviously, the time complexity is $O(n)$ for addition.

§2.6.2.3. Multiplication

Definition 2.6.2.3.1 (Multiplication).

$$C(x) = A(x) \times B(x), \forall x \in X$$

- Naive calculation: $O(n^2)$

$$c_k = \sum_{j=0}^{K} a_j b_{K-j}$$

We want to achieve $O(n \log n)$

Algorithms vs.	Representations		
	Coefficients	Roots	Samples
Evaluation	$O(n)$	$O(n)$	$O(n^2)$
Addition	$O(n)$	∞	$O(n)$
Multiplication	$O(n^2)$	$O(n)$	$O(n)$

§2.6.3. Representations

- Coefficient Vectors
- Roots and a scale term
- Samples

§2.6.4. Vendermonde Matrix

$$V \cdot A = \begin{pmatrix} 1 & x_0 & x_0^2 & ... & x_0^{n-1} \\ 1 & x_1 & x_1^2 & ... & x_1^{n-1} \\ 1 & x_2 & x_2^2 & ... & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & ... & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ ... \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ ... \\ y_{n-1} \end{pmatrix}$$

Recordings (多项式插值).
- 这个本质上也可以看做是一种多项式插值的手段
- 我们希望从 sample 的形式转变为 coefficient 的形式
- 根据线性代数的知识，范德蒙行列式只有在 sample 的点均不相同的情况下才是不可逆的，因此只要 sample 了 n 个不相同的点，就可以保证能够求解可逆矩阵，但是可逆矩阵的时间复杂度是 $O(n^3)$，因此实际插值并不会采用这个原始的算法。

$$\Pi_{0\le i<j\le n-1}(x_j - x_i) = 0$$

We want to calculate A, thus we need to calculate:

$$A = V^{-1}Y$$

§2.6.5. Divide and Conquer for FFT

The original input: A_{eff} and B_{eff} as two vectors, and we need to calculate C_{eff} for the new coefficients after polynomial multiplications.

We know, if we have N samples for two polynomials, just calculate $A(x_k) \cdot B(x_k)$.

因此，如果我们需要解决多项式乘法的问题，实际上我们可以将问题拆分为：

- 预处理计算 $\text{FFT}(A)$ and $\text{FFT}(B)$: we want it to be $O(N \log N)$
- $C_{\text{point}} = A_{\text{point}} \odot B_{\text{point}}$: it is $O(N)$, not the bottleneck
- $\text{IFFT}(C_{\text{point}})$: we want it to be $O(N \log N)$

§2.6.5.1. How to divide?

Divide into Even and Odd Coefficients $O(n)$

$$A_{\text{even}} = \sum_{k=0}^{\lceil \frac{n}{2}-1 \rceil} a_{2k}x^k = [a_0, a_2, ..., a_{2l}]$$

$$A_{\text{odd}} = \sum_{k=0}^{\lceil \frac{n}{2}-1 \rceil} a_{2k+1}x^k = [a_1, a_3, ..., a_{2l}]$$

§2.6.5.2. How to conquer

$$A(x) = A_{\text{even}}(x^2) + x \cdot A_{\text{odd}}(x^2) \text{ for } x \in X$$

Thus, we need to recursively calculate $A_{\text{even}(y)}, y \in X^2 = \{x^2 \mid x \in X\}$

$$T(n, |X|) = 2T\left(\frac{n}{2}, |X|\right) + O(n + |X|) = O(n^2)$$

Actually, the time complexity does not change... However, if we can achieve the conquer time complexity as follows, we can do a great improvement:

$$T(n, |X|) = 2T\left(\frac{n}{2}, \frac{|X|}{2}\right) + O(n + |X|)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Thus, the time complexity is $O(n \log n)$! We should select X in a clever way in which it is **collapsing**, or we can say:

$$|X^2| = \frac{|X|}{2}$$

for the base case, $X = \{1\}$ when $|X| = 1$.

Recordings (Roots of Unity).

- 从数值来讲，就是在复平面内不断求根
- 在复平面内就是不断的取中间的过程

$$(\cos\theta, \sin\theta) = \cos\theta + i\sin\theta = e^{i\theta}$$

$$\theta = 0, \frac{1}{n}\tau, \frac{2}{n}\tau, ..., \frac{n-1}{n}\tau (\tau = 2\pi)$$

Then, we successfully implement FFT with the time complexity for $O(n \log n)$!

- Well defined recursion
- Selected X, for $x_k = e^{\frac{i\tau k}{n}}$

§2.6.6. Inverse Discrete Fourier Transform (IFFT)

We want to return the coefficients from the multiplied samples. The transformation of this form is $A = V^{-1}Y$. Thus, all we need to do is calculate V^{-1}, or the inverse of Vendermonde Matrix!

Theorem 2.6.6.1 (Calculate the inverse).

$$V^{-1} = \frac{1}{n}\overline{V}$$

where $\overline{V}$ is the complex conjugate of V.

Proof. We claim that $P = V \cdot \bar{V} = nI$:

$$\begin{aligned} p_{jk} &= (\text{row } j \text{ of } V) \cdot (\text{col. } k \text{ of } \bar{V}) \\ &= \sum_{m=0}^{n-1} e^{ij\tau m/n} \overline{e^{ik\tau m/n}} \\ &= \sum_{m=0}^{n-1} e^{ij\tau m/n} e^{-ik\tau m/n} \\ &= \sum_{m=0}^{n-1} e^{i(j-k)\tau m/n} \end{aligned}$$

Now if $j = k$, $p_{jk} = \sum_{m=0}^{n-1} = n$. Otherwise it forms a geometric series.

$$\begin{aligned} p_{jk} = &= \sum_{m=0}^{n-1} (e^{i(j-k)\tau/n})^m \\ &= \frac{(e^{i\tau(j-k)/n})^n - 1}{e^{i\tau(j-k)/n} - 1} \\ &= 0 \end{aligned}$$

because $e^{i\tau} = 1$. Thus $V^{-1} = \frac{1}{n}\bar{V}$, because $V \cdot \bar{V} = nI$. □

This claim says that the Inverse Discrete Fourier Transform is equivalent to the Discrete Fourier Transform, but changing x_k from $e^{ik\tau/n}$ to its complex conjugate $e^{-ik\tau/n}$, and dividing the resulting vector by n. The algorithm for IFFT is analogous to that for FFT, and the result is an $O(n \lg n)$ algorithm for IDFT.

Recordings (Inverse).

- 换句话说，范德蒙行列式是可以快速求解的，因为旋转基本根的良好性质。

§3. Conclusion