Part8-Linear-Time Selection

Completed on	@2020/03/21
Key videos	https://www.coursera.org/learn/algorithms-divide-conquer/lecture/aqUNa/randomized-selection-algorithm https://www.coursera.org/learn/algorithms-divide-conquer/lecture/vtehr/deterministic-selection-algorithm-advanced-optional
Note	Introduction and analysis of two linear-time selection algorithms, RSelect and DSelect

Note

▼ Overview

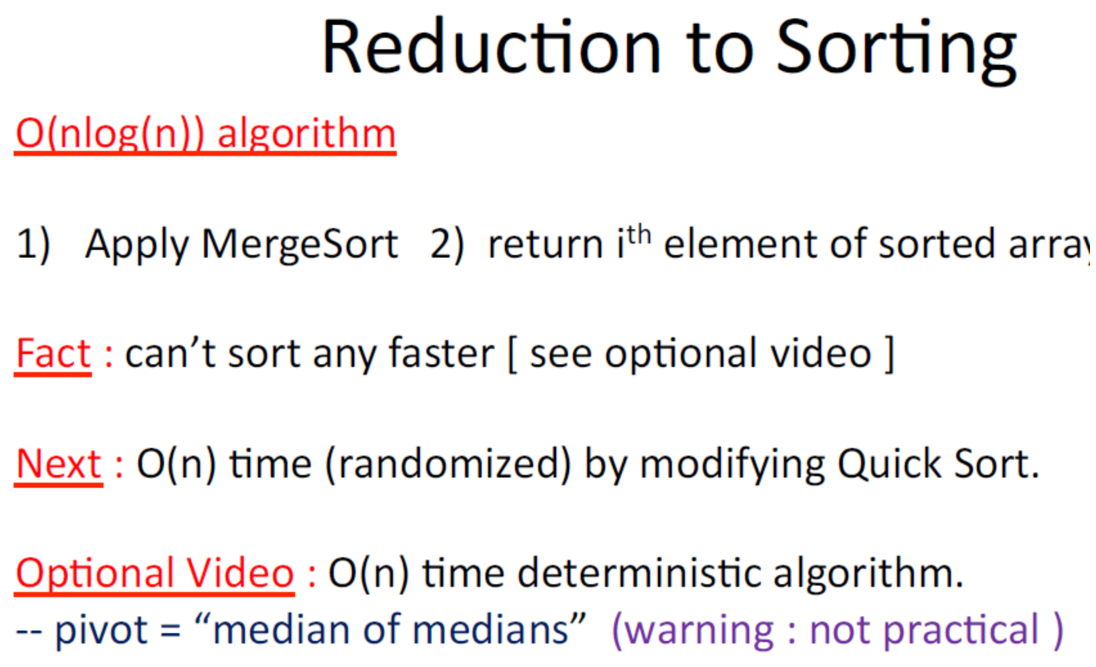
Part VIII --- LINEAR-TIME SELECTION (Required). These lectures study the problem of computing the ith smallest element of an input array (e.g., the median). It's easy to solve this problem in O(n log n) time using sorting, but we can do better! The required material goes over a super-practical randomized algorithm, very much in the spirit of QuickSort, that has *linear* expected running time. Don't forget that it takes linear time just to read the input! The analysis is somewhat different than what we studied for QuickSort, but is equally slick. Basically, there's a cool way to think about the progress the algorithm makes in terms of a simple coin-flipping experiment. Linearity of expectation (yes, it's back...) seals the deal.

Part VIII --- LINEAR-TIME SELECTION (Optional). I couldn't resist covering some advanced related material. The first is an algorithm that has more Turing Award-winning authors than any other algorithm that I know of. It is a deterministic (i.e., no randomization allowed) linear-time algorithm for the Selection problem, based on an ingenious "median-of-medians" idea for guaranteeing good pivot choices. (There are some accompanying lectures notes for this part, available for download underneath each video.) The second optional topic answers the question "can we do better?" for sorting, unfortunately in the negative. That is, a counting argument shows that there is no "comparison-based" sorting algorithm (like MergeSort, QuickSort, or HeapSort) with worst-case running time better than n log n.

▼ Randomized Selection - Algorithm

- 问题描述：找到一串数组中的第i位数

- 算法思路：借用排序的算法思路，MergeSort或者QuickSort ⇒ ⇒ 寻找pivot



Reduction to Sorting

O(nlog(n)) algorithm

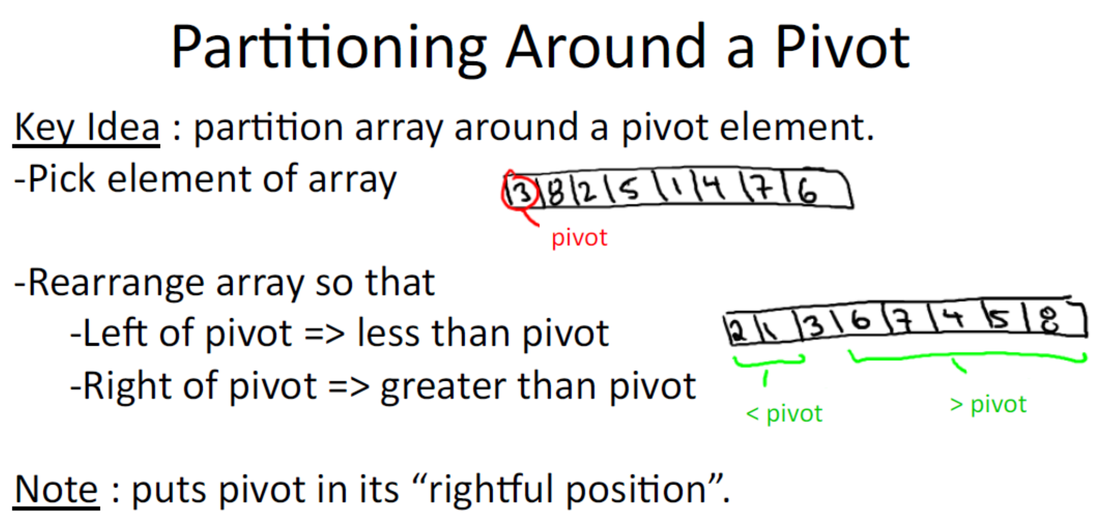
1) Apply MergeSort 2) return i^{th} element of sorted array

Fact : can't sort any faster [see optional video]

Next : O(n) time (randomized) by modifying Quick Sort.

Optional Video : O(n) time deterministic algorithm.
-- pivot = "median of medians" (warning : not practical)

- 关于partition的选择



Partitioning Around a Pivot

Key Idea : partition array around a pivot element.

-Pick element of array

-Rearrange array so that

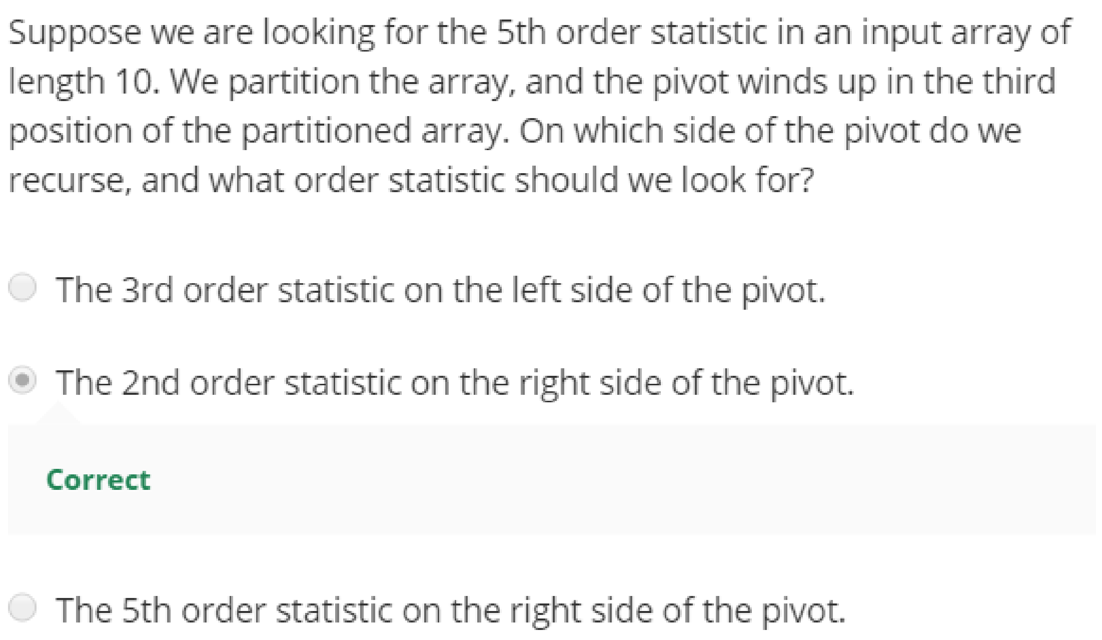
  -Left of pivot => less than pivot

  -Right of pivot => greater than pivot

Note : puts pivot in its "rightful position".

3 | 8 | 2 | 5 | 1 | 4 | 7 | 6
pivot
2 | 1 | 3 | 6 | 7 | 4 | 5 | 8
< pivot
> pivot

例如，选择第五顺位的数，若是初始pivot在第三顺位 ⇒ 子问题即是在右边数列中，找到第二顺位的数字！



Suppose we are looking for the 5th order statistic in an input array of length 10. We partition the array, and the pivot winds up in the third position of the partitioned array. On which side of the pivot do we recurse, and what order statistic should we look for?

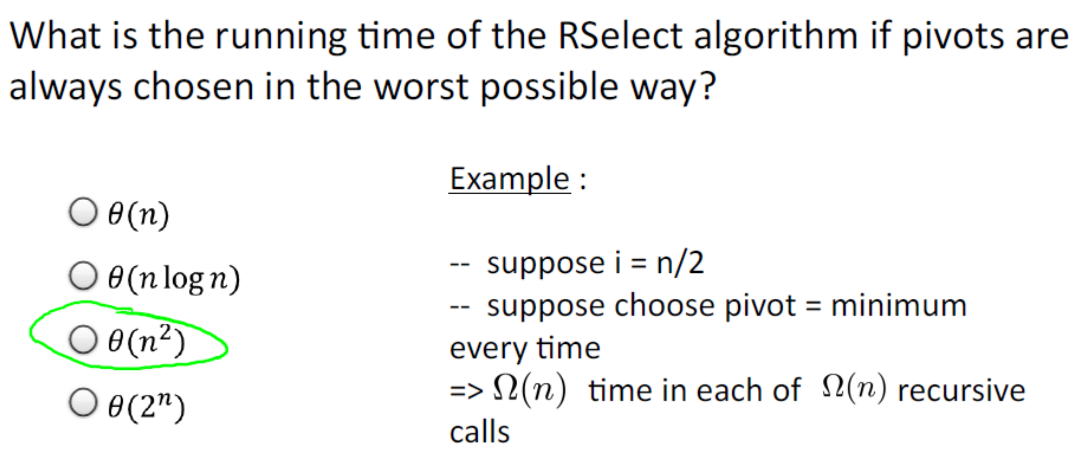
- The 3rd order statistic on the left side of the pivot.
- [✓ The 2nd order statistic on the right side of the pivot.]

  Correct

- The 5th order statistic on the right side of the pivot.

- RSelect算法质量的好坏取决于pivot的质量

  如果总是选取到最差质量的pivot，那么算法复杂度位O(n2)

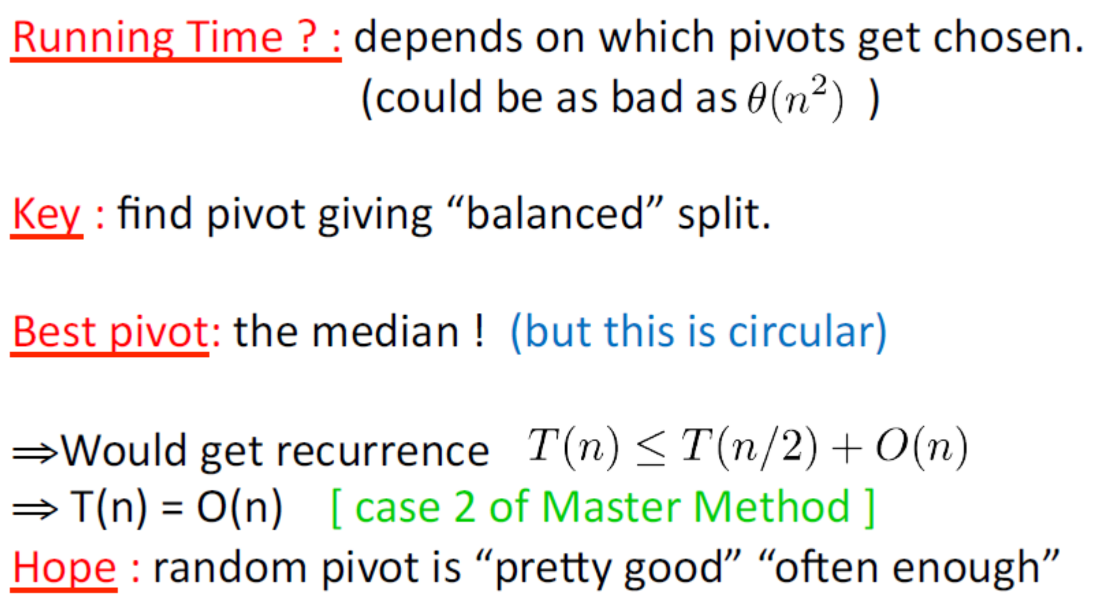

What is the running time of the RSelect algorithm if pivots are always chosen in the worst possible way?

- ○ $\theta(n)$
- ○ $\theta(n \log n)$
- [✓ ○ $\theta(n^2)$]
- ○ $\theta(2^n)$

Example :

-- suppose i = n/2
-- suppose choose pivot = minimum every time
=> $\Omega(n)$ time in each of $\Omega(n)$ recursive calls

- 时间复杂度分析


Running Time ? : depends on which pivots get chosen. (could be as bad as $\theta(n^2)$)

Key : find pivot giving "balanced" split.

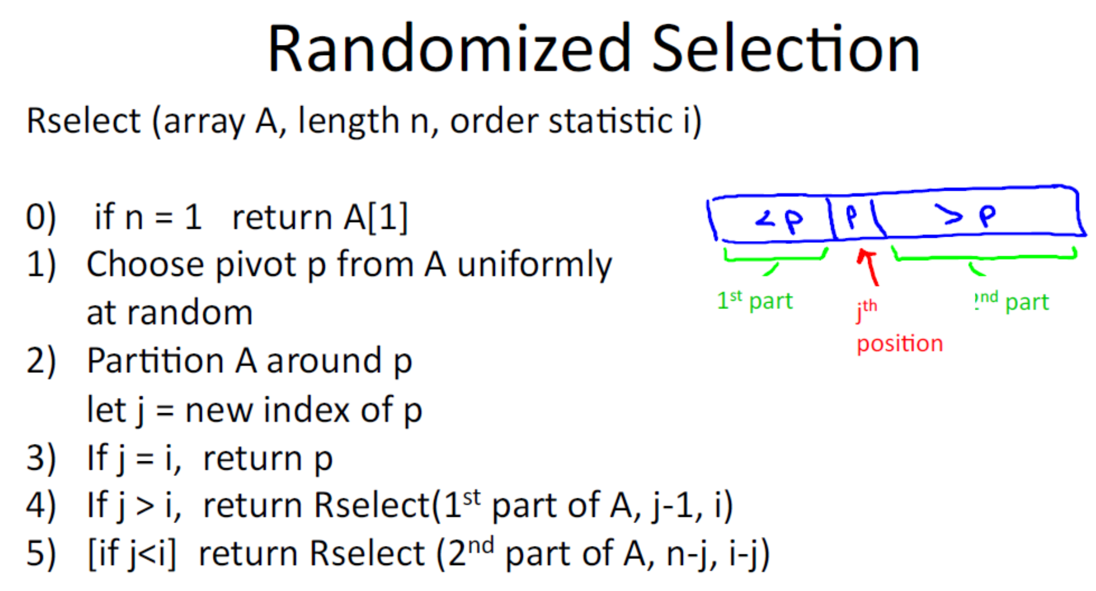
Best pivot: the median ! (but this is circular)

$\Rightarrow$Would get recurrence $T(n) \leq T(n/2) + O(n)$
$\Rightarrow$ T(n) = O(n) [case 2 of Master Method]
Hope : random pivot is "pretty good" "often enough"

随机选取下复杂度约为O(n)，整体分析思路接近于QuickSort，近似使用概率和期望去分析！

▼ Randomized Selection - Analysis

- 伪代码步骤



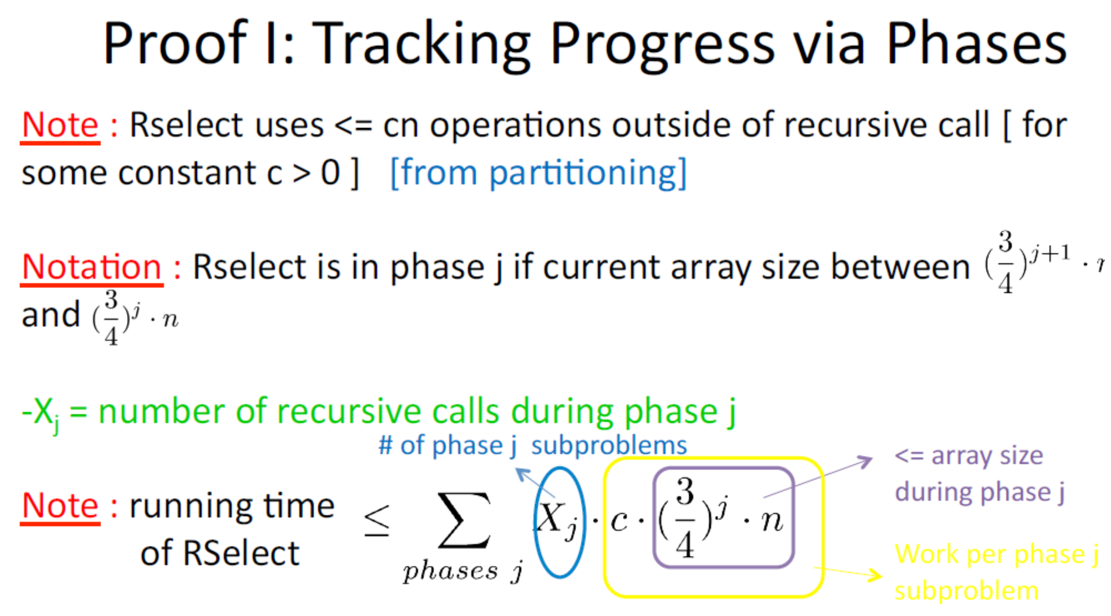
Randomized Selection

Rselect (array A, length n, order statistic i)

```
0)  if n = 1   return A[1]
1)  Choose pivot p from A uniformly
    at random
2)  Partition A around p
    let j = new index of p
3)  If j = i,  return p
4)  If j > i,  return Rselect(1st part of A, j-1, i)
5)  [if j<i]  return Rselect (2nd part of A, n-j, i-j)
```

<p
p
>p
1st part
jth
position
2nd part

- 证明一：对于3/4这个数字表示十分迷惑！不过从该页可以得到总体问题的上线（upper bound）



Proof I: Tracking Progress via Phases

Note : Rselect uses <= cn operations outside of recursive call [for some constant c > 0]   [from partitioning]

Notation : Rselect is in phase j if current array size between $(\frac{3}{4})^{j+1} \cdot n$ and $(\frac{3}{4})^{j} \cdot n$

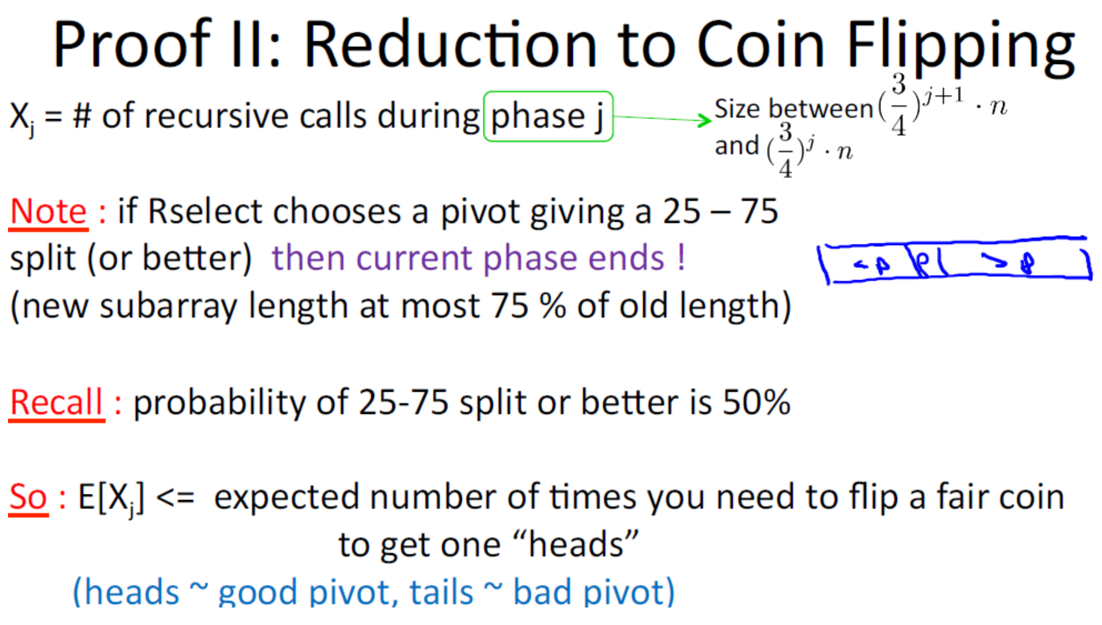
-X_j = number of recursive calls during phase j

Note : running time of RSelect

$$\leq \sum_{phases\ j} \boxed{X_j} \cdot \boxed{c \cdot \boxed{(\frac{3}{4})^j \cdot n}}$$

# of phase j subproblems
<= array size during phase j
Work per phase j subproblem

- 证明二：找到一个满足好的pivot的概率是50%



Proof II: Reduction to Coin Flipping

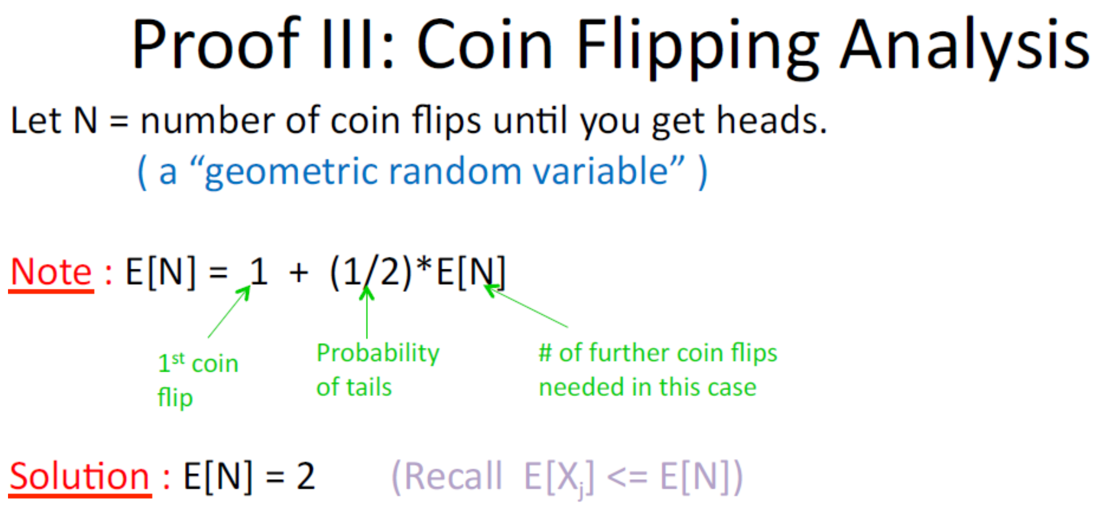
X_j = # of recursive calls during $\boxed{\text{phase j}}$ $\rightarrow$ Size between $(\frac{3}{4})^{j+1} \cdot n$ and $(\frac{3}{4})^{j} \cdot n$

Note : if Rselect chooses a pivot giving a 25 – 75 split (or better) then current phase ends !
(new subarray length at most 75 % of old length)

Recall : probability of 25-75 split or better is 50%

So : $E[X_j]$ <= expected number of times you need to flip a fair coin to get one "heads"
(heads ~ good pivot, tails ~ bad pivot)

- 证明三：计算期望需要的反转的次数



Proof III: Coin Flipping Analysis

Let N = number of coin flips until you get heads.
(a "geometric random variable")

Note : E[N] = 1 + (1/2)*E[N]

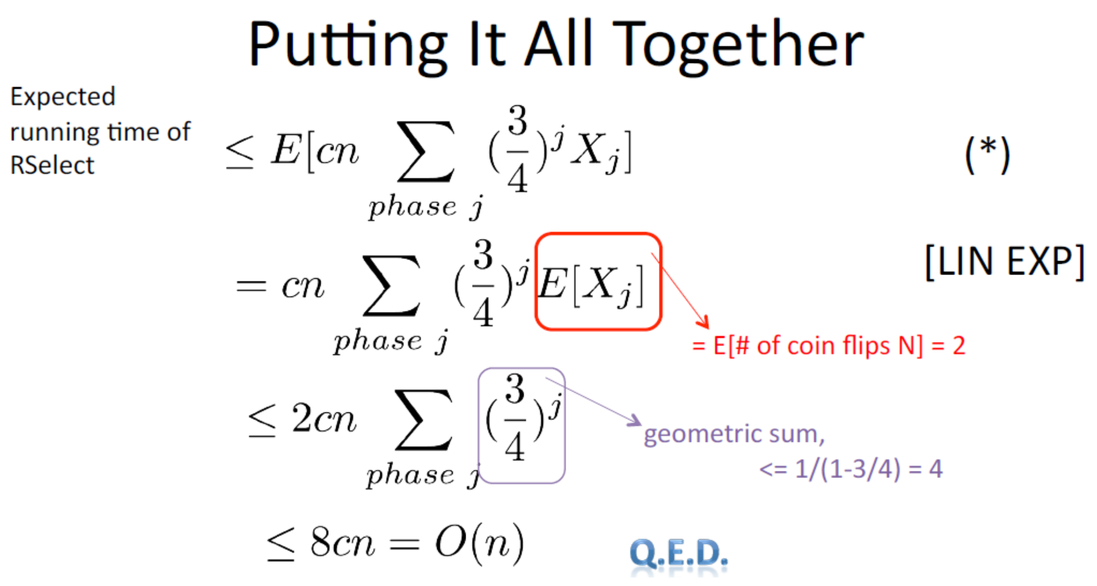
1st coin flip

Probability of tails

# of further coin flips needed in this case

Solution : E[N] = 2 (Recall $E[X_j]$ <= E[N])

- 结果：最终得到算法的复杂度为常数



Putting It All Together

Expected running time of RSelect

$$\leq E[cn \sum_{phase\ j} (\frac{3}{4})^j X_j] \qquad (*)$$

$$= cn \sum_{phase\ j} (\frac{3}{4})^j \boxed{E[X_j]} \qquad \text{[LIN EXP]}$$

= E[# of coin flips N] = 2

$$\leq 2cn \sum_{phase\ j} \boxed{(\frac{3}{4})^j}$$

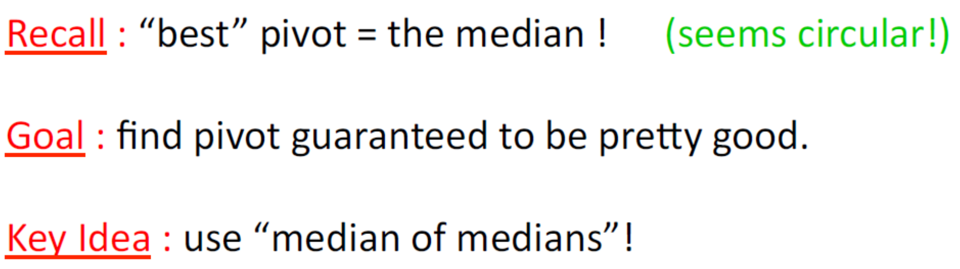
geometric sum, <= 1/(1-3/4) = 4

$$\leq 8cn = O(n)$$

Q.E.D.

▼ Deterministic Selection - Algorithm [Advanced - Optional]

- 思想：在RSelect的基础上，找到中点的中点

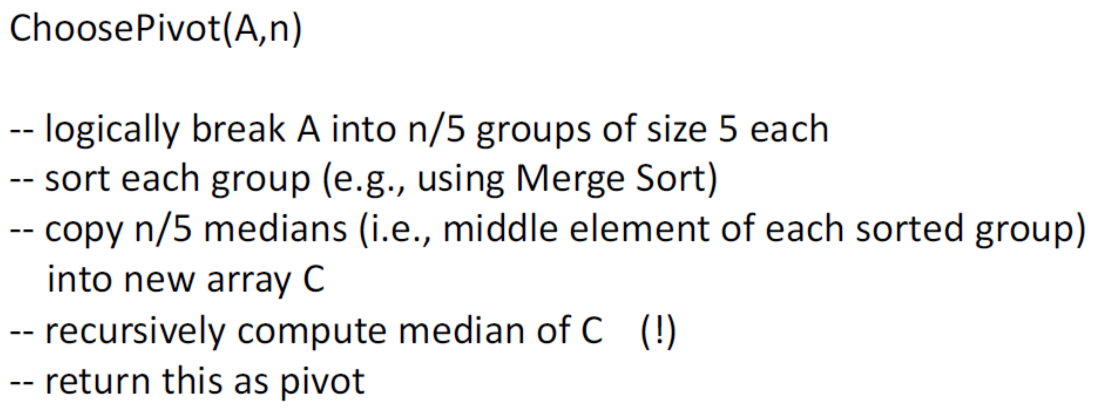


Recall : “best” pivot = the median ! (seems circular!)

Goal : find pivot guaranteed to be pretty good.

Key Idea : use “median of medians”!

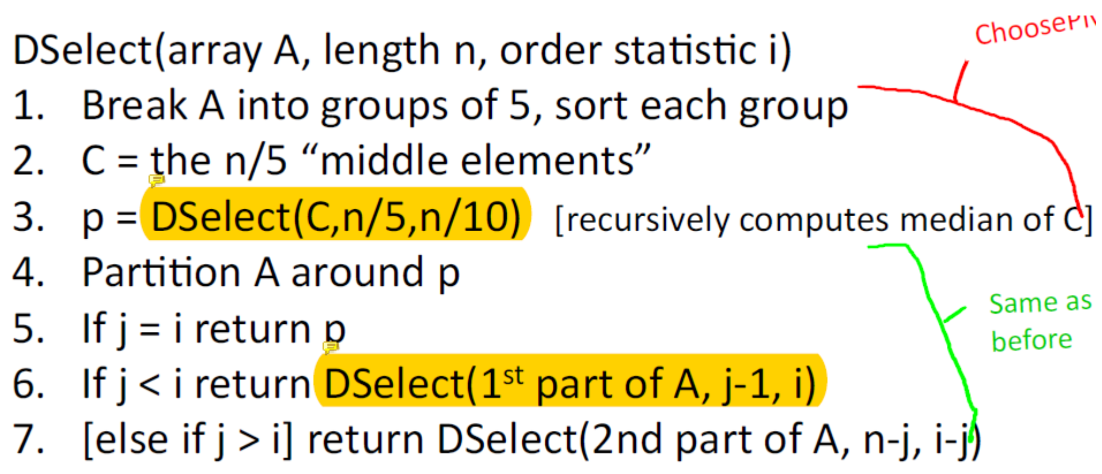
- ChoosePivot ⇒ 将group分为元素个数为5的n组，然后找到最中点



```
ChoosePivot(A,n)

-- logically break A into n/5 groups of size 5 each
-- sort each group (e.g., using Merge Sort)
-- copy n/5 medians (i.e., middle element of each sorted group)
   into new array C
-- recursively compute median of C   (!)
-- return this as pivot
```

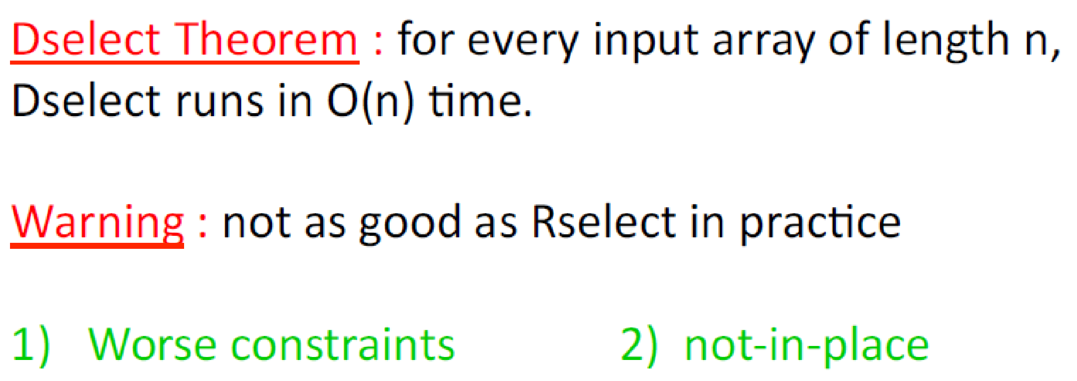
- 伪代码：相当于在RSelect基础上



```
DSelect(array A, length n, order statistic i)
1. Break A into groups of 5, sort each group
2. C = the n/5 "middle elements"
3. p = DSelect(C,n/5,n/10)  [recursively computes median of C]
4. Partition A around p
5. If j = i return p
6. If j < i return DSelect(1st part of A, j-1, i)
7. [else if j > i] return DSelect(2nd part of A, n-j, i-j)
```

ChoosePi
Same as before

- 缺点：约束条件更弱，需要额外的空间复杂度（not-in-place）



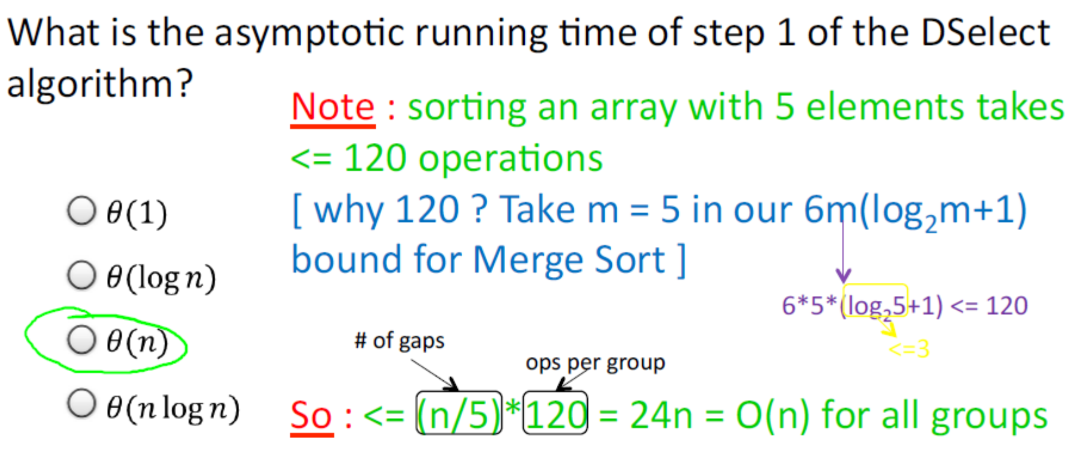
Dselect Theorem : for every input array of length n, Dselect runs in O(n) time.

Warning : not as good as Rselect in practice

1) Worse constraints 2) not-in-place

▼ Deterministic Selection - Analysis I [Advanced - Optional]

- 分析第一步：Break A into groups of 5, sort each group



What is the asymptotic running time of step 1 of the DSelect algorithm?

○ $\theta(1)$

○ $\theta(\log n)$

[✓ $\theta(n)$]

○ $\theta(n \log n)$

Note : sorting an array with 5 elements takes <= 120 operations

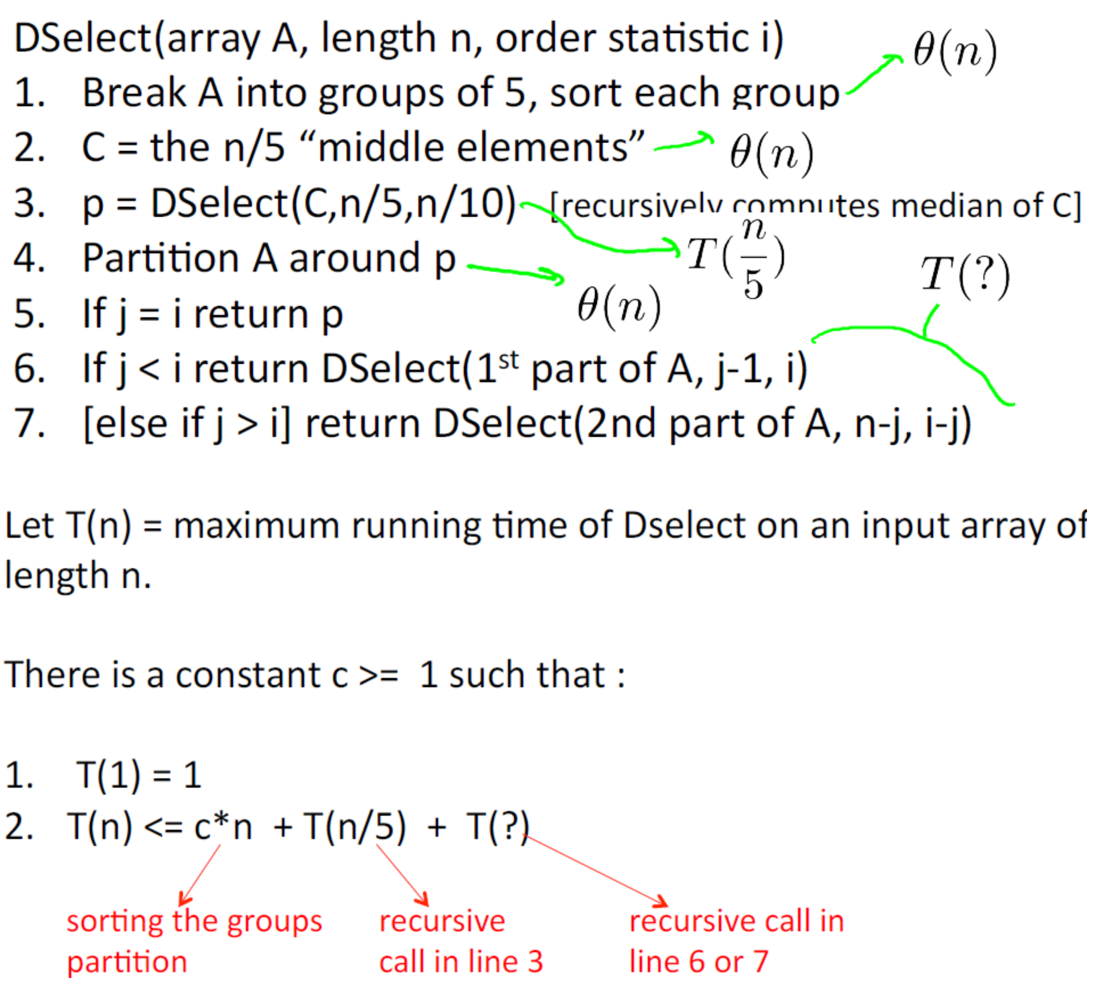
[why 120 ? Take m = 5 in our $6m(\log_2 m+1)$ bound for Merge Sort]

$6*5*(\log_2 5+1) <= 120$

<=3

So : <= $\underbrace{\boxed{(n/5)}}_{\text{\# of gaps}} * \underbrace{\boxed{120}}_{\text{ops per group}}$ = 24n = O(n) for all groups

- 复杂度分析



```
DSelect(array A, length n, order statistic i)
1. Break A into groups of 5, sort each group        $\theta(n)$
2. C = the n/5 "middle elements"                    $\theta(n)$
3. p = DSelect(C,n/5,n/10) [recursively computes median of C]   $T(\frac{n}{5})$
4. Partition A around p                             $\theta(n)$
5. If j = i return p
6. If j < i return DSelect(1st part of A, j-1, i)                $T(?)$
7. [else if j > i] return DSelect(2nd part of A, n-j, i-j)
```

Let T(n) = maximum running time of Dselect on an input array of length n.

There is a constant c >= 1 such that :

1. $T(1) = 1$
2. $T(n) \le c*n + T(n/5) + T(?)$

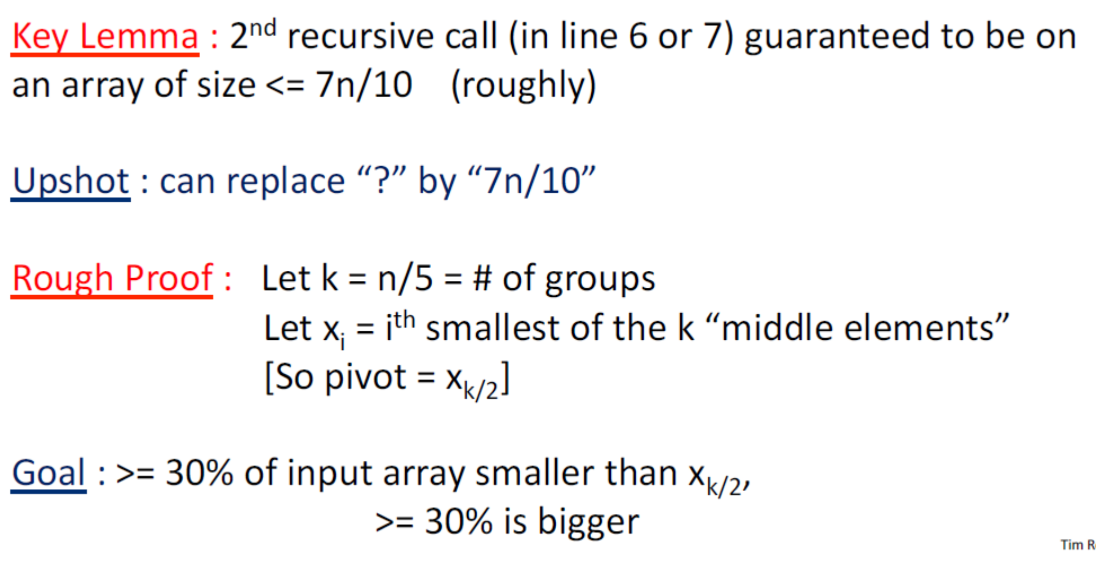
sorting the groups partition

recursive call in line 3

recursive call in line 6 or 7

主要的速度限制在于最后的6或7步骤

- 引理：借助近似估计与归纳证明 ⇒ 所选的pivot比30%数要大！



Key Lemma : 2nd recursive call (in line 6 or 7) guaranteed to be on an array of size <= 7n/10 (roughly)

Upshot : can replace "?" by "7n/10"

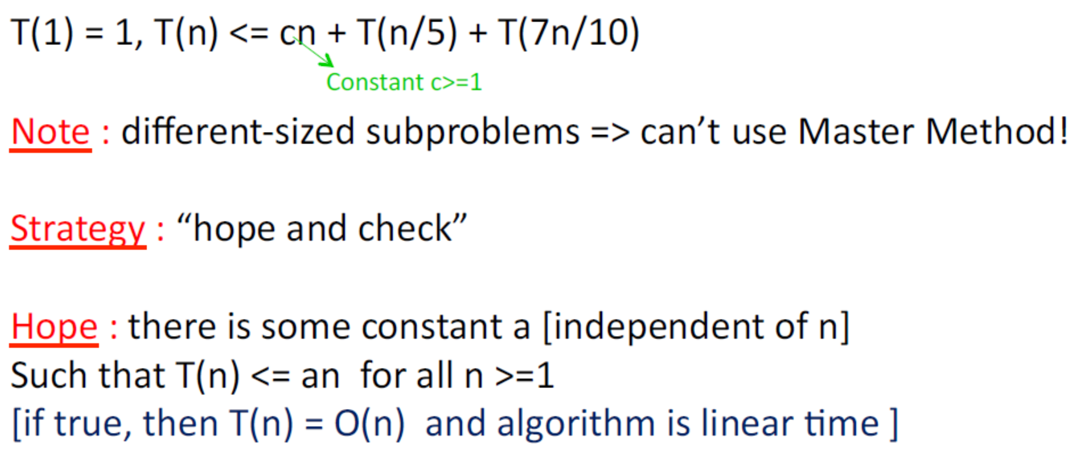
Rough Proof : Let k = n/5 = # of groups
Let x_i = ith smallest of the k "middle elements"
[So pivot = $x_{k/2}$]

Goal : >= 30% of input array smaller than $x_{k/2}$,
>= 30% is bigger

Tim R

▼ Deterministic Selection - Analysis II [Advanced - Optional]

- 归纳验证复杂度，由于子问题大小不同，因此不能直接运用Master Method



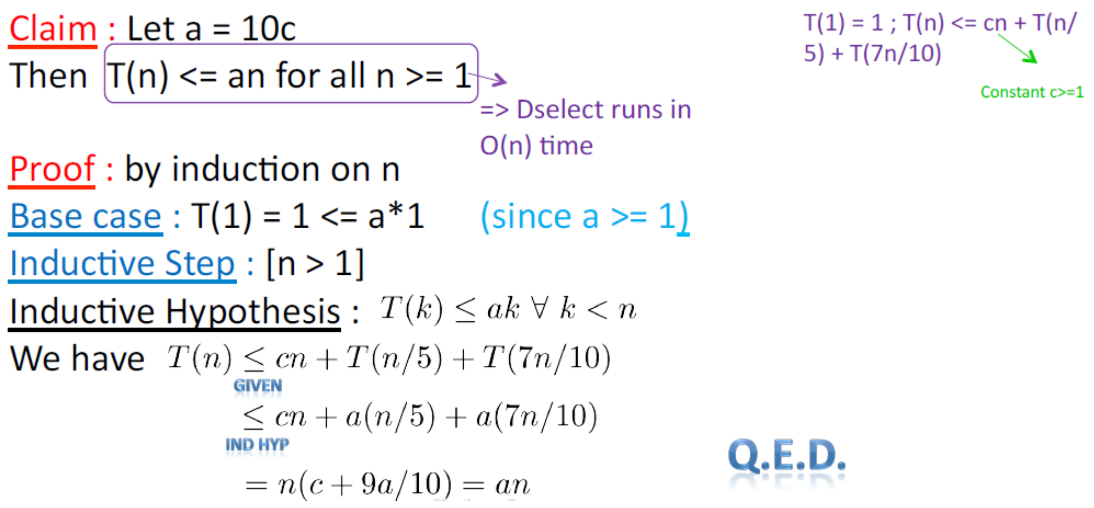
$T(1) = 1$, $T(n) \le \underset{\text{Constant } c>=1}{cn} + T(n/5) + T(7n/10)$

Note : different-sized subproblems => can't use Master Method!

Strategy : "hope and check"

Hope : there is some constant a [independent of n]
Such that T(n) <= an for all n >=1
[if true, then T(n) = O(n) and algorithm is linear time]

- 假设验证：



T(1) = 1 ; T(n) <= cn + T(n/5) + T(7n/10)
Constant c>=1

Claim : Let a = 10c
Then $\boxed{T(n) \le an \text{ for all } n \ge 1}$ => Dselect runs in O(n) time

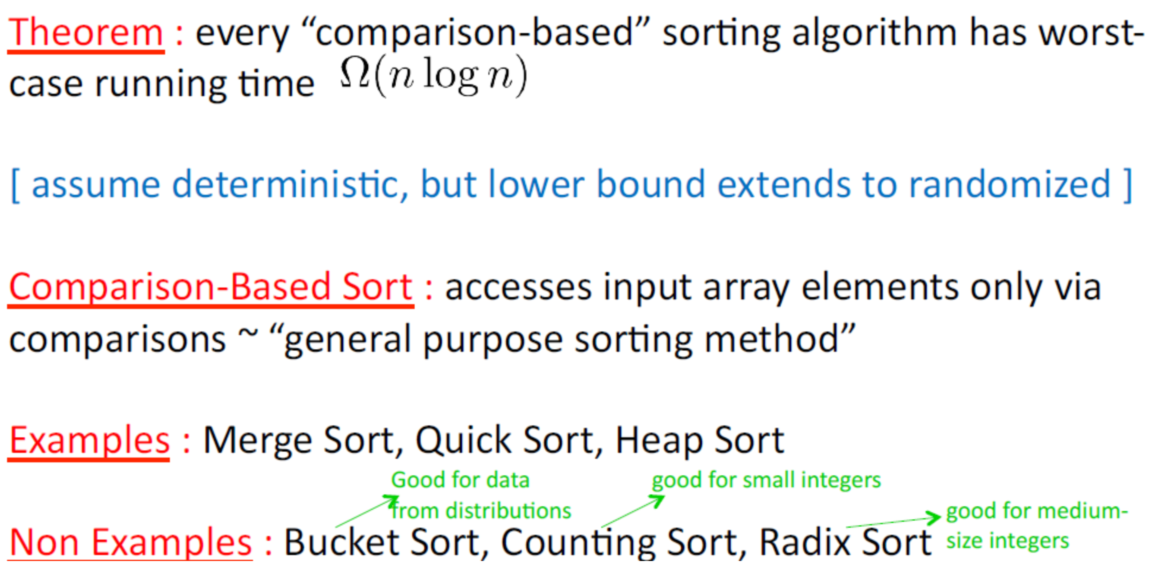
Proof : by induction on n
Base case : T(1) = 1 <= a*1 (since a >= 1)
Inductive Step : [n > 1]
Inductive Hypothesis : $T(k) \le ak \ \forall\ k < n$
We have $T(n) \underset{\text{GIVEN}}{\le} cn + T(n/5) + T(7n/10)$

$$\underset{\text{IND HYP}}{\le} cn + a(n/5) + a(7n/10)$$

$$= n(c + 9a/10) = an$$

Q.E.D.

▼ Omega(n log n) Lower Bound for Comparison-Based Sorting [Advanced - Optional]



Theorem : every "comparison-based" sorting algorithm has worst-case running time $\Omega(n \log n)$

[assume deterministic, but lower bound extends to randomized]

Comparison-Based Sort : accesses input array elements only via comparisons ~ "general purpose sorting method"

Examples : Merge Sort, Quick Sort, Heap Sort

Non Examples : Bucket Sort, Counting Sort, Radix Sort

Good for data from distributions

good for small integers

good for medium-size integers

References

- 选择算法

[Selection algorithm](https://en.wikipedia.org/wiki/Selection_algorithm)
https://en.wikipedia.org/wiki/Selection_algorithm

[选择算法](https://zh.wikipedia.org/wiki/%E9%80%89%E6%8B%A9%E7%AE%97%E6%B3%95)
https://zh.wikipedia.org/wiki/%E9%80%89%E6%8B%A9%E7%AE%97%E6%B3%95

- 随机化算法

[Randomized algorithm](https://en.wikipedia.org/wiki/Randomized_algorithm#Quicksort)
https://en.wikipedia.org/wiki/Randomized_algorithm#Quicksort