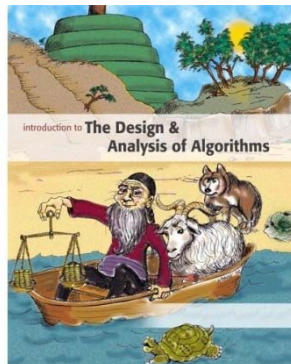




Introduction to

Algorithm Design and Analysis

[7] Selection



Yu Huang

<http://cs.nju.edu.cn/yuhuang>
Institute of Computer Software
Nanjing University



In the last class...

- **MergeSort**
 - Design
 - Cost – time & space
- **Lower bounds for *comparison-based sorting***
 - Worst-case
 - Average-case



The Selection

- **Selection – warm-ups**
 - Finding *max* and *min*
 - Finding the *second largest* key
- **Adversary argument** and lower bound
- **Selection – select the *median***
 - Expected linear time
 - Worst-case linear time
- **A Lower Bound for Finding the Median**



The Selection Problem

- **Problem definition**
 - Suppose E is an array containing n elements with keys from some linearly order set, and let k be an integer such that $1 \leq k \leq n$. The selection problem is to find an element with the k^{th} smallest key in E .
- **Special cases**
 - Find the max/min – $k=n$ or $k=1$
 - Find the *median* ($k = \frac{n}{2}$)



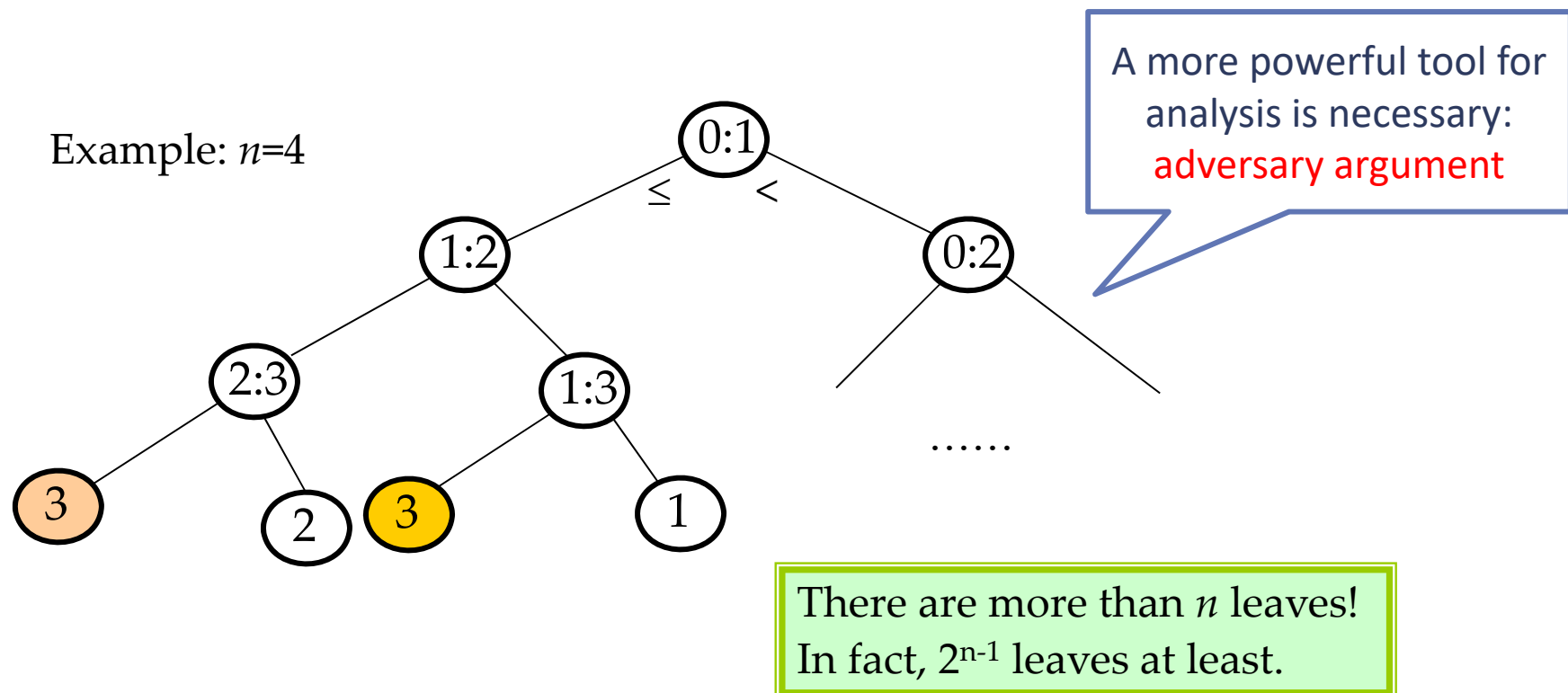
Lower Bound of Finding the Max

- For **any** algorithm \mathcal{A} that can compare and copy numbers exclusively, in the worst case, \mathcal{A} cannot do fewer than $n-1$ comparisons to find the largest entry in an array with n entries.
 - Proof: an array with n distinct entries is assumed. We can exclude a specific entry from being the largest entry only after it is determined to be “loser” to at least one entry. So, $n-1$ entries must be “losers” in comparisons done by the algorithm. However, each comparison has only one loser, so at least $n-1$ comparisons must be done.



Decision Tree and Lower Bound

Since the decision tree for the selection problem must have at least n leaves, the height of the tree is at least $\lceil \log n \rceil$. **It's not a good lower bound.**



Finding *max* and *min*

- **The strategy**
 - Pair up the keys, and do $n/2$ comparisons (if n odd, having $E[n]$ uncompared);
 - Doing findMax for larger key set and findMin for small key set respectively (if n odd, $E[n]$ included in both sets)
- **Number of comparisons**
 - For even n : $n/2 + 2(n/2 - 1) = 3n/2 - 2$
 - For odd n : $(n-1)/2 + 2((n-1)/2 + 1 - 1) = \lceil 3n/2 \rceil - 2$

How to prove this lower bound?

**Adversary
Argument !**



Unit of Information

- **Max and Min**
 - That x is *max* can only be known when it is sure that every key other than x has **lost some comparison**.
 - That y is *min* can only be known when it is sure that every key other than y has **win some comparison**.
- **Each win or loss is counted as one unit of information**
 - *Any* algorithm must have at least $2n-2$ units of information to be sure of specifying the *max* and *min*.



Adversary Strategy

Status of keys x and y			Units of new
Compared by an algorithm	Adversary response	New status	information
N,N	$x > y$	W,L	2
W,N or WL,N	$x > y$	W,L or WL,L	1
L,N	$x < y$	L,W	1
W,W	$x > y$	W,WL	1
L,L	$x > y$	WL,L	1
W,L or WL,L or W,WL	$x > y$	No change	0
WL,WL	Consistent with Assigned values	No change	0

The principle: let the key win if it never lose, or,
let the key lose if it never win, and

change one value if necessary



Lower Bound by the Adversary Argument

- Construct an input to force *the* algorithm to do more comparisons as possible
 - To give away as few as possible units of new information with each comparison.
 - It can be achieved that 2 units of new information are given away only when the status is N,N.
 - It is *always* possible to give adversary response for other status so that at most one new unit of information is given away, *without any inconsistencies*.
- So, the ***Lower Bound*** is $n/2 + n - 2$ (for even n)

$$\frac{n}{2} \times 2 + (n - 2) \times 1 = 2n - 2$$



An Example

Comparison	x_1		x_2		x_3		x_4		x_5		x_6	
	S	V	S	V	S	V	S	V	S	V	S	V
x_1, x_2					N	*	N	*	N	*	N	*
x_1, x_5									L	5		
x_3, x_4												
x_3, x_6												
x_3, x_1												
x_2, x_4												
x_5, x_6									WL	5	L	3
x_6, x_4							L	2			WL	3

Now, x_3 is the only one which never loses, so, Max is x_3

Now, x_3 is the only

8 comparisons!
The lower bound is 7.

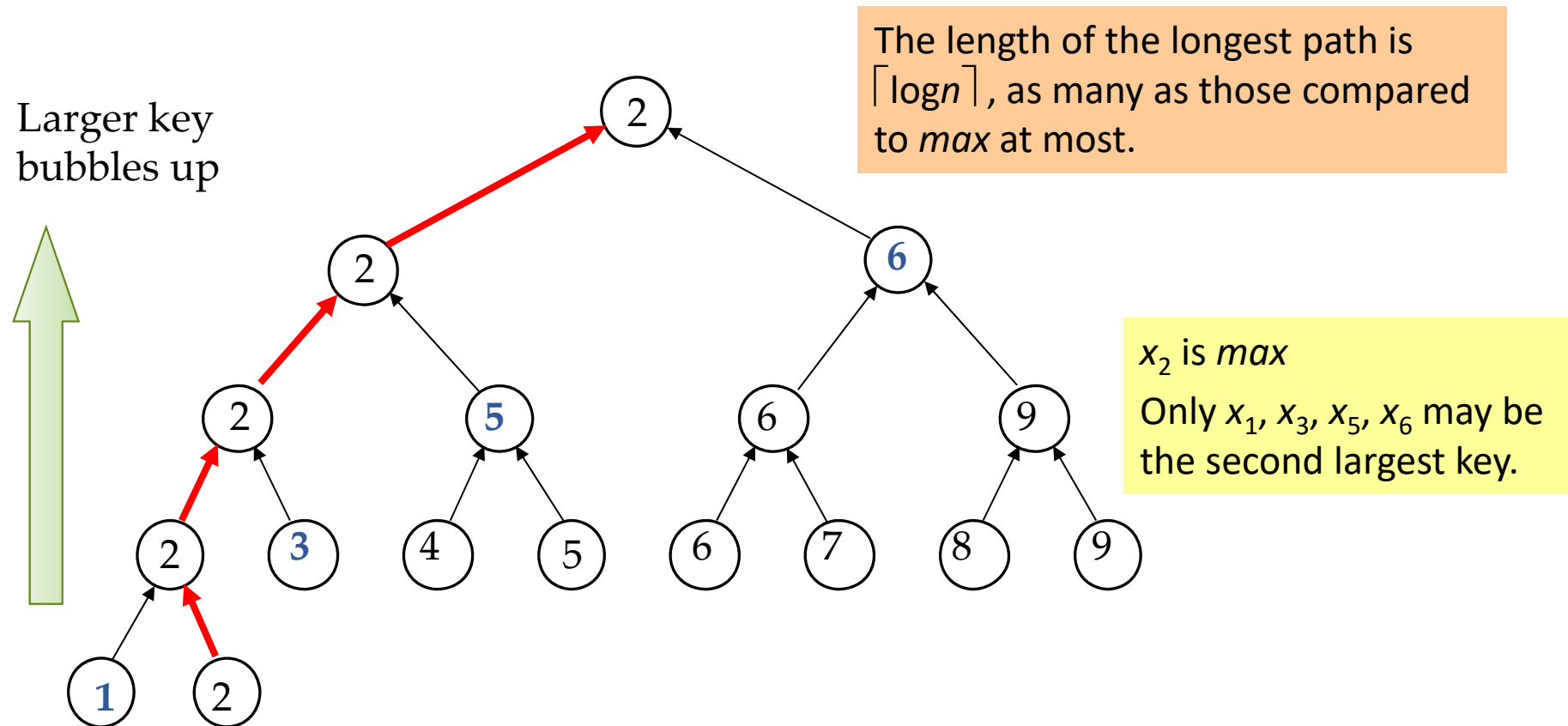


Find the 2nd Largest Key

- **Brute force - using FindMax twice**
 - Need $2n-3$ comparisons.
- **For a better algorithm**
 - Collect some useful information from the first FindMax
- **Observations**
 - The key which **loses to a key other than max** cannot be the 2nd largest key.
 - To check “whether you lose to max?”



Tournament for the 2nd Largest Key



Analysis of Finding the 2nd

- Any algorithm that finds *secondLargest* must also find *max* before. $(n-1)$
- The *secondLargest* can only be in those which lose directly to *max*.
- On its path along which bubbling up to the root of tournament tree, *max* beat $\lceil \log n \rceil$ keys at most.
- Pick up *secondLargest* $(\lceil \log n \rceil - 1)$
- Total cost: $n + \lceil \log n \rceil - 2$



Lower Bound by Adversary

- **Theorem**

- Any algorithm (that works by comparing keys) to find the second largest in a set of n keys must do at least $n + \lceil \log n \rceil - 2$ comparisons in the worst case.

- **Proof**

- There is an adversary strategy that can force any algorithm that finds *secondLargest* to compare *max* to $\lceil \log n \rceil$ distinct keys.



Weighted Key

- Assigning a weight $w(x)$ to each key
 - The initial values are all 1.
- Adversary strategy

Note: for one comparison, the weight increasing is no more than doubled.

Case	Adversary reply	Updating of weights
$w(x) > w(y)$	$x > y$	$w(x) := w(x) + w(y); w(y) := 0$
$w(x) = w(y) > 0$	$x > y$	$w(x) := w(x) + w(y); w(y) := 0$
$w(y) > w(x)$	$y > x$	$w(y) := w(x) + w(y); w(x) := 0$
$w(x) = w(y) = 0$	Consistent with previous replies	No change



Zero loss



Lower Bound by Adversary: Details

- Note: the sum of weights is always n .
- Let x is *max*, then x is the only nonzero weighted key, that is $w(x)=n$.
- By the adversary rules:

$$w_k(x) \leq 2w_{k-1}(x)$$

- Let K be the number of comparisons x wins against previously undefeated keys:

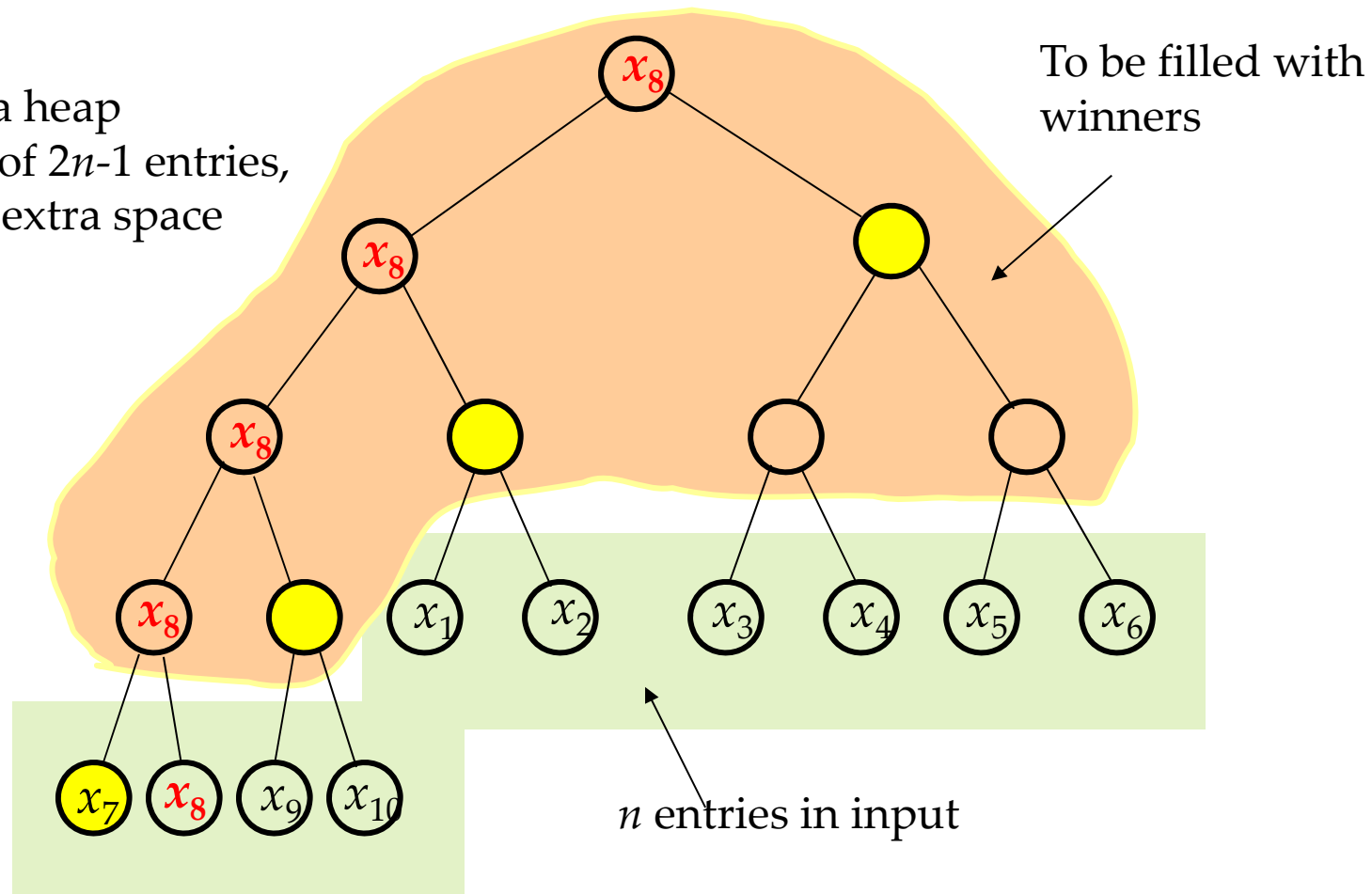
$$n = w_K(x) \leq 2^K w_0(x) = 2^K$$

- So, $K \geq \lceil \log n \rceil$



Tracking the Losers to *MAX*

Building a heap
structure of $2n-1$ entries,
using $n-1$ extra space



Finding the Median: the Strategy

- **Observation**
 - If we can partition the problem set of keys into 2 subsets: S_1 , S_2 , such that any key in S_1 is smaller than that of S_2 , the median must be located in the set with more elements.
- **Divide-and-Conquer**
 - Only one subset is needed to be processed recursively.



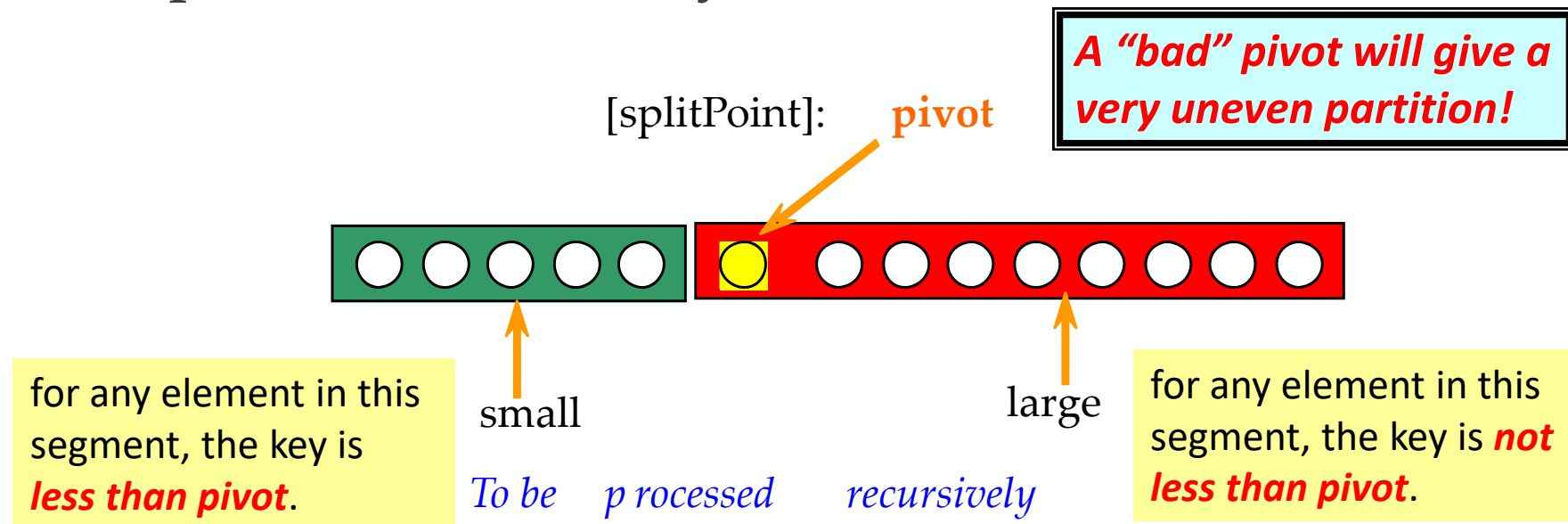
Adjusting the Rank

- The rank of the median (of the original set) in the subset considered can be evaluated easily.
- An example
 - Let $n=255$
 - The rank of median we want is 128
 - Assuming $|S_1|=96$, $|S_2|=159$
 - Then, the original median is in S_2 , and the new rank is $128-96=32$



Partitioning: Larger and Smaller

- Dividing the array to be considered into two subsets: “small” and “large”, the one with more elements will be processed recursively.



Selection: the Algorithm

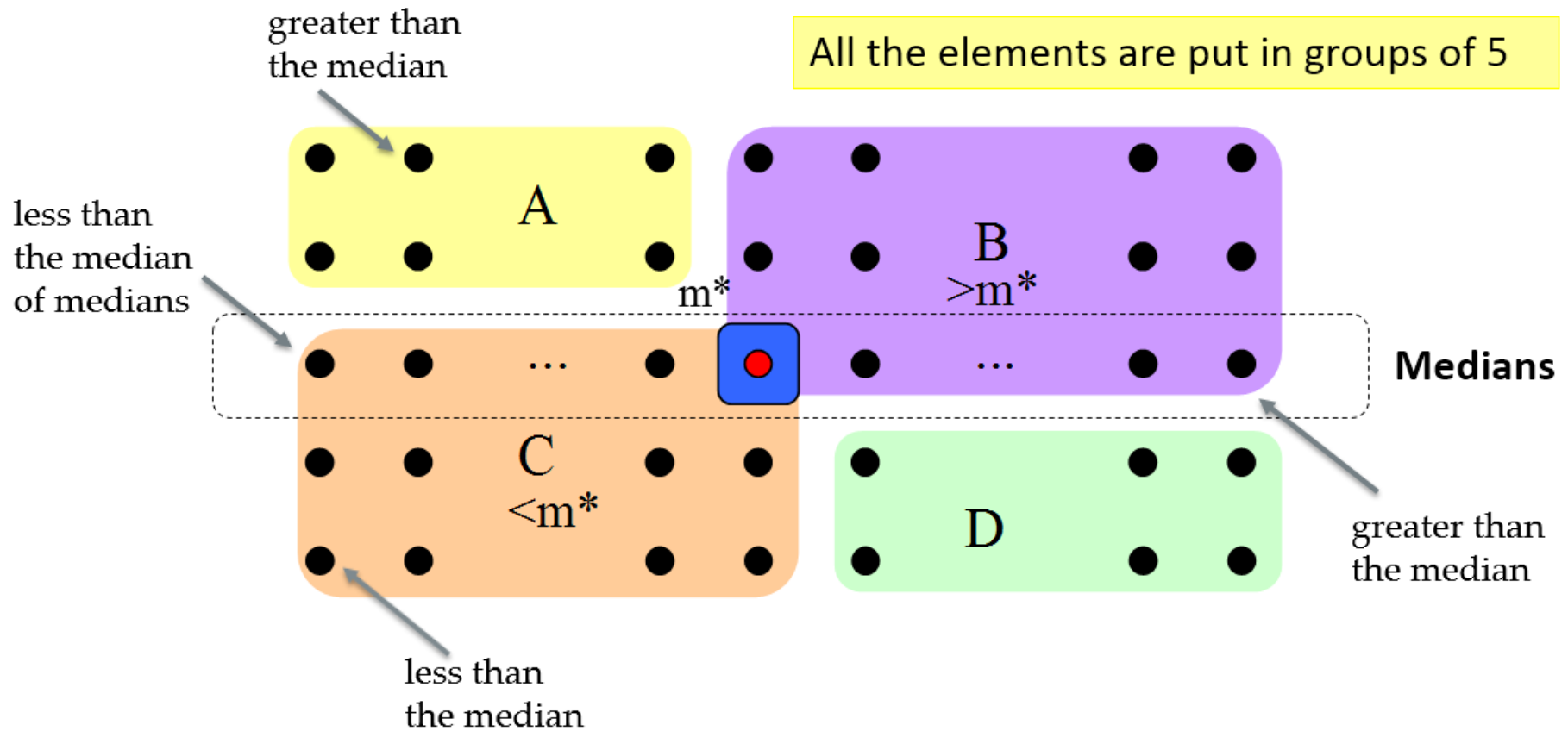
- Input: S , a set of n keys; and k , an integer such that $1 \leq k \leq n$.
- Output: The k th smallest key in S .
- Note: Median selection is only a special case of the algorithm, with $k = \lceil n/2 \rceil$.
- Procedure
- Element select(SetOfElements S , int k)
 - if ($|S| \leq 5$) return direct solution; else
 - Constructing the subsets S_1 and S_2 ;
 - Processing one of S_1, S_2 with more elements, recursively.

Key issue:

How to construct the **partition** ?



Partition improved: the Strategy



Constructing the Partition

- Find the m^* , the median of medians of all the groups of 5, as illustrated previously.
 - Compare each key in sections A and D to m^* , and
 - Let $S_1 = C \cup \{x \mid x \in A \cup D \text{ and } x < m^*\}$
 - Let $S_2 = B \cup \{x \mid x \in A \cup D \text{ and } x > m^*\}$
- (m^* is to be used as the pivot for the partition)



Divide and Conquer

```
if ( $k = |S_1| + 1$ )  
    return  $m^*$ ;  
else if ( $k \leq |S_1|$ )  
    return select( $S_1, k$ ); //recursion  
else  
    return select( $S_2, k - |S_1| - 1$ ); //recursion
```



Analysis

- For simplicity:
 - Assuming $n=5(2r+1)$ for all calls of *select*.

- $$W(n) \leq 6\left(\frac{n}{5}\right) + W\left(\frac{n}{5}\right) + 4r + W(7r+2)$$

The extreme case: all the elements in $A \cup D$ in one subset.

Finding the median in every group of 5

Finding the median of the medians

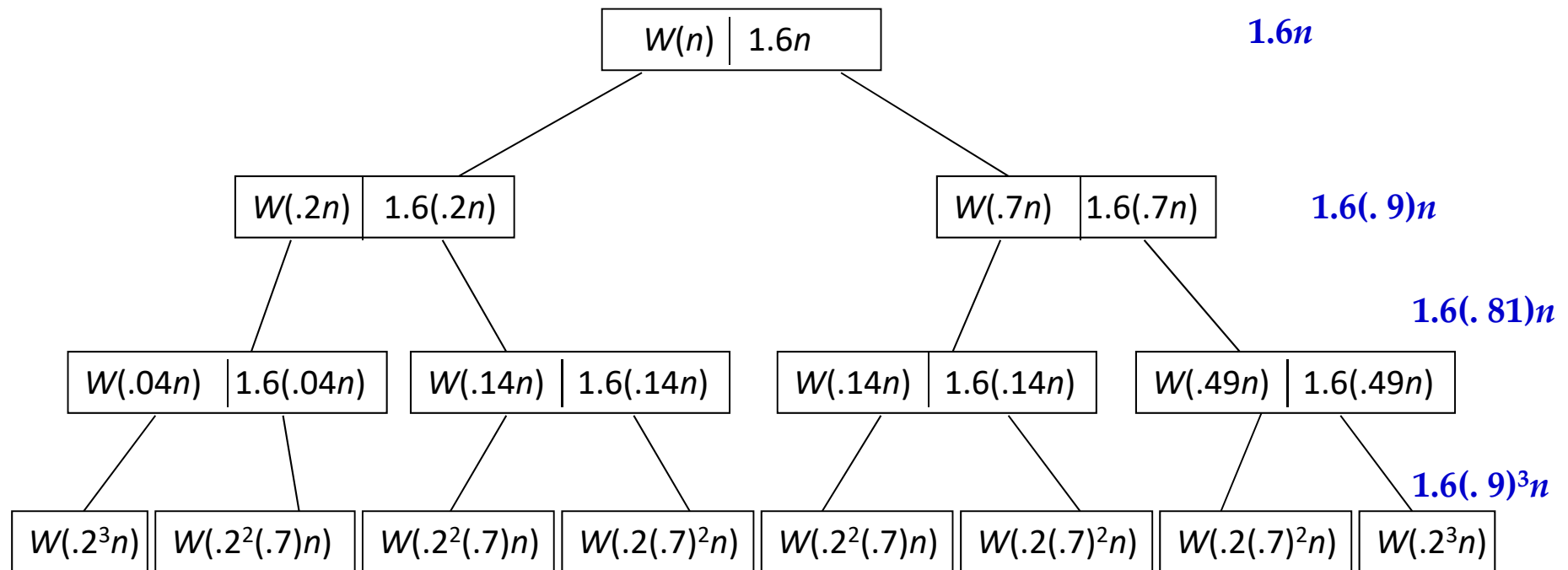
Comparing all the elements in $A \cup D$ with m^*

- *Note: r is about $n/10$, and $0.7n+2$ is about $0.7n$, so*

$$W(n) \leq 1.6n + W(0.2n) + W(0.7n)$$



Worst Case Complexity of *Select*



Note: Row sums is a decreasing geometric series, so

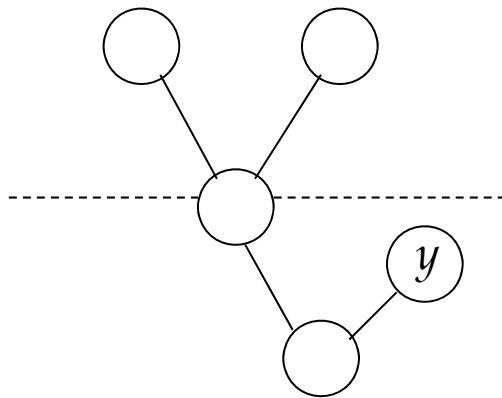
$$W(n) \in \Theta(n)$$



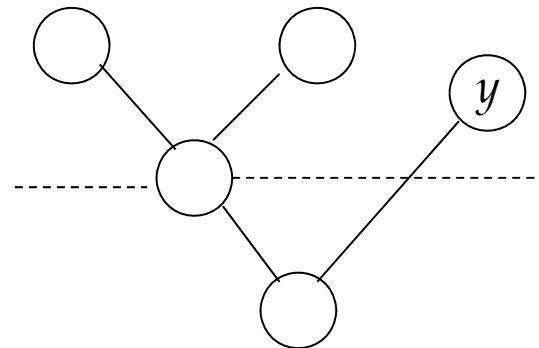
Relation to Median

- **Observation**

- Any algorithm of selection must know the relation of every element to the *median*.



Median



The adversary makes you wrong in either case

Crucial Comparison

- **A crucial comparison**
 - Establishing the relation of some x to the median.
- **Definition** (for a comparison involving a key x)
 - **Crucial comparison for x** : the first comparison where $x > y$, for some $y \geq \text{median}$, or $x < y$ for some $y \leq \text{median}$
 - **Non-crucial comparison**: the comparison between x and y where $x > \text{median}$ and $y < \text{median}$, or vice versa



Adversary for Lower Bound

- Status of the key during the running of the Algorithm:
 - L : Has been assigned a value *larger* than median
 - S : Has been assigned a value *smaller* than median
 - N : Has not yet been in a comparison

- Adversary rule:

Comparands	Adversary's action
N, N	one L , the another S
L, N or N, L	change N to S
S, N or N, S	change N to L
(In all other cases, just keep consistency)	



Notes on the Adversary Arguments

- All actions explicitly specified above make the comparisons **un-crucial**.
 - At least, $(n-1)/2$ L or S can be assigned freely.
 - If there are already $(n-1)/2$ S , a value **larger** than median must be assigned to the new key, and if there are already $(n-1)/2$ L , a value **smaller** than median must be assigned to the new key. The last assigned value is the median.
- So, an adversary can force the algorithm to do $(n-1)/2$ un-crucial comparisons at least (In the case that the algorithm start out by doing $(n-1)/2$ comparisons involving two N).



Lower Bound for Selection Problem

- **Theorem:**
 - Any algorithm to find the median of n keys (for odd n) by comparison of keys must do at least $3n/2 - 3/2$ comparisons in the worst case.
- **Argument:**
 - There must be done $n-1$ crucial comparisons at least.
 - An adversary can force the algorithm to perform as many as $(n-1)/2$ uncrucial comparisons.
 - Note: the algorithm can always start out by doing $(n-1)/2$ comparisons involving 2 N -keys, so, only $(n-1)/2$ L or S left for the adversary to assign freely as the adversary rule.



Thank you!

Q & A

Yu Huang

<http://cs.nju.edu.cn/yuhuang>

