Chapter 9: Medians and 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 ! Every element except the winner must drop at least one match
- 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

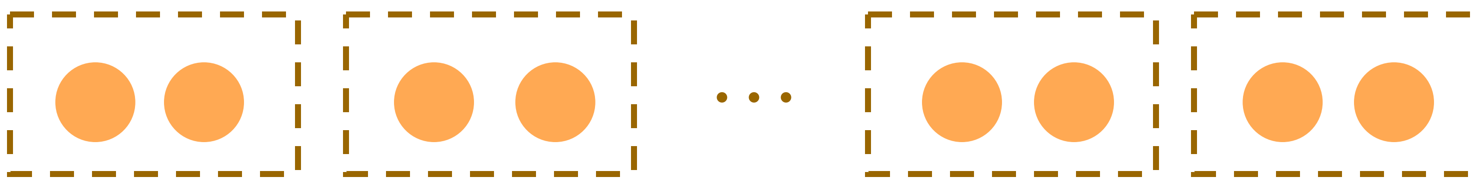
➔ Total = 2n - 2 comparisons

Is there a better solution ??

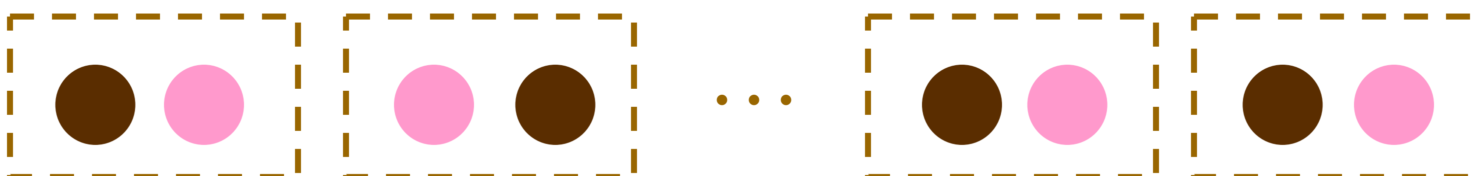
Finding Both Max and Min

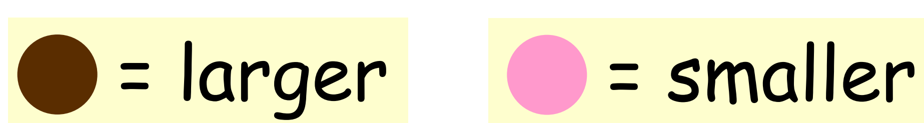
- Better Solution: (Case 1: if n is even)

First, partition items into n/2 pairs;



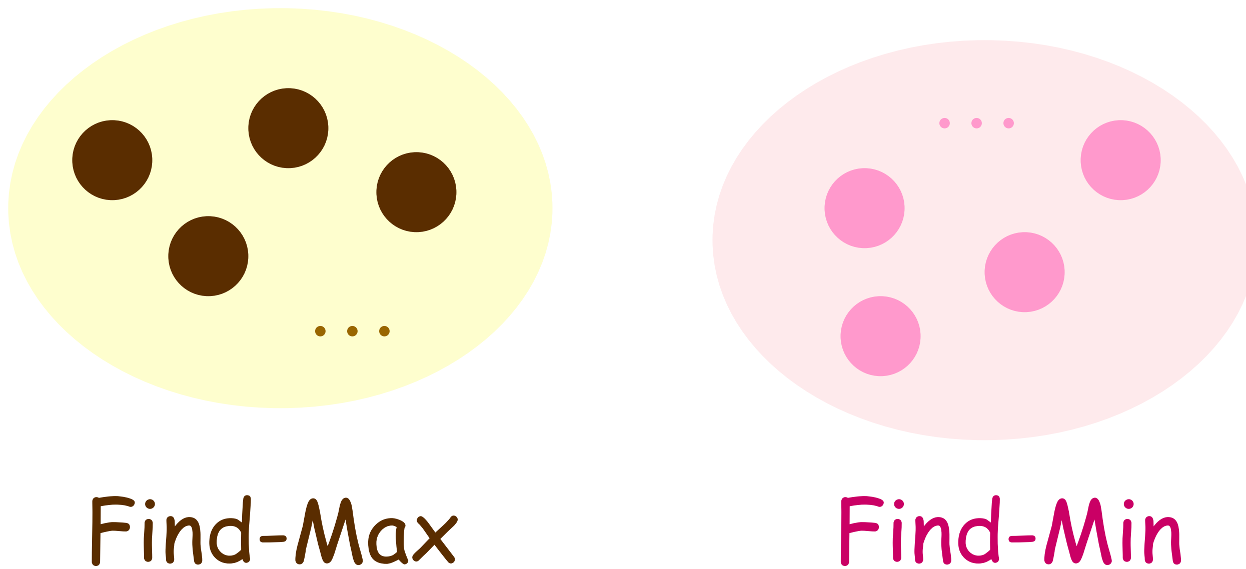
Next, compare items within each pair;





Finding Both Max and Min

- Then, max = Find-Max in larger items
  min = Find-Min in smaller items



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 $\lfloor 3n/2 \rfloor$ comparisons

    ➔ better than finding max and min separately

Selecting k^{th} smallest item

in unsorted array

Selection in Expected Linear Time

Randomized-Select(A, p, r, i)

1. if p==r return A[p]
2. q = Randomized-Partition(A, p, r)
3. k = q – p + 1
4. if i ==k //the pivot value is the answer
   return A[q]
5. else if i < k
   return Randomized-Select(A, p, q-1, i)
6. else return Randomized-Select(A, q+1, r, i-k)

Example

- p = 1, r = 8, i = 6

1	3	7	8	2	6	4	5

Random pivot

- After Randomized-Partition

1	3	2	4	5	6	8	7

- q = 5
  k = q-p+1 = 5

Example

- i > k
- Randomized-Partition(*A*, 6, 8, 1)

6	8	7

Random pivot

- After Randomized-Partition

6	8	7

- q = 6, k = q-p+1 = 1, i = 1
- 6 is the answer

Running Time (1)

- Worst case: $T(n) = O(n) + T(n-1) = O(n^2)$
- Average case:
- $E[T(n)] = O(n) + 1/n \sum_{1 \le k \le n} E[T(\max(k-1, n-k))]$
  $= O(n) + 2/n \sum_{\lfloor n/2 \rfloor \le k \le n-1} E[T(k)]$ (why?)
- We can prove $E[T(n)] \le cn$ using mathematematic induction method.
- Basis: $T(n) = O(1)$ for n less some constant

Running Time (2)

- Induction Step:
- Assume E[T(n)] ≤ cn hold for n ≤ k′
- We need to prove n = k′ + 1 hold

$$\mathrm{E}[T(n)] \leq \frac{2}{n} \sum_{k=\left\lfloor \frac{n}{2} \right\rfloor}^{n-1} ck + an$$

$$= \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\left\lfloor \frac{n}{2} \right\rfloor - 1} k \right) + an$$

$$= \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(\left\lfloor \frac{n}{2} \right\rfloor - 1)\lfloor n/2 \rfloor}{2} \right) + an$$

Running Time (3)

$$\le \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{\left(\frac{n}{2}-2\right)\left(\frac{n}{2}-1\right)}{2}\right) + an$$

$$= \frac{2c}{n}\left(\frac{n^2-n}{2} - \frac{\frac{n^2}{4}-\frac{3n}{2}+2}{2}\right) + an$$

$$= \frac{c}{n}\left(\frac{3n^2}{4} + \frac{n}{2} - 2\right) + an$$

$$= \mathrm{c}\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an$$

Running Time (4)

$$\leq \frac{3cn}{4} + \frac{c}{2} + an$$

$$= cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$$

- In order to complete the proof, we need

$$\frac{cn}{4} - \frac{c}{2} - an \geq 0 \Rightarrow \frac{cn}{4} - an \geq \frac{c}{2}$$

$$\Rightarrow n\left(\frac{c}{4} - a\right) \geq \frac{c}{2} \Rightarrow n \geq \frac{\frac{c}{2}}{\frac{c}{4} - a} = \frac{2c}{c - 4a}$$

- Thus, if we assume T(n) = O(1) for $n < \frac{2c}{c-4a}$, then E[T(n)] = O(n)

Selection in Linear Time

- In next slides, we describe a recursive call

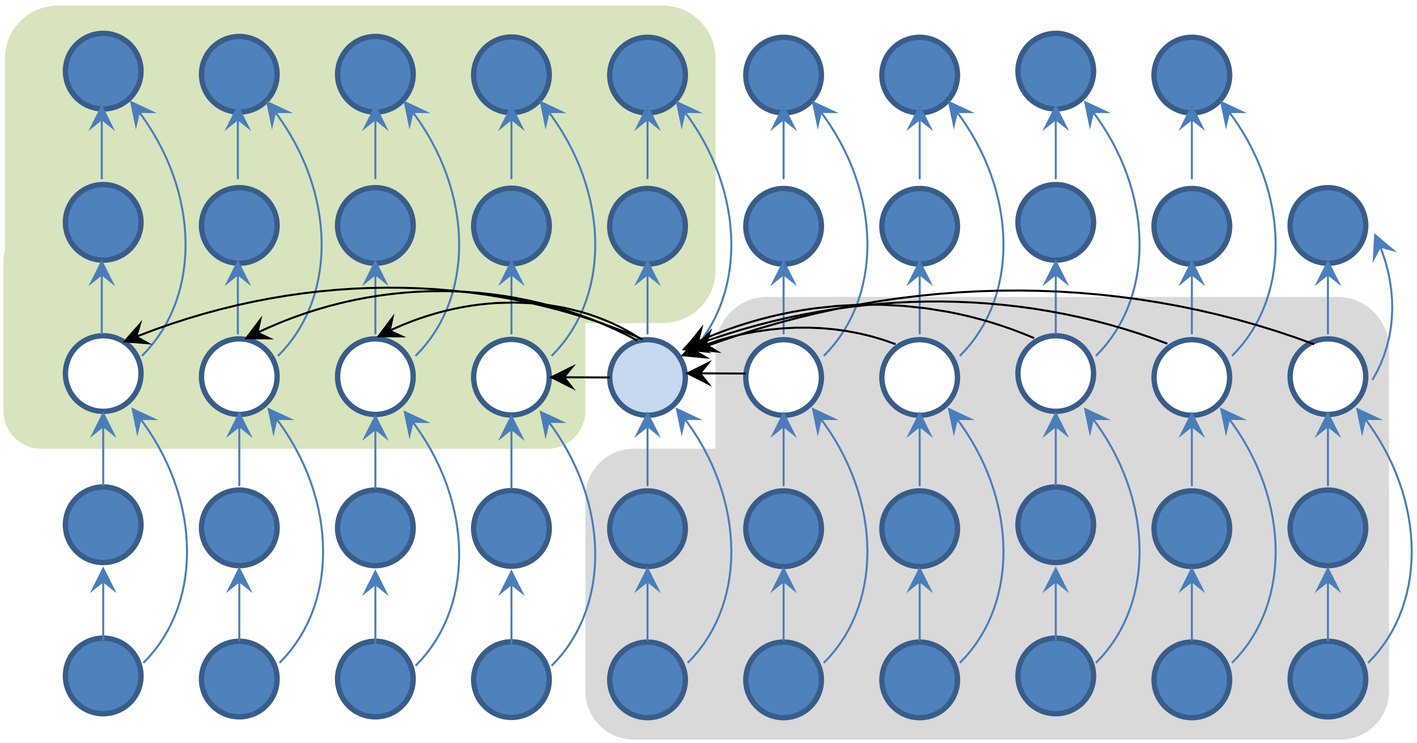
  Select(S, k)

  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 median of median as the Pivot



Selection in worst-case linear time (1)

Select(A, p, r, i) /* First, find a good pivot */

//Partition A into g groups, each group has five items (one group may have fewer items) and sort each group separately;

1. $g = (r - p + 1)/5$
2. For j = p to $p + g - 1$;
3. Sort($A[j]$, $A[j+g]$, $A[j+2g]$, $A[j+3g]$, $a[j+4g]$) in place

// Find the pivot m recursively as the median of the group medians

$$m = \text{Select}(A, p+2g, p+3g-1, \lceil g/2 \rceil);$$

Selection in worst-case linear time (2)

```
// Partition with pivot m

4. q = Partition(A, p, r, m)

5.  k= q - p + 1
            if i==k return A[q];

6.  else if (i < k)
            return Select(A, p, q-1, i)

7.  else  return Select(A, q+1, r, i-k);
```

Find #23 in 49 numbers

1. Select(S, 1, 49, 23)

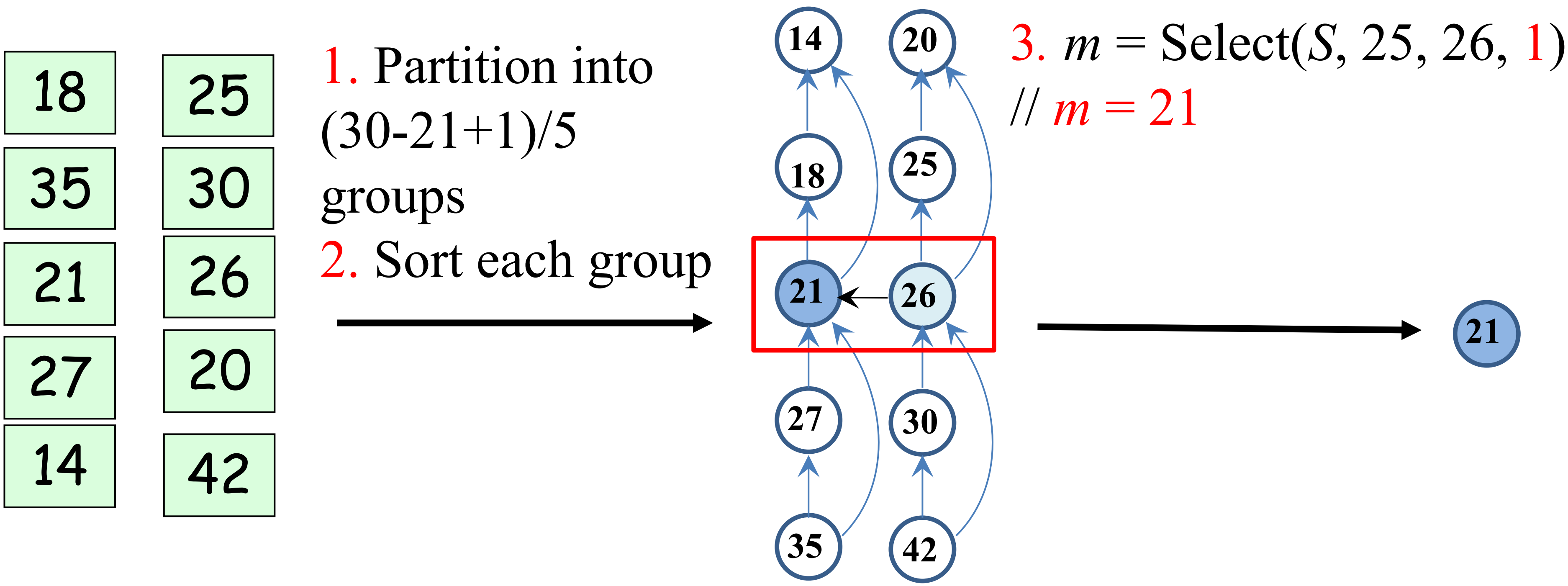
31	44	24	39	14	41	29	6	20	49
32	38	4	16	5	33	30	43	36	19
12	1	21	34	40	2	47	46	3	28
15	35	10	13	11	25	8	26	45	42
18	23	22	27	48	9	37	17	7	

2. Insertion sort for every group

12	1	4	13	5	2	8	6	3	19
15	23	10	16	11	9	29	17	7	28
18	35	21	27	14	25	30	26	20	42
31	38	22	34	40	33	37	43	36	49
32	44	24	39	48	41	47	46	45	

Find #23 in 49 numbers

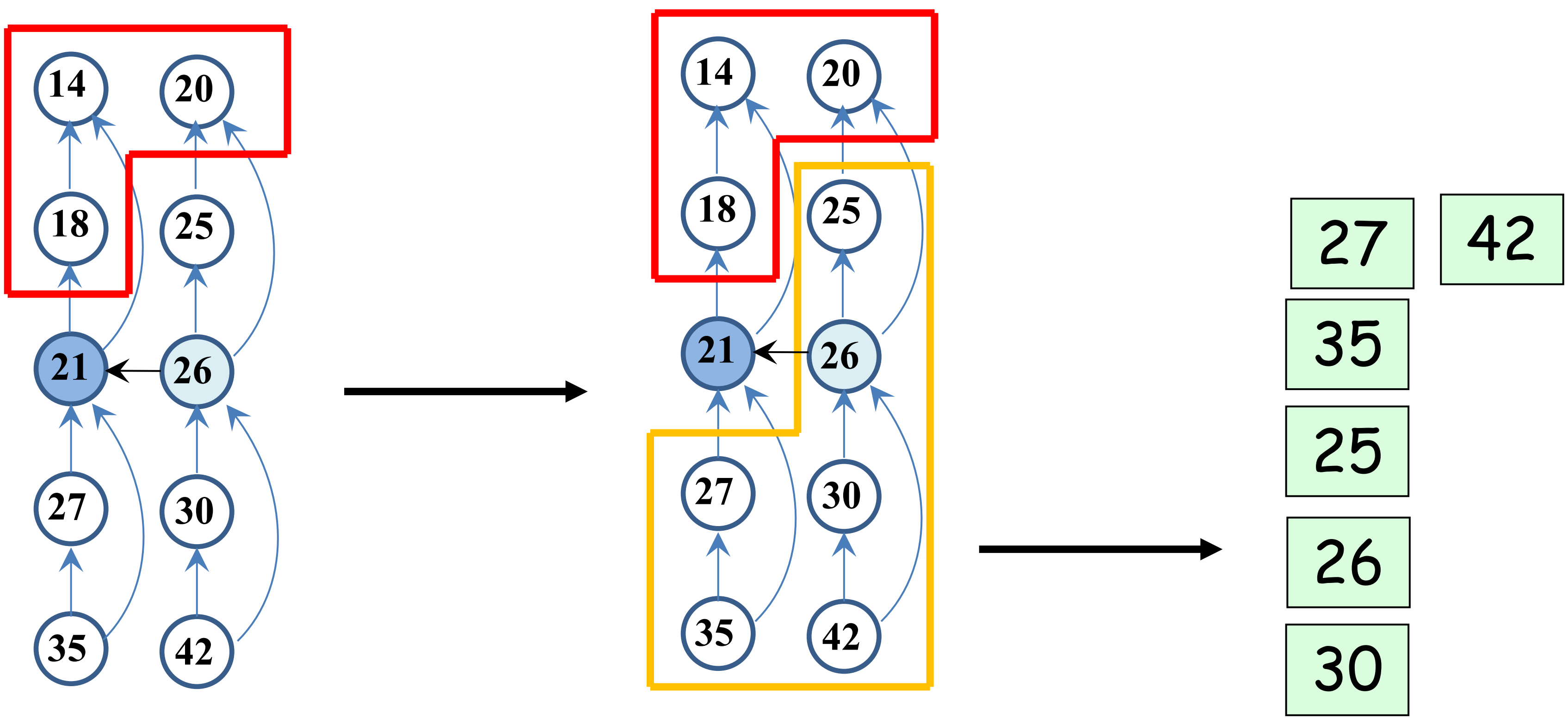
3. Find median *m* of *S*: *m* = Select(S, 21, 30, 5);



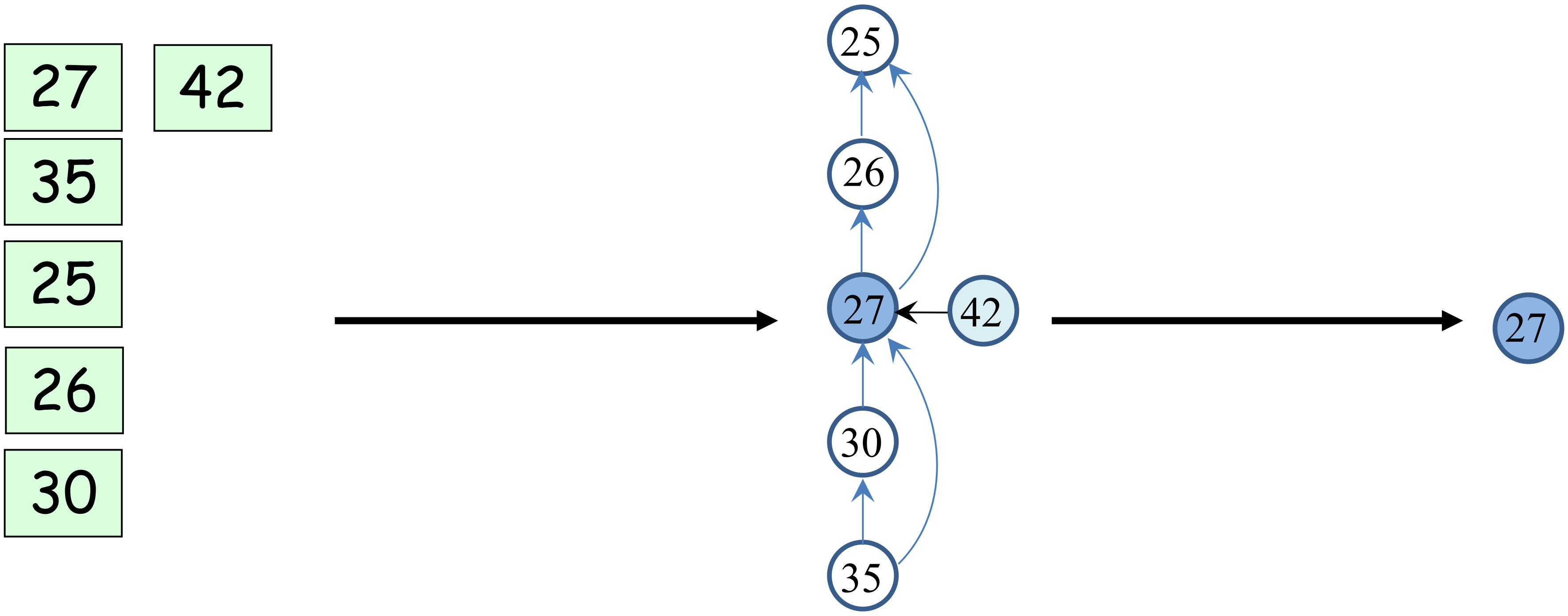
Find #23 in 49

4. q = Partition(S, p, r, m)
// p = 21, r = 30
// m ($pivot$) = 21
// return q = 24 (index)

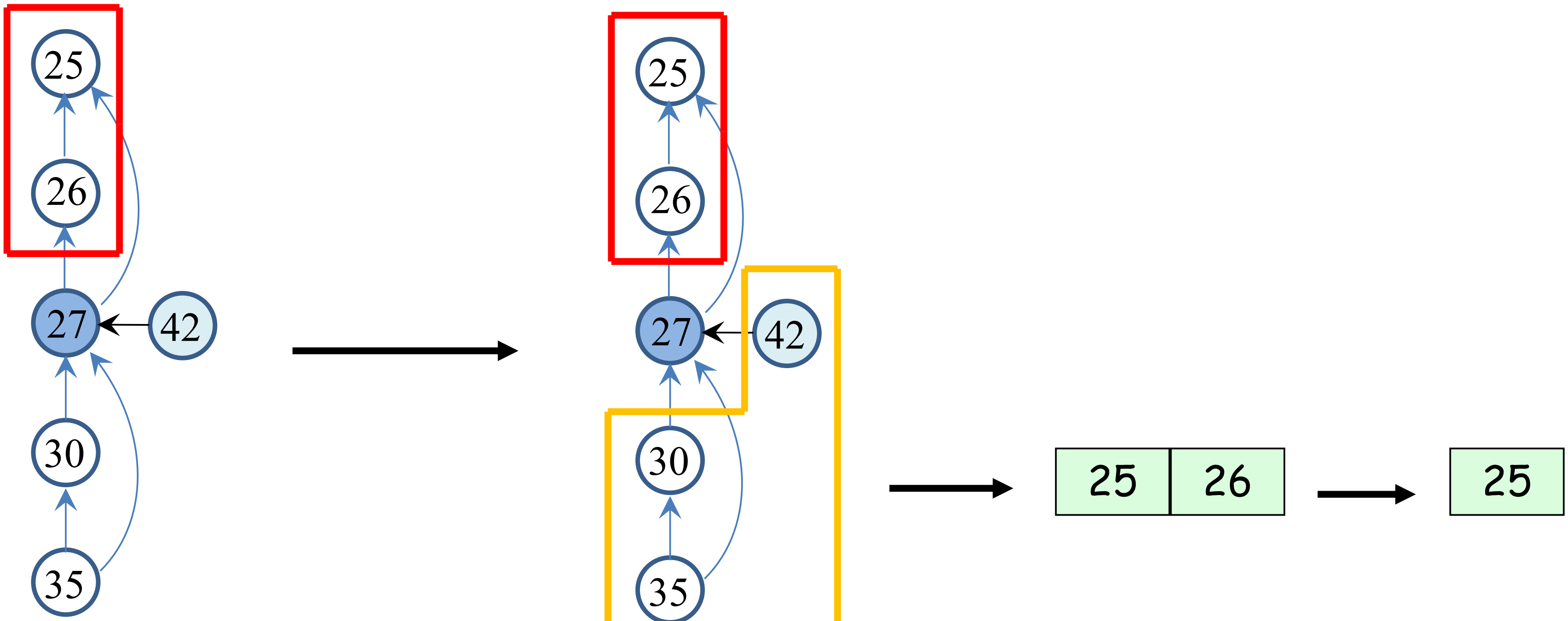
5. $k = q\text{-}p+1$ // k = 24 -21 + 1 = 4
6. if $i = k$ return $S(q)$ // i = 5
7. elseif ($i < k$)
   return Select(S, p, q-1, i)
8. else return Select(S, q+1, r, i-k)



Find #23 in 49

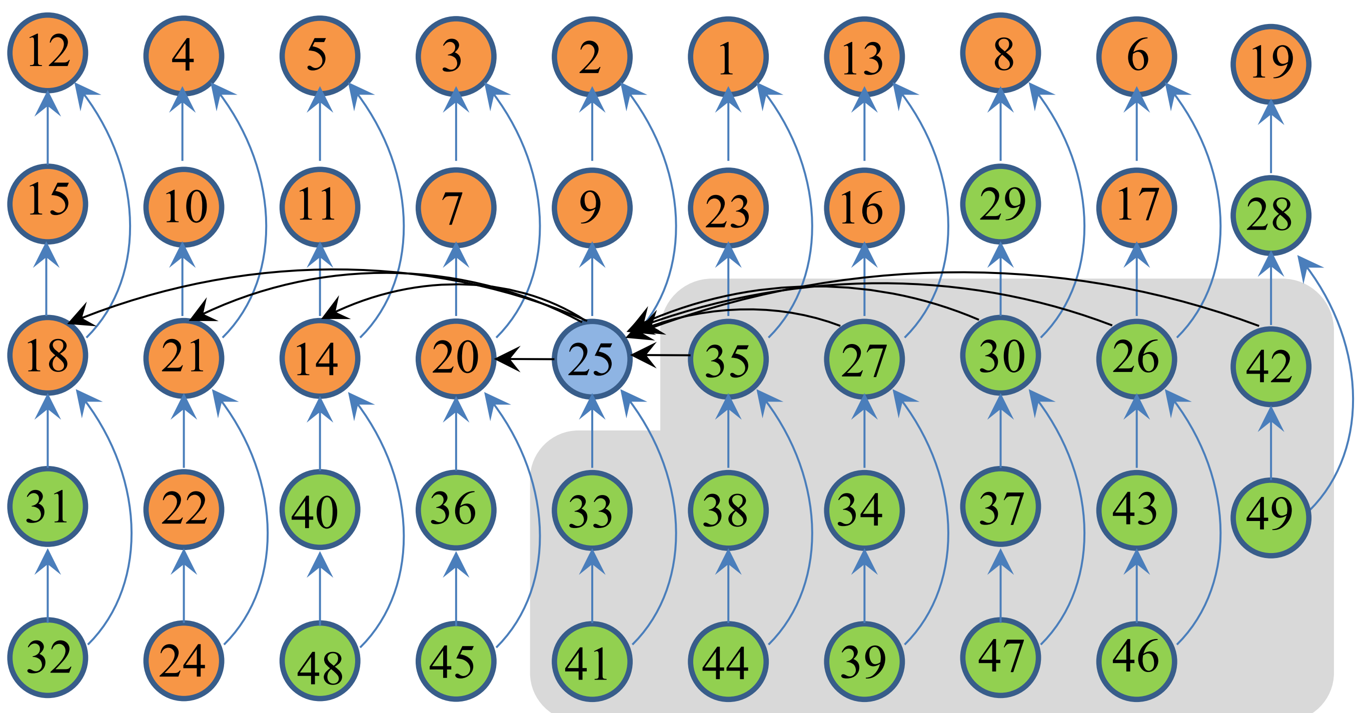


Find #23 in 49



Find #23 in 49

5. k = items in S smaller than 25 = 24
6. $k > 23$
7. Select(S, p, q-1, i) // p = 1, q= 25, i =23
8. Repeat Select until get #23



Running Time

- In our selection algorithm, we choose m, which is the median of medians, to be a pivot and partition S into two sets X = {items smaller than mediam} and Y = {items larger than mediam}
- 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 7|S|/10 + 6

- Consequently,

$$T(n) = T(\lceil n/5 \rceil) + \Theta(n) + T(7n/10 + 6)$$

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

Substitution

- Assume T(n) $\leq cn$ hold for $n \leq k'$. We need to prove $n = k' + 1$ hold.

$$T(n) \leq c\lceil n/5 \rceil + c(7n/10+6) + an$$

$$\leq cn/5 + c + 7cn/10 + 6c + an$$

$$= 9cn/10 + 7c + an$$

$$= cn + (-cn/10 + 7c + an),$$

$$\leq cn \qquad \text{if } -cn/10 + 7c + an \leq 0$$

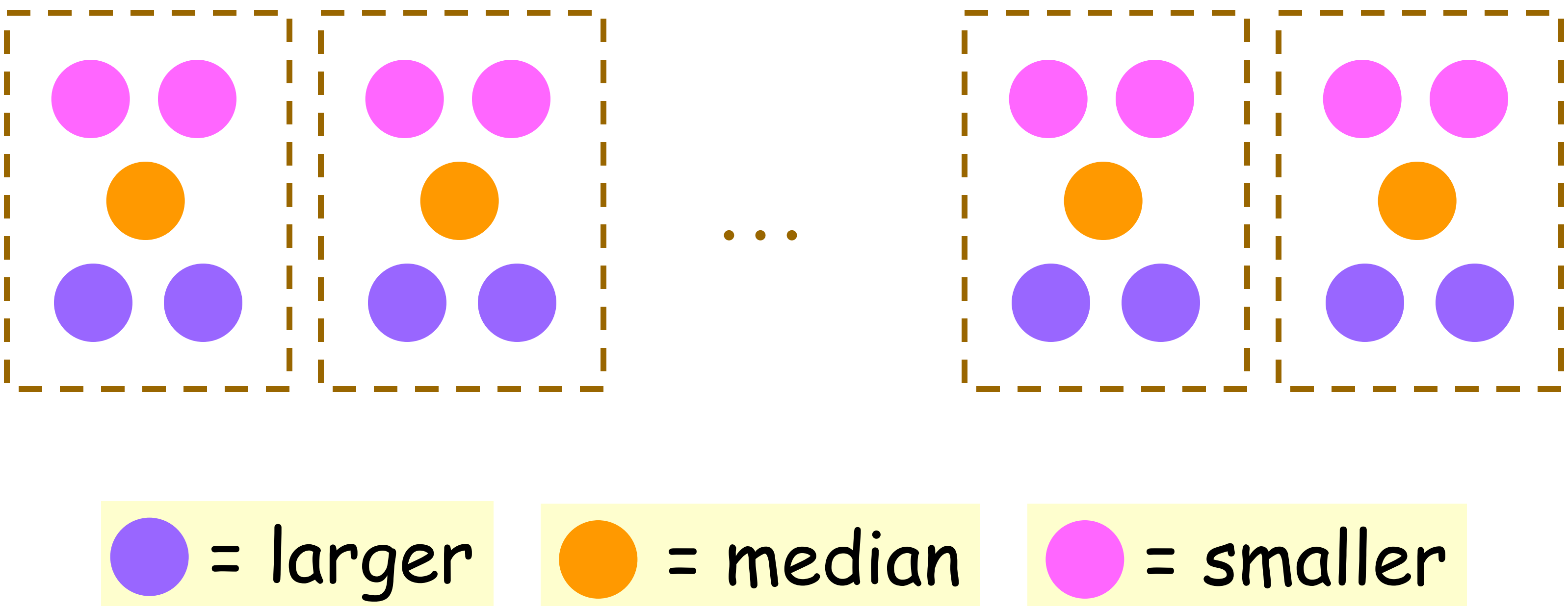
=> $c \geq 10an/(n-70)$ when $n > 70$

If we assume $n \geq 140$, we have $n/(n-70) \leq 2$.

So, we choose $c \geq 20a$

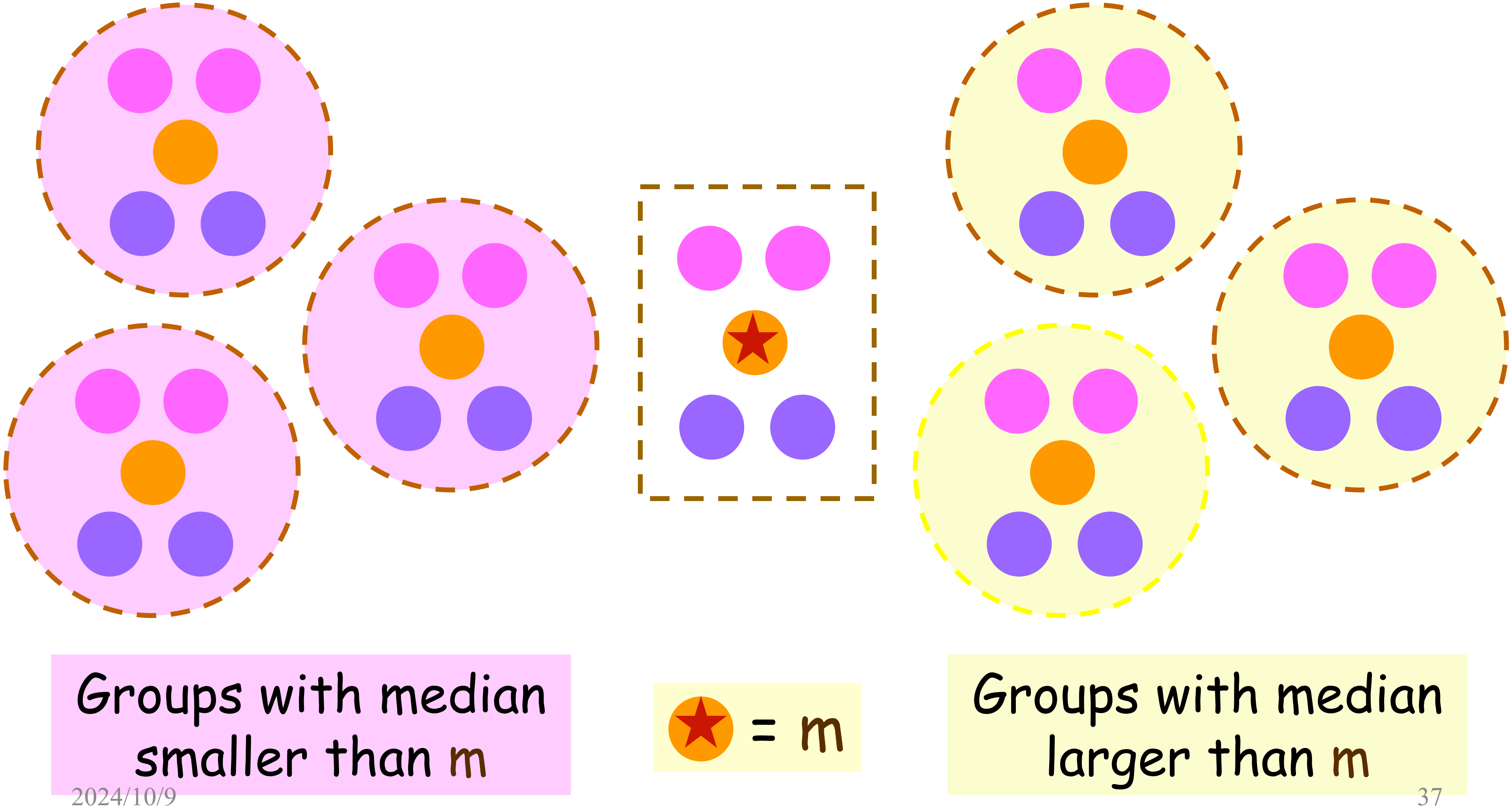
Median of Medians

- 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

The number of items with value greater than m is at least

$$\underbrace{3}_{\text{each full group has 3 'crossed' items}}(\underbrace{\lceil \lceil n/5 \rceil / 2 \rceil}_{\text{min \# of groups}} \underbrace{- 2}_{\text{two groups may not have 3 'crossed' items}})$$

➔ number of items: at least 3n/10 – 6

Median of Medians

Previous page implies that at most

$$7n/10 + 6 \text{ items}$$

are smaller than m

➔ For large enough n (say, $n \geq 140$)

$$7n/10 + 6 \leq 3n/4$$

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

Median of Medians

Similarly, we can show that at most

$7n/10 + 6$ items are larger than m

➔ |Y| 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

Practice at Home

- Exercises: 9.1-1, 9.1-3, 9.2-3, 9.3-1, 9.3-3, 9.3-5, 9.3-7, 9.3-9, 9.3-10
- Problem 9-1
- (Bonus) Programming Report: a. Use the SELECT algorithm to find the *k*th smallest element of an input array of $n > 10{,}000{,}000$. (Please compare the running time of your algorithm with the input elements divided into groups 3, 5, 7, 9, and Randomized-Select. Average the execution time of 50 ~100 experiments for each group size and Randomized-Select). b. Compare the performance obtained in a) with an iterative version of the SELECT algorithm.