

# VE281

Data Structures and Algorithms

Linear Time Selection

# Outline

- Randomized selection algorithm
- Deterministic selection algorithm

# The Selection Problem

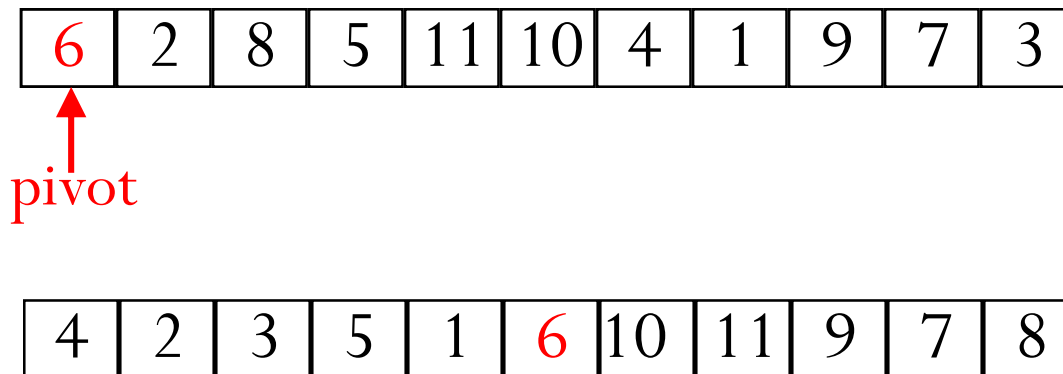
- Input: array  $A$  with  $n$  distinct numbers and a number  $i$ 
  - “**Distinct**” for simplicity
- Output:  $i$ -th smallest element in the array
  - Assume index starts from 1
- Example:  $A = (6, 3, 5, 4, 2)$ ,  $i = 3$ 
  - Should return 4
- Special cases
  - $i = 1$ : the smallest item. Runtime:  $O(n)$
  - $i = n$ : the largest item. Runtime:  $O(n)$
  - $i = n/2$ : the median

# Solution: Reduction to Sorting

- Step 1: Do merge sort
- Step 2: output the  $i$ -th element of the sorted array
- Time complexity is  $O(n \log n)$
- Can we do better?
  - This essentially asks whether selection is **fundamentally easier** than sorting
  - Answer: Yes!
  - We will show an  $O(n)$  time randomized algorithm by modifying quick sort
  - Also will show an  $O(n)$  time deterministic algorithm (However, not as practical as the randomized algorithm)

# Recall: Partitioning in Quick Sort

- Pick a pivot
- Put all elements  $<$  pivot to the left of pivot
- Put all elements  $\geq$  pivot to the right of pivot
- Move pivot to its correct place in the array



# Basic Idea

- Suppose we are looking for 6<sup>th</sup> smallest item in an array of length 12. We do partition.
  - Suppose the pivot is at position 4. Then we only need to focus on the sub-array right of the pivot and look for the 2<sup>nd</sup> item in the array
  - Suppose the pivot is at position 8. Then we only need to focus on the sub-array left of the pivot and look for the 6<sup>th</sup> item in the array
  - In both cases, recurse!

# Randomized Selection

```
Rselect(int A[], int n, int i) {  
    // find i-th smallest item of array A of size n  
    if(n == 1) return A[1];  
    Choose pivot p from A uniformly at random;  
    Partition A using pivot p;  
    Let j be the index of p;  
    if(j == i) return p;  
    if(j > i) return Rselect(1st part of A, j-1, i);  
    else return Rselect(2nd part of A, n-j, i-j);  
}
```

# Runtime of Rselect

- The runtime for **a fixed input array** and a value  $i$ .
- Depends on the quality of the chosen pivots
- Consider  $i = n/2$ . What is a worst pivot sequence and what is the worst runtime?
  - $\Theta(n^2)$
- Best runtime for an arbitrary  $i$ ?
  - $\Theta(n)$ . Your random selection of pivot directly gives you the  $i$ -th smallest item. The time is due to the partition.
- Average runtime?



# Average Runtime of Rselect

- Theorem: for every input array of length  $n$ , the average runtime of Rselect is  $O(n)$ 
  - Holds for every input data (no assumption on data)
  - “Average” is over random pivot choices made by the algorithm

# Average Runtime Analysis

- Note: Rselect uses  $\leq cn$  operations outside of recursive call (from partitioning)
- Observation: the length of the array the algorithm works on decreases
- Definition: We say **Rselect is in phase  $j$**  if current array size is between  $(\frac{3}{4})^{j+1}n$  and  $(\frac{3}{4})^j n$
- $X_j$  denote the number of recursive calls in phase  $j$
- $runtime \leq \sum_j X_j \cdot c \cdot (\frac{3}{4})^j n$  We need to further get  $E[X_j]$

$$E[runtime] \leq E \left[ \sum_j X_j \cdot c \cdot \left(\frac{3}{4}\right)^j n \right] = cn \sum_j \left(\frac{3}{4}\right)^j E[X_j]$$

# Average Runtime Analysis

- If Rselect chooses a pivot so that the **left sub-array**'s size is  $am$ , where  $a \in [\frac{1}{4}, \frac{3}{4}]$  and  $m$  is the old length, then the current phase ends
  - Because new sub-array length is at most 75% of the old length
  - **“Good pivot”**
- What is the probability of  $a \in [\frac{1}{4}, \frac{3}{4}]$  (i.e., good pivot)?
  - Answer: 0.5
- Claim:  $E[X_j] \leq$  Expected number of times you need to flip a fair coin to get a “head”
  - Heads: good pivot; tails: bad pivot
  - As mentioned above, if you get a “good pivot”, the current phase ends

# Coin Flipping Analysis

- Let  $N$  be the number of coin flips until you get heads
  - $N$  is a geometric random variable:  $P(N = k) = \frac{1}{2^k}$ ,  $k = 1, 2, \dots$

#flips when 1<sup>st</sup> is head    #flips when 1<sup>st</sup> is tail

- $$E[N] = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot (1 + E[N]) \Rightarrow E[N] = 2$$

Prob. 1<sup>st</sup> flip is head

Prob. 1<sup>st</sup> flip is tail

Therefore,  $E[X_j] \leq E[N] = 2$

# Average Runtime Analysis

$$\begin{aligned} E[\text{runtime}] &\leq E \left[ \sum_j X_j \cdot c \cdot \left(\frac{3}{4}\right)^j n \right] \\ &= cn \sum_j \left(\frac{3}{4}\right)^j E[X_j] \leq 2cn \sum_j \left(\frac{3}{4}\right)^j \leq 2cn \frac{1}{1 - \frac{3}{4}} \\ &= 8cn = O(n) \end{aligned}$$

# Outline

- Randomized selection algorithm
- Deterministic selection algorithm

# A Good Pivot

- Best pivot: the median
  - But, this is a circular problem
- Goal: find pivot guaranteed to be good enough
- Idea: use “median of medians”

# A Deterministic ChoosePivot

## **ChoosePivot(A, n)**

- A subroutine called by the deterministic selection algorithm
- Steps:
  1. Break A into  $n/5$  groups of size 5 each
  2. Sort each group (e.g., use insertion sort)
  3. Copy  $n/5$  medians into new array C
  4. Recursively compute median of C
    - By calling the deterministic selection algorithm!
  5. Return the median of C as pivot



# Deterministic Selection Algorithm

```
Dselect(int A[], int n, int i) {  
    // find i-th smallest item of array A of size n  
    if(n == 1) return A[1];  
    Break A into groups of 5, sort each group;  
    C = n/5 medians;  
    p = Dselect(C, n/5, n/10);           ChoosePivot  
    Partition A using pivot p;  
    Let j be the index of p;  
    if(j == i) return p;  
    if(j > i) return Dselect(1st part of A, j-1, i);  
    else return Dselect(2nd part of A, n-j, i-j);  
}
```

Same as  
before

The function has two recursive calls

# Runtime of Dselect

- Theorem: For every input array of length  $n$ , Dselect runs in  $O(n)$  time
- Warning: not as good as Rselect in practice
  - Worse constants
  - Not-in-place

# Runtime of Dselect

```
Dselect(int A[], int n, int i) {  
    // find i-th smallest item of array A of size n  
    1 if(n == 1) return A[1];  
    2 Break A into groups of 5, sort each group;  
    3 C = n/5 medians;  
    4 p = Dselect(C, n/5, n/10);  
    5 Partition A using pivot p;  
    6 Let j be the index of p;  
    7 if(j == i) return p;  
    8 if(j > i) return Dselect(1st part of A, j-1, i);  
    9 else return Dselect(2nd part of A, n-j, i-j);  
}
```

- What's the runtime of step 2?

$\Theta(n)$ , because sorting array of 5 elements takes constant time

# Runtime of Dselect

Assume the runtime is  $T(n)$

```
Dselect(int A[], int n, int i) {  
  // find i-th smallest item of array A of size n  
  1 if(n == 1) return A[1];  
  2 Break A into groups of 5, sort each group;  $\Theta(n)$   
  3 C = n/5 medians;  $\Theta(n)$   
  4 p = Dselect(C, n/5, n/10);  $T(n/5)$   
  5 Partition A using pivot p;  $\Theta(n)$   
  6 Let j be the index of p;  
  7 if(j == i) return p;  
  8 if(j > i) return Dselect(1st part of A, j-1, i);  
  9 else return Dselect(2nd part of A, n-j, i-j);  
}
```

$\left. \begin{array}{l} \text{8} \\ \text{9} \end{array} \right\} T(?)$

# Recurrence

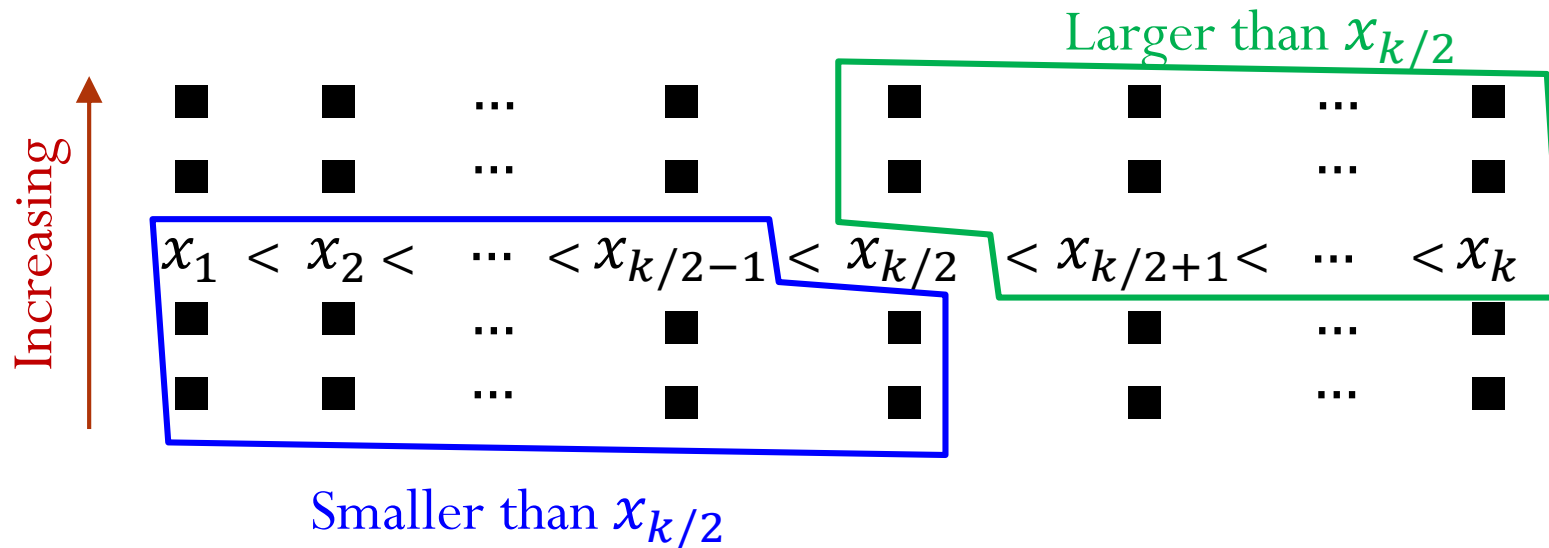
- There exists a positive constant  $C$  such that
  - $T(1) \leq c$
  - $T(n) \leq cn + T\left(\frac{n}{5}\right) + T(?)$
- The next question is what is the size of the array of the second recursive call

# Lemma on Size

- Lemma: 2<sup>nd</sup> recursive call guaranteed to be on an array of size  $\leq 0.7n$  (roughly)
- (Rough) proof:
  - Let  $k = n/5$ : number of groups
  - Let  $x_i$  be the  $i$ -th smallest of the  $k$  medians
  - Thus, the pivot is  $x_{k/2}$
  - Goal
    - $\geq 30\%$  of input array smaller than  $x_{k/2}$
    - $\geq 30\%$  of input array larger than  $x_{k/2}$

# Proof of Lemma

- Imagine we layout elements of A in a 2-D grid



- At least  $\sim (3/5) * (1/2) = 30\%$  elements smaller than  $x_{k/2}$
- At least  $\sim 30\%$  elements larger than  $x_{k/2}$
- Result: Number of elements  $< x_{k/2}$  is in between 30% and 70%. The same for number of elements  $> x_{k/2}$

# Example

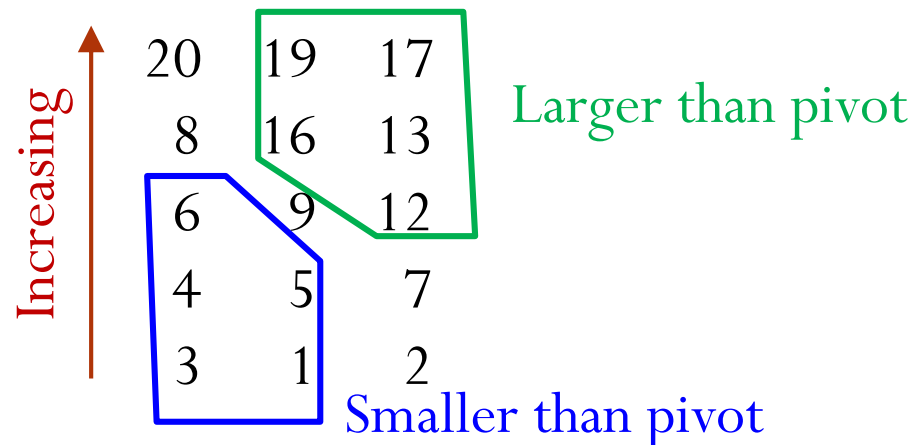
- Input:

7, 2, 17, 12, 13 | 8, 20, 4, 6, 3 | 19, 1, 9, 5, 16

- After sorting each group of 5 elements

2, 7, 12, 13, 17 | 3, 4, 6, 8, 20 | 1, 5, 9, 16, 19

← pivot





# Recurrence

- There exists a positive constant  $c$  such that
  - $T(1) \leq c$
  - $T(n) \leq cn + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right)$
- Note: different-sized sub-problems. Cannot use master method!
- How can we solve this?
  - Strategy: Hope and check
- Hope: there is a constant  $a$  (independent of  $n$ ) such that  $T(n) \leq an$  for all  $n > 1$ 
  - Then  $T(n) = O(n)$
- We choose  $a = 10c$

# Proof $T(n) = O(n)$

- Claim: suppose there exists a positive constant  $c$  such that

1.  $T(1) \leq c$

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

Then  $T(n) \leq 10cn$

- Proof by induction

- Base case:  $T(1) \leq 10c$

- Inductive step: inductive hypothesis  $T(k) \leq 10ck, \forall k < n$ .

Then

$$T(n) \leq cn + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \leq cn + 2cn + 7cn = 10cn$$

Dselect runs in linear time