CS4311
Design and Analysis of Algorithms

Lecture 8: Order Statistics

About this lecture

- Finding max, min in an unsorted array (upper bound and lower bound)
- Finding both max and min (upper bound)
- Selecting the k^{th} smallest element

k^{th} smallest element $\equiv$ k^{th} order statistics

Finding Maximum

in unsorted array

Finding Maximum (Method I)

- Let S denote the input set of n items
- To find the maximum of S, we can:

Step 1: Set max = item 1

Step 2: for k = 2, 3, …, n

if (item k is larger than max)

Update max = item k;

Step 3: return max;

comparisons = n – 1

Finding Maximum (Method II)

- Define a function Find-Max as follows:

Find-Max(R, k) `/* R is a set with k items */`

1. if ($k \leq 2$) return maximum of R;
2. Partition items of R into $\lfloor k/2 \rfloor$ pairs;
3. Delete smaller item from R in each pair;
4. return Find-Max(R, $k - \lfloor k/2 \rfloor$);

Calling Find-Max(S,n) gives the maximum of S

Finding Maximum (Method II)

Let $T(n)$ = # comparisons for Find-Max with problem size n

So, $T(n) = T(n - \lfloor n/2 \rfloor) + \lfloor n/2 \rfloor$ for $n \geq 3$

$T(2) = 1$

Solving the recurrence (by substitution),

we get $T(n) = n - 1$

Lower Bound

Question: Can we find the maximum using fewer than $n - 1$ comparisons?

Answer: No ! To ensure that an item x is not the maximum, there must be at least one comparison in which x is the smaller of the compared items

So, we need to ensure $n-1$ items not max

➔ at least $n - 1$ comparisons are needed

Finding Both Max and Min

in unsorted array

Finding Both Max and Min

Can we find both max and min quickly?

Solution 1:

First, find max with $n - 1$ comparisons

Then, find min with $n - 1$ comparisons

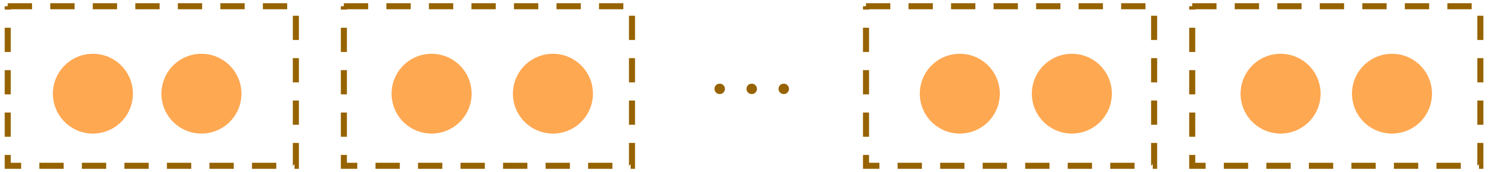
➔ Total = $2n - 2$ comparisons

Is there a better solution ??

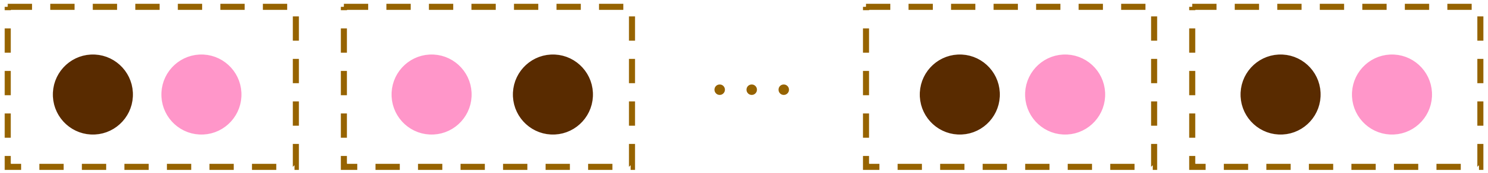
Finding Both Max and Min

Better Solution: (Case 1: if n is even)

First, partition items into $n/2$ pairs;



Next, compare items within each pair;

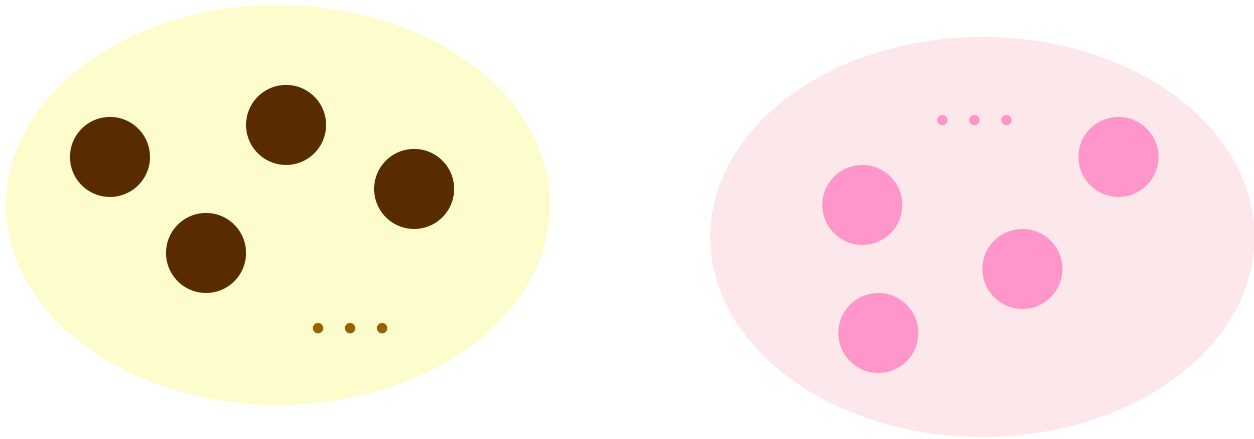


= larger = smaller

Finding Both Max and Min

Then, max = Find-Max in larger items

min = Find-Min in smaller items



Find-Max Find-Min

comparisons = 3n/2 - 2

Finding Both Max and Min

Better Solution: (Case 2: if n is odd)

We find max and min of first n - 1 items;

if (last item is larger than max)

    Update max = last item;

if (last item is smaller than min)

    Update min = last item;

comparisons = $3(n-1)/2$

Finding Both Max and Min

Conclusion:

To find both max and min:

if n is odd: $3(n-1)/2$ comparisons

if n is even: $3n/2 - 2$ comparisons

Combining: at most $3\lfloor n/2 \rfloor$ comparisons

→ better than finding max and min separately

Lower Bound

Textbook Ex 9.1-2 (Very challenging):

- Show that we need at least

$$\lceil 3n/2 \rceil - 2 \text{ comparisons}$$

to find both max and min in worst-case

Hint: Consider how many numbers may be max or min (or both). Investigate how a comparison affects these counts

Selecting k^{th} smallest item

in unsorted array

Selection in Linear Time

- In next slides, we describe a recursive call

  Select(S,k)

  which supports finding the k^{th} smallest element in S
- Recursion is used for two purposes:

  (1) selecting a good pivot (as in Quicksort)

  (2) solving a smaller sub-problem

Select(S, k)

```
/* First,find a good pivot */
```

1. Partition S into $\lceil |S|/5 \rceil$ groups, each group has five items (one group may have fewer items);
2. Sort each group separately;
3. Collect median of each group into S';
4. Find median m of S':

$$m = \text{Select}(S', \lceil \lceil |S|/5 \rceil /2 \rceil);$$

```
4.  Let q = # items of S smaller than m;
5.  If (k == q + 1)
        return m;
/* Partition with pivot */
6.  Else partition S into X and Y
        X = {items smaller than m}
        Y = {items larger than m}
/* Next,form a sub-problem */
7.  If (k < q + 1)
        return Select(X, k)
8.  Else
        return Select(Y, k-(q+1));
```

Selection in Linear Time

Questions:

1. Why is the previous algorithm correct? (Prove by Induction)

2. What is its running time?

Running Time

- In our selection algorithm, we chose m, which is the median of medians, to be a pivot and partition S into two sets X and Y
- In fact, if we choose any other item as the pivot, the algorithm is still correct
- Why don't we just pick an arbitrary pivot so that we can save some time ??

Running Time

- A closer look reviews that the worst-case running time depends on $|X|$ and $|Y|$
- Precisely, if $T(|S|)$ denote the worst-case running time of the algorithm on S, then

$$T(|S|) = T(\lceil |S|/5 \rceil) + \Theta(|S|)$$
$$+ \max\{T(|X|), T(|Y|)\}$$

Running Time

- Later, we show that if we choose m, the "median of medians", as the pivot,

  both |X| and |Y| will be at most 3|S|/4

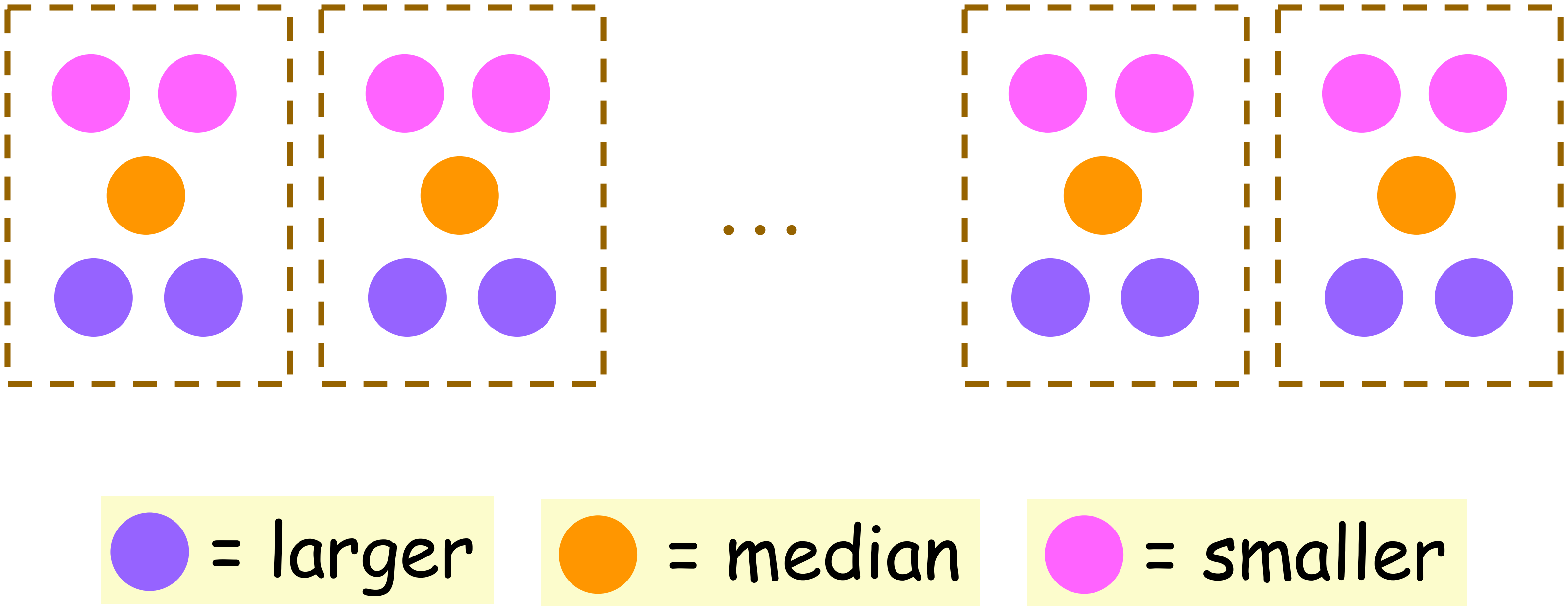
- Consequently,

$$T(n) = T(\lceil n/5 \rceil) + \Theta(n) + T(3n/4)$$

$\Rightarrow$ $T(n) = \Theta(n)$ (obtained by substitution)

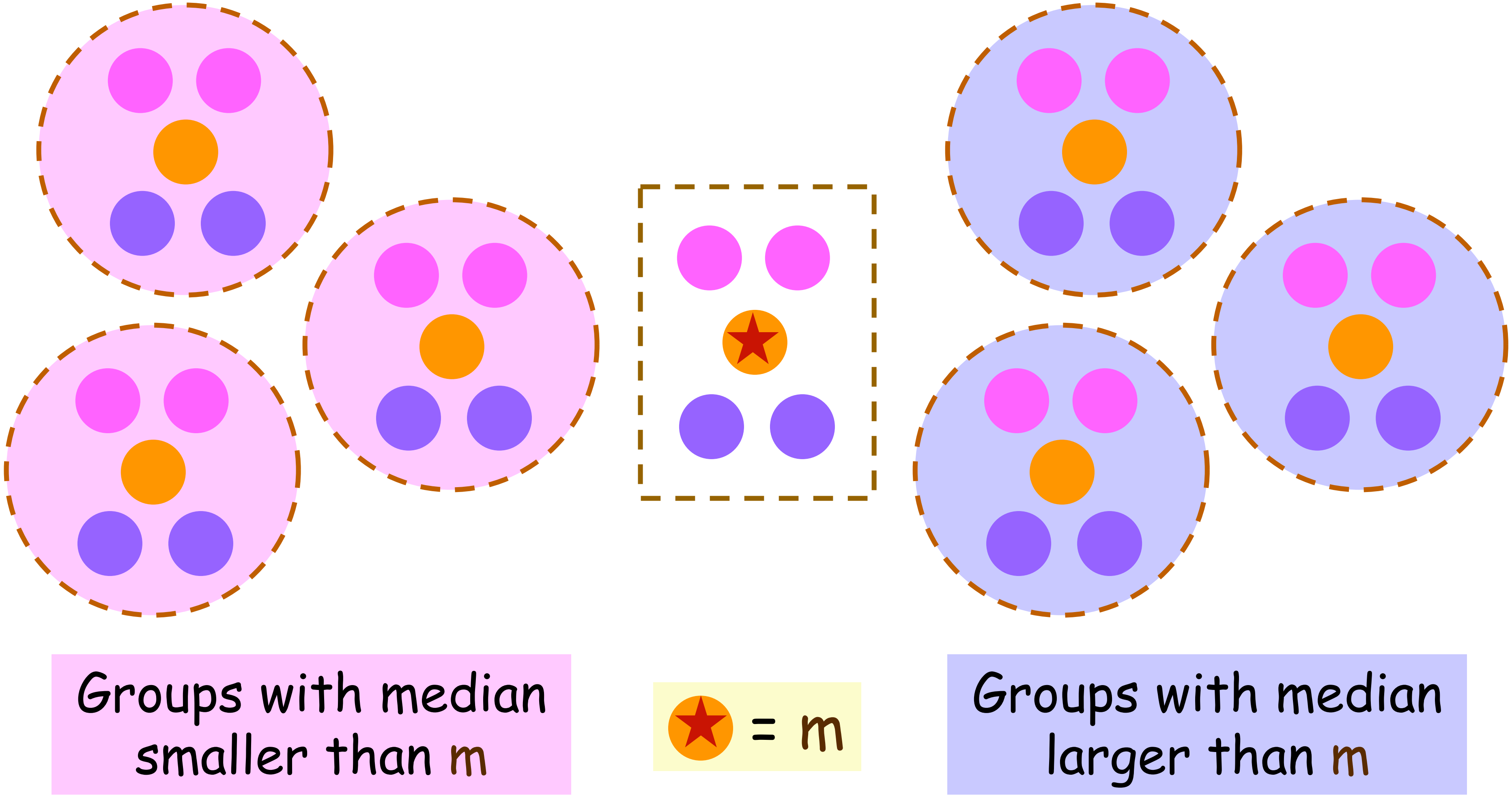
Median of Medians

- Let's begin with $\lceil n/5 \rceil$ sorted groups, each has 5 items (one group may have fewer)



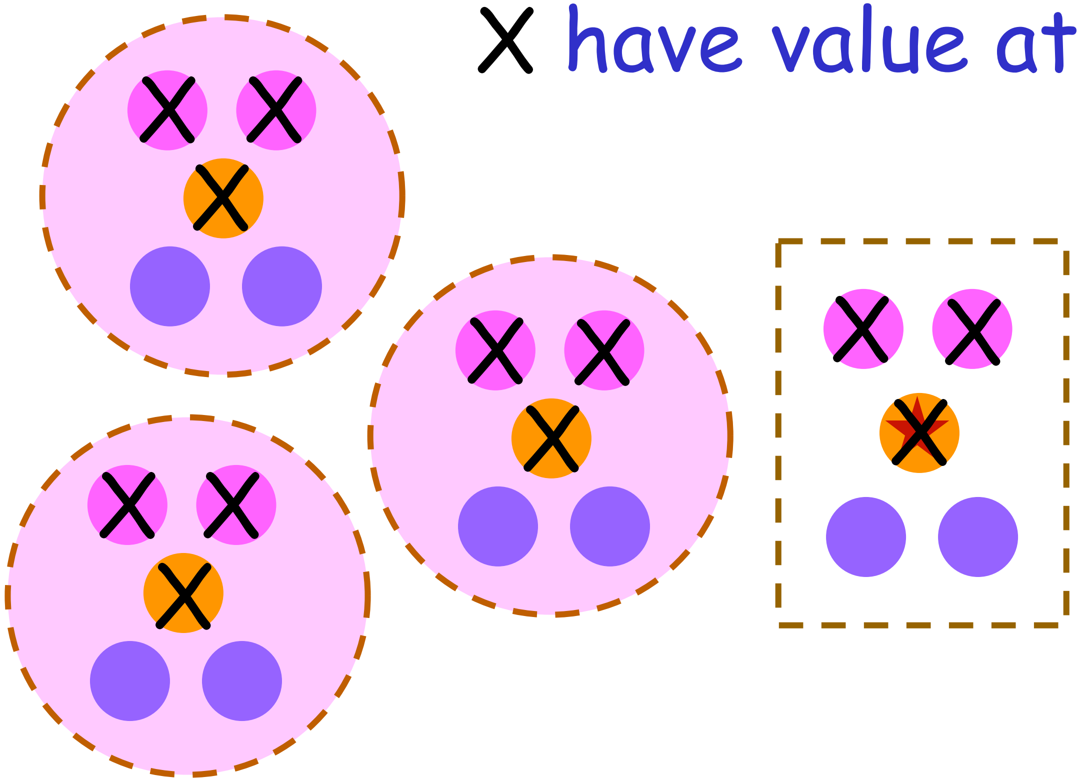
Median of Medians

- Then, we obtain the median of medians, m



Median of Medians

Then, we know that all items marked with X have value at most m



Groups with median smaller than m

X = "value $\leq$ m"

Median of Medians

The number of items with value at most m is at least

$$\underbrace{3}_{\text{each full group has 3 `crossed' items}}\Big(\underbrace{\left\lceil \lceil n/5 \rceil / 2 \right\rceil - 1}_{\text{min \# of groups}}\Big) \underbrace{- 2}_{\text{one group may have only 1 `crossed' item}}$$

➔ number of items: at least $3n/10 - 5$

Median of Medians

Previous page implies that at most

$7n/10 + 5$ items

are greater than m

➔ For large enough n (say, $n \geq 100$)

$$7n/10 + 5 \leq 3n/4$$

➔ $|Y|$ is at most $3n/4$ for large enough n

Median of Medians

Similarly, we can show that at most

$7n/10 + 5$ items are smaller than m

➔ |X| is at most $3n/4$ for large enough n

Conclusion:

The "median of medians" helps us control the worst-case size of the sub-problem

➔ without it, the algorithm runs in $\Theta(n^2)$ time in the worst-case