算法以及复杂性

回顾 & 思考

- 计算机能做什么?
  - 可计算问题
  - 不可计算问题
- 问题:
  - 可计算问题都能够被有效解决吗?

新的视角

如何快速计算问题

是否存在算法能够快速计算某问题?

如何定义快速

- 什么是运行时间?
- 什么是时间复杂度?
- 什么是好的时间复杂度?

什么是运行时间?

可能的定义

- 我的电脑跑了多少秒?
- 相关因素
  - 我用了什么算法
  - 问题的输入是什么
  - 我的电脑多少钱，我选电脑的眼光，我电脑用了几年。

我们希望的定义

- 排除不同电脑带来的影响
- 使用统一的计算模型（如图灵机）
  - 考虑图灵机上进行的单位操作的数量
- 定义：运行时间=固定计算模型下**单位操作**的数量

进一步思考

- 与运行时间相关的因素
  - 我用了什么算法
  - 我的输入是什么
- 下一个目标
  - 评价某个算法的快速程度

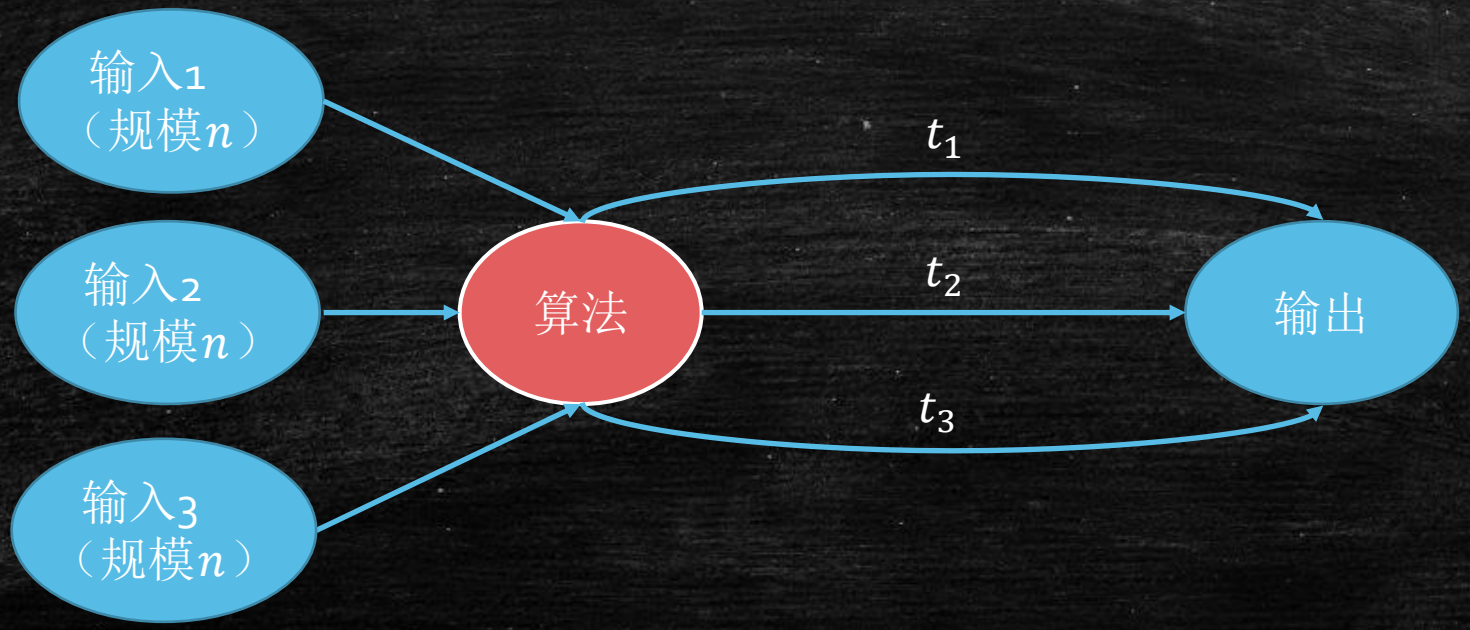
什么是时间复杂度

问题输入如何影响运行时间?

- 与运行时间相关的因素
  - 我用了什么算法
  - 我的输入是什么
- 不同输入
  - 规模不同
  - 长相不同
- 问题：如何设计规则让不同算法同台竞技?

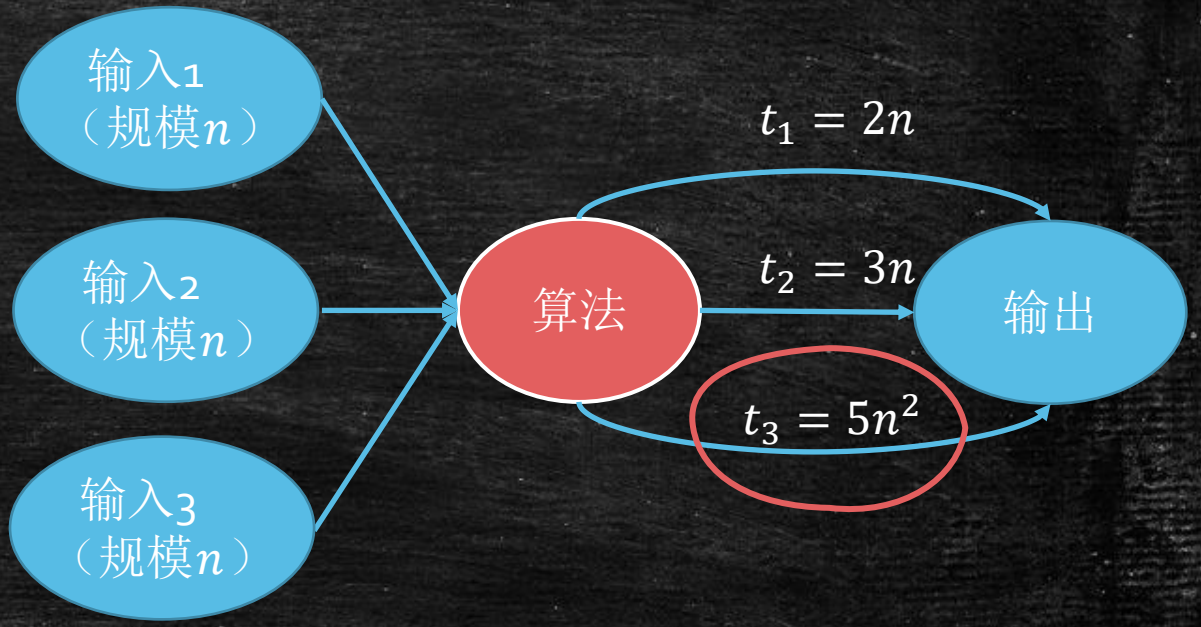
时间复杂度

- 定义一个运行时间关于输入规模的函数 $T(n)$。
  - 比较不同算法的 $T(n)$ 的优劣。
- 问题：那同样规模，不同长相的输入，运行时间不同怎么办？



时间复杂度（最坏情况）

- 时间复杂度的定义
  - 考虑**最坏情况**
  - 所有规模为n的输入中，
  - 最大的运行时间。
- 例子: $T(n) = 5n^2$
  - 算法对所有规模为n的输入，
  - 最差也能在$5n^2$的运行时间内解决。



什么是好的时间复杂度

不同的时间复杂度

- $T(n) = 1000$
- $T(n) = 9999n+2$
- $T(n) = 500n + 300$
- $T(n) = 20n^2 + 30n$
- $T(n) = 3 \cdot 2^n$
- $T(n) = 100 \cdot e^{3n}$
- 哪个好?
- 我们希望这个函数**增长**的慢，在n**比较大**时表现得比较好。

多项式时间复杂度

- 因为种种原因，我们认为多项式时间的时间复杂度是不错的。
- $T(n) = a_k n^k + a_{k-1} n^k + \cdots + a_1 n + a_0$
- 剧透：
  - **确定型图灵机**能在多项式时间复杂度内解决的问题集合叫做P。
  - **非确定型图灵机**能在多项式时间复杂度内解决的问题集合叫做NP。

如何分析时间复杂度

定义计算模型

- 图灵机?
  - 和我们电脑的实际单位操作略有出入，和我脑子里想的也有出入。
- Word RAM 模型
  - 和我们编写的程序在电脑上的单位操作更符合。
- 我们会使用 Word RAM 模型来分析时间复杂度。
- 关键信息：
  - Word RAM的计算能力和图灵机等价。
  - Word RAM的计算能力在多项式内也和图灵机等价。

Word RAM 模型（非严格定义）

- Word RAM Model
  - RAM: Random Access Model（随机访问模型）。
- The model was created by Michael Fredman and Dan Willard in 1990 to simulate programming languages like C.
- **单位操作**
  - $a[1] \leftarrow a[5]$
  - $a[2] \leftarrow a[3] + a[5]$
  - $a[5] \leftarrow a[4] \times a[6]$
  - ……

RAM

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	…	$i=2^w$
$a[i]$	$\leq 2^w$	$\leq 2^w$	$\leq 2^w$	$\leq 2^w$	$\leq 2^w$	$\leq 2^w$	…	$\leq 2^w$

w: word 的位数

Word 的位数设置

- 我们以后会经常遇到的问题形式
  - Sorting
    - Sort integers $a_1, a_2, \dots, a_n$.
  - Find Max
    - Find the max integer among $a_1, a_2, \dots, a_n$
- 如何设置位数
  - 输入越大，需要的位数越大。
  - 我们希望我们的位数定义在一个恰当的范围。
  - 既能合理得存下输入，也不能拥有超能力。
  - w定义在$\max\{\log n, \log a_i\}$这个级别。
- 设置的效果
  - 与**输入规模相仿**的的基本算数运算，可以被理解为单位操作。

与图灵机的一些区别

- 图灵机不是随机访问
- 不是很严格得说，图灵机可以多花 n 倍的时间来模拟Word RAM，所以，他们在多项式时间内等价。

让我们分析一个算法（程序）

简单的例子

1: int sum=0,n;	$+3$
2: input(n);	$+1$
3: int a[n];	$+n$
4: input(a);	$+n$
5: for i=0 to n	$+n$
6: sum+=a[i];	$+n$
7: output(sum);	$+1$

还是有点麻烦

- 你希望区分如下复杂度吗？
  - $312n^3 + 124n^2 + 15$
  - $351n^3 + 100n^2 + 150$
- 我们希望什么？
  - 区分复杂度的级别，而不是细节。
- $T(n) = O(n^2)$。
  - 含义：当n足够大时，$T(n)$的级别不超过n^2。
  - $n^2 + 100n + 10000$
  - $1000n^2 + 10n$
  - $n + 100$

Big O Notation的严格定义

- Big O Notation (上界)
  - $T(n) = O(g(n))$
  - $\exists C, n_0$, s.t. $\forall n > n_0$, $T(n) \leq C \cdot g(n)$
- Big Ω Notation (下界)
  - $T(n) = \Omega(g(n))$
  - $\exists C, n_0$, s.t. $\forall n > n_0$, $T(n) \geq C \cdot g(n)$
- Big Θ Notation (所处的精确级别)
  - $T(n) = O(g(n))$
  - $T(n) = \Omega(g(n))$

讨论

- 证明 n^2 is not $O(n)$.
- 令 $p(n) = a_k n^k + a_{k-1} n^{k-1} + a_{k-2} n^{k-2} \ldots + a_1 n + a_0$
  - $a_k \geq 0$,
- 证明 $p(n) = O(n^k)$?
- 证明 $p(n) = \Theta(n^k)$?
- 证明
  - $3^n \neq O(2^n)$?
  - $\log_2 n = \Omega(\ln n)$?
- 存不存在两个函数 f 和 g,
  - $f(n) \neq O(g(n))$ and $f(n) \neq \Omega(g(n))$?
  - Require non-decreasing?

理论计算机中关心的话题

如何快速计算问题

- 一个问题是不是能被不错地解决
  - 不错：以一个OK的速度，即多项式时间复杂度。
  - 哪些问题是能够被不错地解决
  - 一个重要理论问题：P? NP。
  - 一些相关的概念：P, NP, NP-Hard.....
- 我已经知道他能被不错得解决之后，可不可以更快。
  - 如何在更快得解决各种P问题（设计更好的算法）。

让我们认识以及分析一些有趣的算法

寻找n个数中的第k小的数

Selection Problem

- **Input:** A set S of n integers $x_1, x_2, \dots, x_n$ and an integer k
- **Output:** The k-th smallest integer x^* among $x_1, x_2, \dots, x_n$

One-by-one Selection

- **Input:** A set S of n integers $x_1, x_2, \dots, x_n$ and an integer k
- **Output:** The k-th smallest integer x^* among $x_1, x_2, \dots, x_n$
- Plan 1
  - Select the smallest integer. $O(n)$
  - Select the 2nd smallest integer. $O(n-1)$
  - …
  - Select the k-th smallest integer. $O(n-k+1)$
  - Total running time $\boldsymbol{O(nk)}$

Sorting

- **Input:** A set S of n integers $x_1, x_2, \dots, x_n$ and an integer k
- **Output:** The k-th smallest integer x^* among $x_1, x_2, \dots, x_n$
- Plan 2
  - $x_1, x_2, \dots, x_n$ Sort the integers in ascending order. $O(n \log n)$
  - Output the k-th integer. $O(1)$
  - Total running time $\boldsymbol{O(n \log n)}$

Divide and Conquer

- **Input:** A set S of n integers $x_1, x_2, \dots, x_n$ and an integer k
- **Output:** The k-th smallest integer x^* among $x_1, x_2, \dots, x_n$
- Plan 3: Divide and Conquer
  - Divide:
    - Pick an arbitrary value v among $x_1, x_2, x_3, \dots$.
    - Divide $x_1, x_2, x_3, \dots$ into three subsets:
      - L : $x < v$
      - M : $x = v$
      - R : $x > v$
  - Recurse: find x^* in the subset contains x^*.
  - Combine: we already have x^*!

Divide

- Choose $v = 3$.

- What is L , M, and R?

- L: 

- M: 

- R: 

Divide

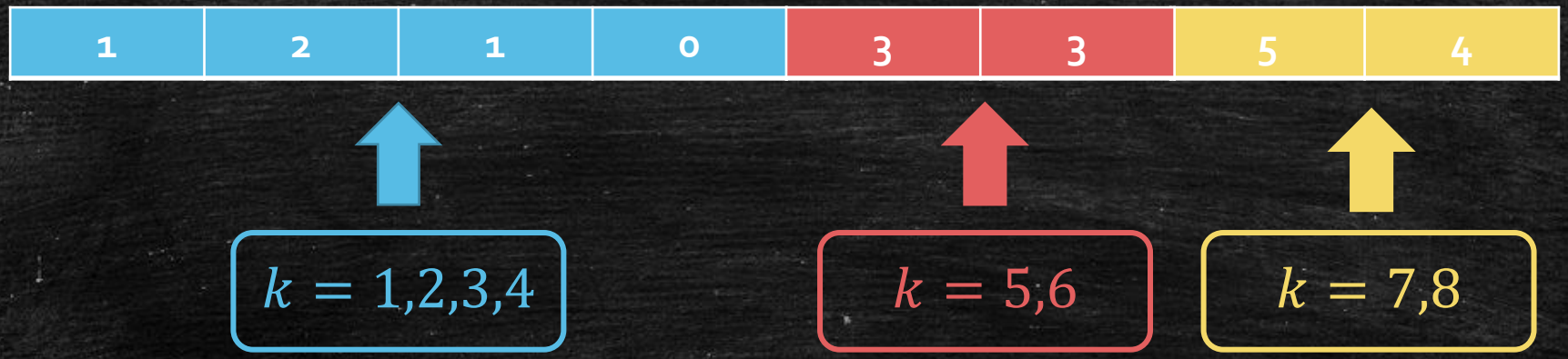
- L: 

- M: 

- R: 

Recurse

- Roughly sorted list



Recurse

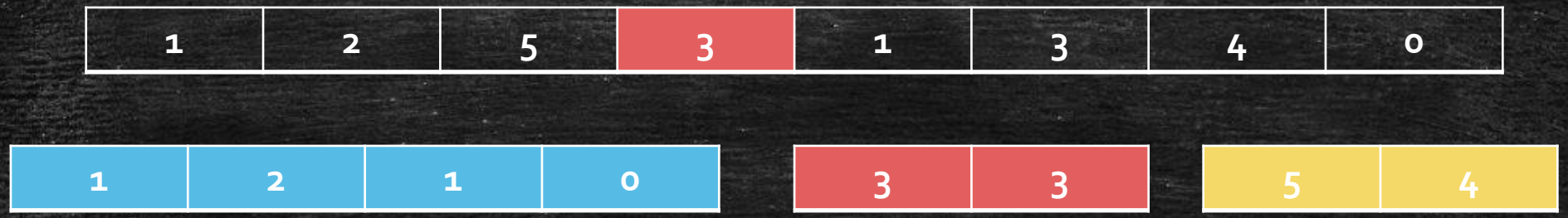
- How to find x^* in L,M,R?
  - Recall x^* is the k-th smallest integer in S.
- L:

  - x^* is the k-th integer in L
- M:

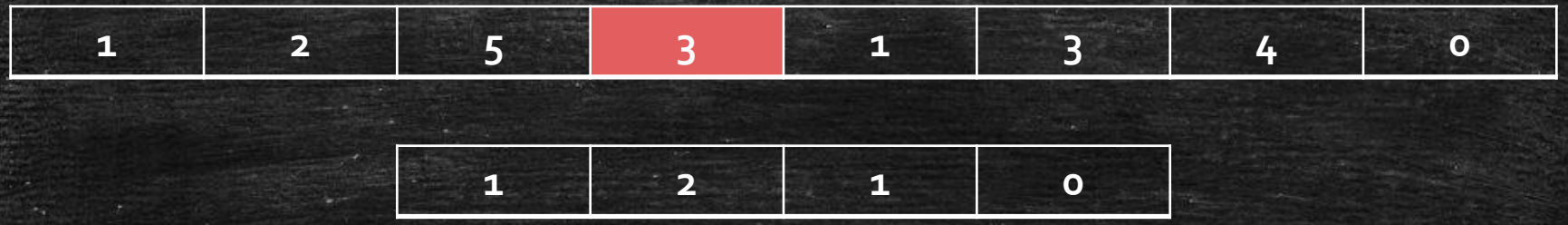
  - $x^* = 3$
- R:

  - x^* is the $(k - |\mathrm{L}| - |\mathrm{M}|)$-th integer in R

Example: $k = 4$

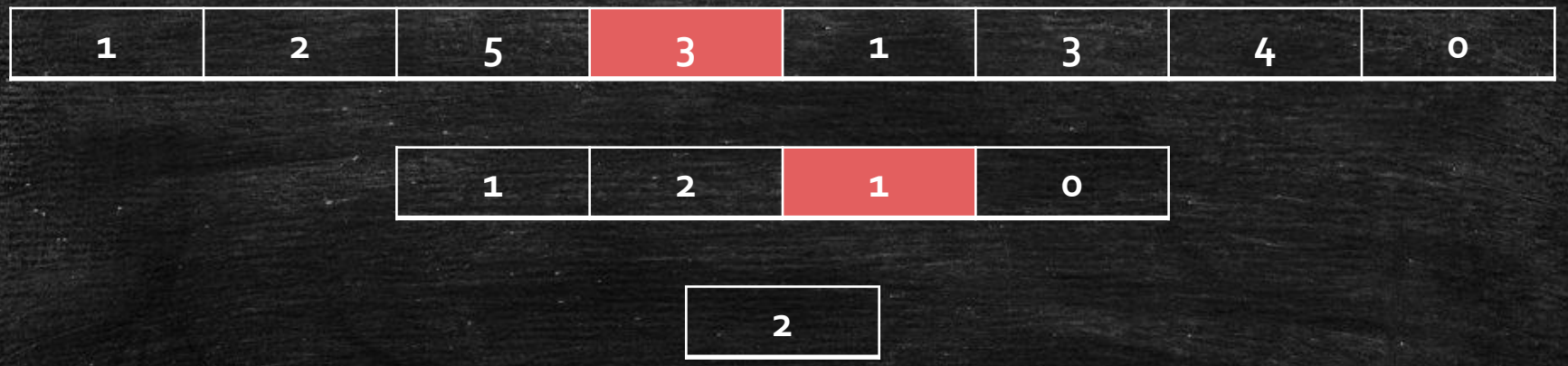


Example: $k = 4$

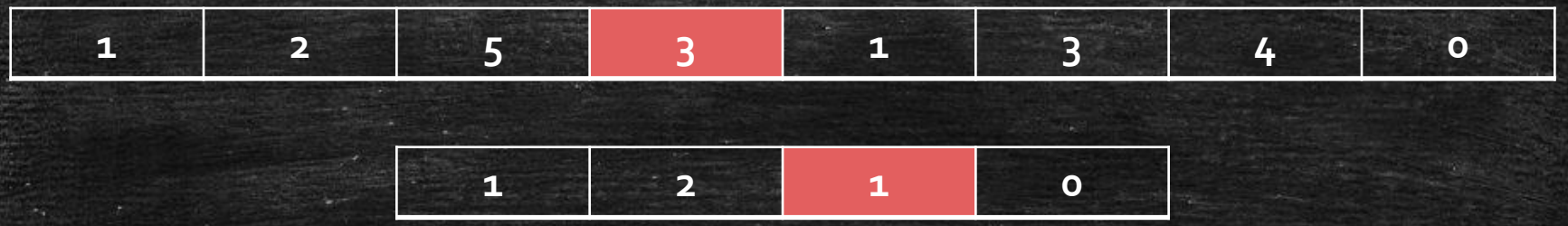


Example: $k = 4$

Example: $k = 4$



Example: $k = 4$



Output 2

Formalize

Function Select(S,k)

- Divide:
  - Pick an arbitrary value v among $x_1, x_2, x_3, \dots$.
  - Divide $x_1, x_2, x_3, \dots$ into three subsets:
    - L : $x < v$,
    - M : $x = v$,
    - R : $x > v$.
- Recurse:
  - Recurse the subset contains x^*.
    - If $k \leq |L|$, output Select(L,k);
    - If $|L| < k \leq |L| + |M|$, output v;
    - If $|L| + |M| < k$, output Select(R, k − |L| − |M|).

Is it correct?

Running Time

We want to know $T(n)$

Function Select(S,k)

- Divide:
  - Pick an arbitrary value v among $x_1, x_2, x_3, \dots$.
  - Divide $x_1, x_2, x_3, \dots$ into three subsets:
    - L : $x < v$,
    - M : $x = v$,
    - R : $x > v$.

Divide: $O(n)$

- Recurse:
  - Recurse the subset contains x^*.
    - If $k \leq |L|$, output Select(L,k);
    - If $|L| < k \leq |L| + |M|$, output v;
    - If $|L| + |M| < k$, output Select(R, k − |L| − |M|).

$T(|L|)$

$O(1)$

$T(|R|)$

Running Time

- $T(n) \leq O(n) + \max\{T(|L|), T(|R|)\}$

$$\leq O(n) + T(n-1)$$

$$\leq O(n) + O(n-1) + T(n-2) \leq \cdots$$

$$= O(n) + O(n-1) + O(n-2) + \cdots + O(1) = \boldsymbol{O(n^2)}$$

Fact

- $|L| + |M| + |R| = |S| = n$
- $|M| \geq 1$
- $|L|, |R| \leq n-1$

- Very Bad!
  - One-by-one: $O(nk)$
  - Sorting: $O(n \log n)$

Is it really that bad?

- Yes, the unluckiest case:
  - $k = 1$
  - Each time, v is the largest integer.
  - $T(n) = O(n) + T(n-1) = O(n) + O(n-1) + T(n-2) = \cdots = O(n^2)$
- What if we are **super lucky**?
  - Each time, v is in the middle.
  - $T(n) = T\left(\frac{n}{2}\right) + O(n) = T\left(\frac{n}{4}\right) + O\left(\frac{n}{2}\right) + O(n) = \cdots = O(n)$.
- What if we are lucky?
  - Each time, v is in the middle range $[\frac{1}{3}n, \frac{2}{3}n]$.
  - $T(n) = T\left(\frac{2}{3}n\right) + O(n) = T\left(\frac{4}{9}n\right) + O\left(\frac{2}{3}n\right) + O(n) = \cdots = O(n)$.
- Idea: to make us reasonably lucky in average by randomness.

What is the next?

- Improving the running time **with** randomness.
- Improving the running time **without** randomness.

Using Randomness!

Function Select(S,k)

- Divide:
  - Pick an arbitrary value v among $x_1, x_2, x_3, \dots$.
  - Divide $x_1, x_2, x_3, \dots$ into three subsets:
    - L : $x < v$,
    - M : $x = v$,
    - R : $x > v$.
- Recurse:
  - Recurse the subset contains x^*.
    - If $k \leq |L|$, output Select(L,k);
    - If $|L| < k \leq |L| + |M|$, output v;
    - If $|L| + |M| < k$, output Select(R, k $-$ |L| $-$ |M|).

Using Randomness!

Function Select(S,k)

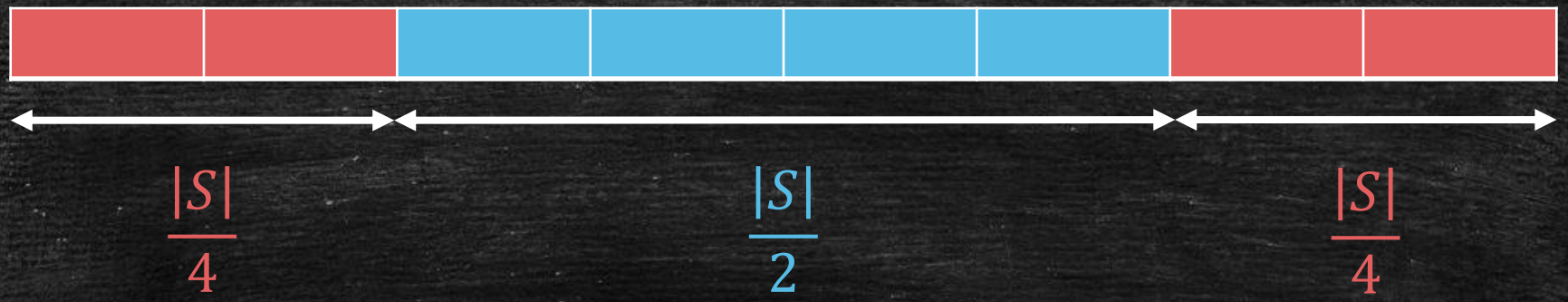
- Divide:
  - Pick an ~~arbitrary~~ value v among $x_1, x_2, x_3, \dots$.
  - Divide $x_1, x_2, x_3, \dots$ into three subsets:
    - L : $x < v$,
    - M : $x = v$,
    - R : $x > v$.
- Recurse:
  - Recurse the subset contains x^*.
    - If $k \leq |L|$, output Select(L,k);
    - If $|L| < k \leq |L| + |M|$, output v;
    - If $|L| + |M| < k$, output Select(R, k $-$ |L| $-$ |M|).

Quick Selection

Function Select(S,k)

- Divide:
  - Pick a **random** value v among $x_1, x_2, x_3, \dots$.
  - Divide $x_1, x_2, x_3, \dots$ into three subsets:
    - L : $x < v$,
    - M : $x = v$,
    - R : $x > v$.
- Recurse:
  - Recurse the subset contains x^*.
    - If $k \leq |L|$, output Select(L,k);
    - If $|L| < k \leq |L| + |M|$, output v;
    - If $|L| + |M| < k$, output Select(R, k $-$ |L| $-$ |M|).

When we are lucky



- : Lucky pivot area
- : Bad pivot area
- Fact 1: With $\frac{1}{2}$ probability, we are lucky!
- Fact 2: If we are always lucky, $T(n) = T\left(\frac{3n}{4}\right) + O(n) = O(n)$

Analysis

- $\tau(n)$: Time we reduce n to $\frac{3n}{4}$
- $T(n) = \tau(n) + T(\frac{3n}{4})$
- $E[\tau(n)]$: The expected time we reduce n to $\frac{3n}{4}$
- $E[T(n)] = E\left[\tau(n) + T\left(\frac{3n}{4}\right)\right]$

  $= E[\tau(n)] + E\left[T\left(\frac{3n}{4}\right)\right]$
- $E[\tau(n)] = O(n)$
- $E[T(n)] = O(n) + E\left[T\left(\frac{3n}{4}\right)\right] = \boldsymbol{O(n)}$

Fact

Since we are lucy with probably $\frac{1}{2}$, so the expected number of rounds it takes to become lucky is 2.

Evaluate Random Algorithm by Expected Running Time!

Other Viewpoints

- Worst Case Running Time
  - $O(n^2)$
- The Probability it runs in $O(n)$?

What if we do not want randomness?

Throw Randomness!

Function Select(S,k)

- Divide:
  - Pick a **random** value v among $x_1, x_2, x_3, \dots$.
  - Divide $x_1, x_2, x_3, \dots$ into three subsets:
    - L : $x < v$,
    - M : $x = v$,
    - R : $x > v$.
- Recurse:
  - Recurse the subset contains x^*.
    - If $k \leq |L|$, output Select(L,k);
    - If $|L| < k \leq |L| + |M|$, output v;
    - If $|L| + |M| < k$, output Select(R, k $-$ |L| $-$ |M|).

Throw Randomness!

Function Select(S,k)

- Divide:
  - Pick a ~~random~~ value v among $x_1, x_2, x_3, \dots$.
  - Divide $x_1, x_2, x_3, \dots$ into three subsets:
    - L : $x < v$,
    - M : $x = v$,
    - R : $x > v$.
- Recurse:
  - Recurse the subset contains x^*.
    - If $k \leq |L|$, output Select(L,k);
    - If $|L| < k \leq |L| + |M|$, output v;
    - If $|L| + |M| < k$, output Select(R, k $-$ |L| $-$ |M|).

Median of medians (1973)

Blum, M.; Floyd, R. W.; Pratt, V. R.; Rivest, R. L.; Tarjan, R. E.

Function Select(S,k)

- Divide:
  - Pick a **good pivot** value v among $x_1, x_2, x_3, \dots$.
  - Divide $x_1, x_2, x_3, \dots$ into three subsets:
    - L : $x < v$,
    - M : $x = v$,
    - R : $x > v$.
- Recurse:
  - Recurse the subset contains x^*.
    - If $k \leq |L|$, output Select(L,k);
    - If $|L| < k \leq |L| + |M|$, output v;
    - If $|L| + |M| < k$, output Select(R, k $- |\mathrm{L}| - |\mathrm{M}|$).

Trade-off

- The time of finding a good pivot.
- The quality of the pivot.
- We can find the median by sorting the array, but it takes too much time.
- We can find an arbitrary pivot in $O(1)$ but it may be too bad.
- Look at the recursive running time:
  - $T(n) = T(c \cdot n) + findPivot + O(n)$.

How to select a good pivot?

- Partition S into subsets with size 5.

How to select a good pivot?

- Partition S into subsets with size 5.

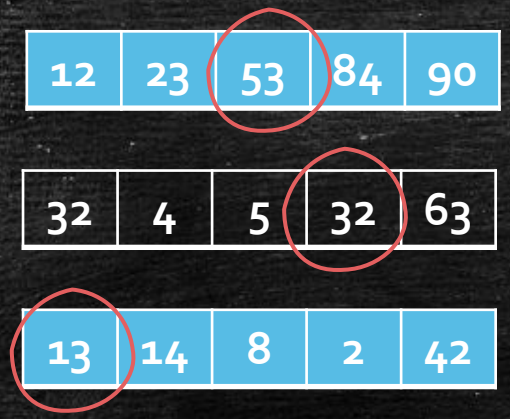
12	23	53	84	90

32	4	5	32	63

13	14	8	2	42

How to select a good pivot?

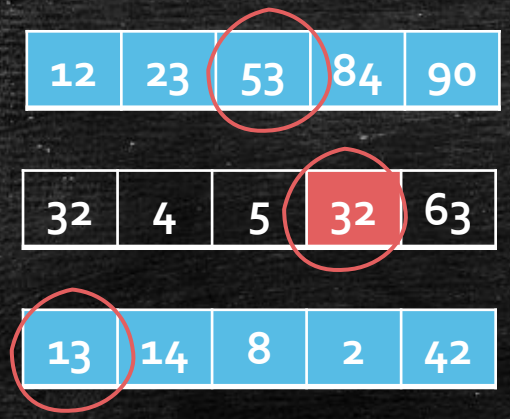
- Partition S into subsets with size 5.



- Find the medians of them: v_1, v_2, v_3
  - 53, 32, 13

How to select a good pivot?

- Partition S into subsets with size 5.



- Find the medians of them: v_1, v_2, v_3
  - 53, 32, 13
- Fix v to be the median of v_1, v_2, v_3
  - $v = 32$

How long it takes?

- Partition S into subsets with size 5.

$O(n)$

- Find the medians of them: v_1, v_2, v_3
  - 53, 32, 13

$O(n)$

- Fix v to be the median of v_1, v_2, v_3
  - $v = 32$

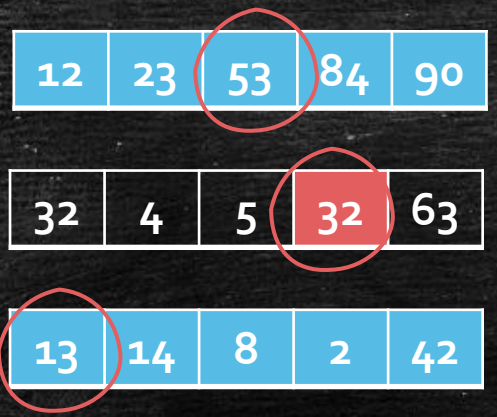
$T(n/5)$

Why it is good?

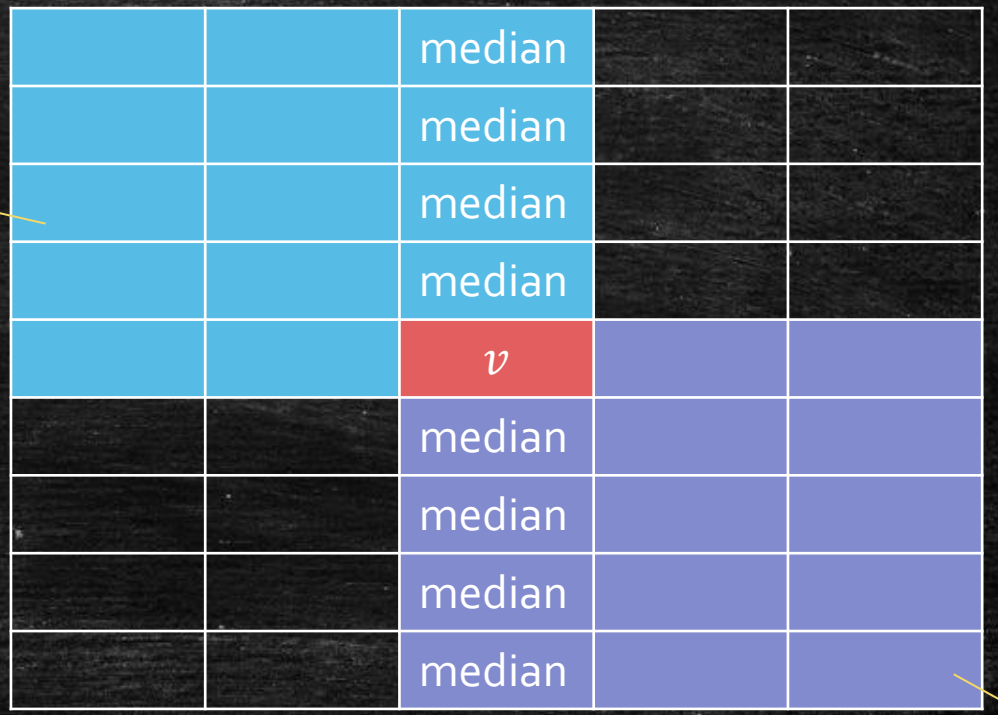
- It should be in the middle range
- Why? Two questions
  - How many integers should be no greater than v?
  - How many integers should be no less than v?

Answer them step by step

- Partition S into subsets



Smaller than v.



Larger than v.

- Answer
  - We have $\frac{n}{5}$ groups, so $\frac{n}{5}$ medians.
  - v is no greater than $n/10$ medians, no less than $n/10$ medians.
  - Each median is no greater than 2 integers, no less than 2 integers.
  - v is no greater than $\frac{3n}{10}$ integers, no less than $3n/10$ integers.

The running time

Function Select(S,k)

- Divide:
  - Pick a **good pivot** value v among $x_1, x_2, x_3, \dots$.
  - Divide $x_1, x_2, x_3, \dots$ into three subsets:
    - L : $x < v$,
    - M : $x = v$,
    - R : $x > v$.
- Recurse:
  - Recurse the subset contains x^*.
    - If $k \leq |L|$, output Select(L,k);
    - If $|L| < k \leq |L| + |M|$, output v;
    - If $|L| + |M| < k$, output Select(R, k $-$ |L| $-$ |M|).

$T\left(\frac{n}{5}\right) + O(n)$

$O(n)$

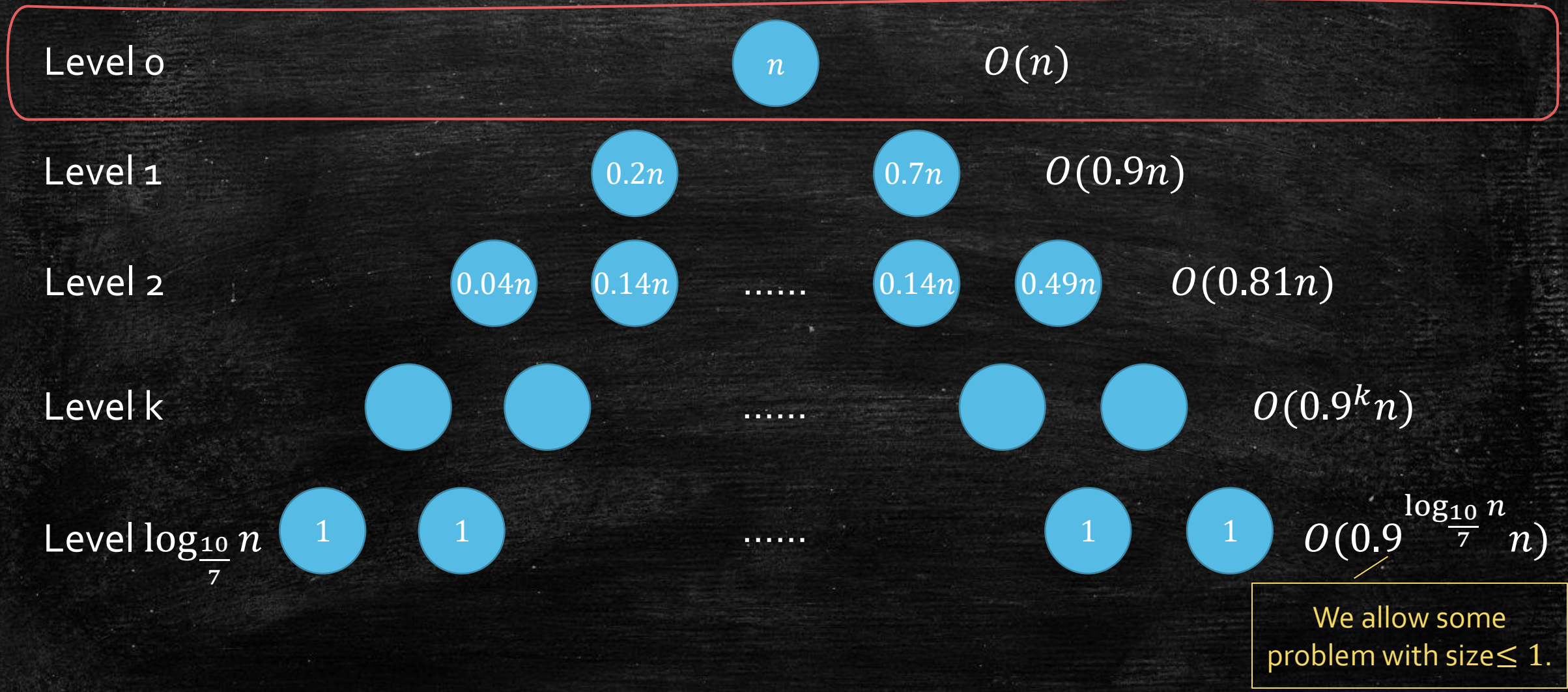
$T(|L|) \leq T(n - \frac{3}{10}n)$

$T(|R|) \leq T(n - \frac{3}{10}n)$

Guess time!

$$T(n) = T(0.2n) + T(0.7n) + O(n)$$

Guessing Time



Make a guess

$$\boldsymbol{T(n) = T(0.2n) + T(0.7n) + O(n)}$$

- Guess: $T(n) \leq Bn$!
- Try to prove it inductively
  - Basic step: $T(1) = 1 \leq Bn$
  - Inductive step:

$$\begin{aligned} T(n) &\leq T(0.2\mathrm{n}) + T(0.7\mathrm{n}) + Cn \\ &\leq 0.9Bn + Cn \\ &\leq Bn \end{aligned}$$

- We have $T(n) \leq 10Cn = O(n)$

Assume $O(n) \leq Cn$

It holds when $B \geq 10C$

如果还有时间的话

那就再来一个有趣的

Karatsuba 算法