Algorithmic Problem Solving

A case study

By the end of this video you will be able to…

- An example of an algorithmic problem.
- The application of our problem solving strategy to this example.

How would you find the k^{th} smallest element in an array of integers?

Step 1

How would you find the k^{th} smallest element in an array of integers?

Step 2

example

Step 3

How would you find the k^{th} smallest element in an array of integers?

Assuming
- Array not presorted
- Duplicate elements okay
- Can change the array

Step 4

How would you find the k^{th} smallest element in an array of integers?

Find smallest … discard
Then next smallest … discard

Repeat k times!

Step 4

How would you find the k^{th} smallest element in an array of integers?

Find smallest … discard
Then next smallest … discard

Repeat k times!

O(n)
O(n)

Step 4

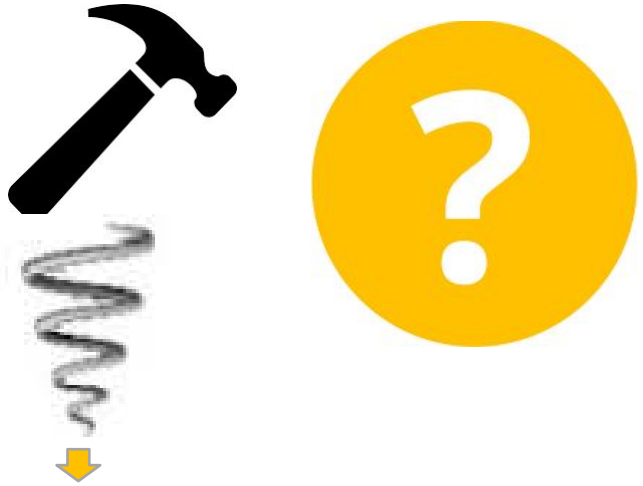
How would you find the k^{th} smallest element in an array of integers?

Find smallest … discard
Then next smallest … discard

Repeat k times!

If k is n/2, get $O(n^2)$!

Step 5



How would you find the k^{th} smallest element in an array of integers?

1. Sort!
2. Return element in index k.

O(n log n)

O(1)

Step 6

```
public void processData()
{
  do
  {
    int data = getData();
    if(data < 0)
      performOperation1(data);
    else
      performOperation2(data);
  }
  while(hasMoreData());
}
```

Clip 1 for Tuesday 1/19 session

- At this point, we're ready to move to the whiteboard and code up a solution.
- Our strategy is: sort, then read off kth element.
- Code on whiteboard
  - Header: use meaningful name, careful with return value + args
  - Validate inputs: throw exception otherwise

- Whiteboarding tips

Step 6

```
public int kthSmallestViaSort(int[] array, int k) {
    if (k <= 0 || k > array.length) {
        throw new IllegalArgumentException();
    }

    Arrays.sort(array);

    return array[k-1];
}
```

Whiteboarding tips

- Write big!
- Keep space between lines of code ; easier to edit.
- Use meaningful, legible, short names: not i,j
- Practice

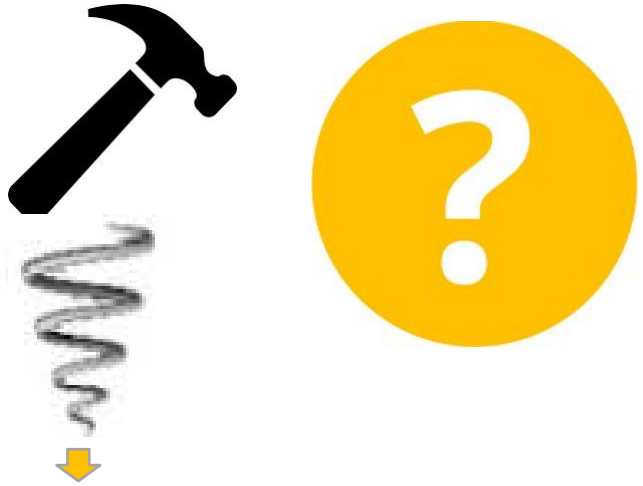
Clip 2 for Tuesday 1/19 session

- The solution we coded up is neat but relatively simple. We didn't demonstrate much sophisticated coding.
- At this stage, the interviewer might try to prod further, go deeper.
- Be prepared to answer questions about efficiency:
  - Can we solve the problem more quickly?
  - What if we knew more about the parameters?
  - What if we knew we'd be solving more instances of the same problem over and over?

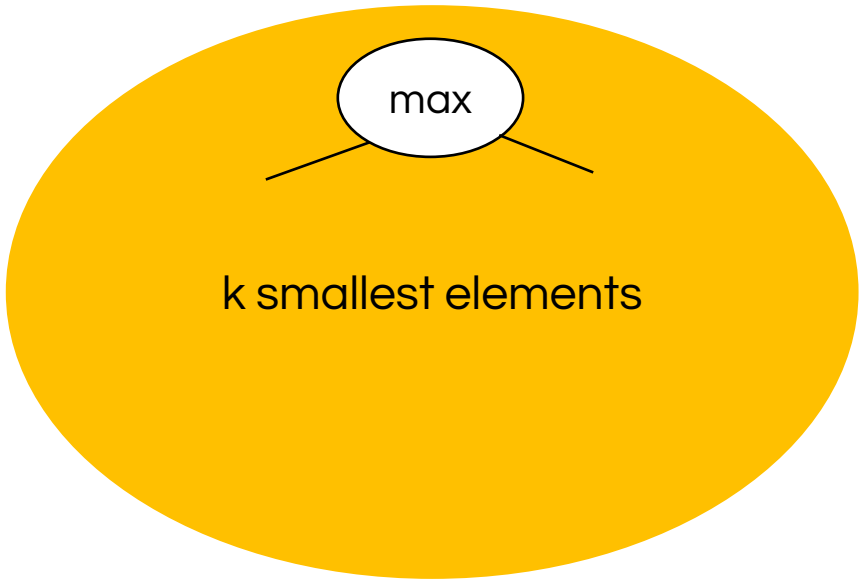
And beyond ...

Did we really need to sort the whole array?

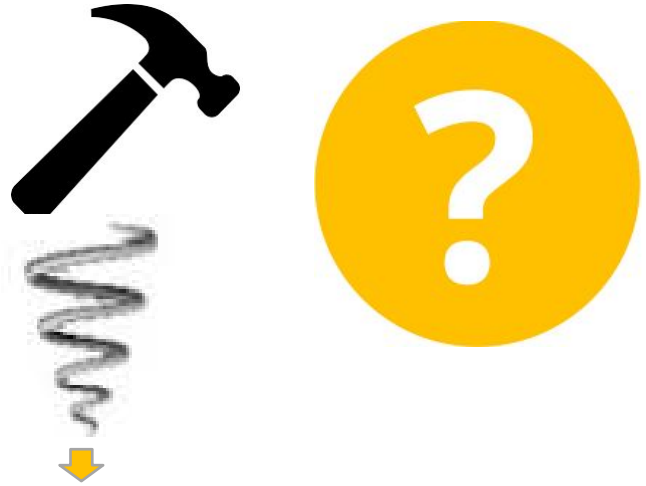
Using max-heap



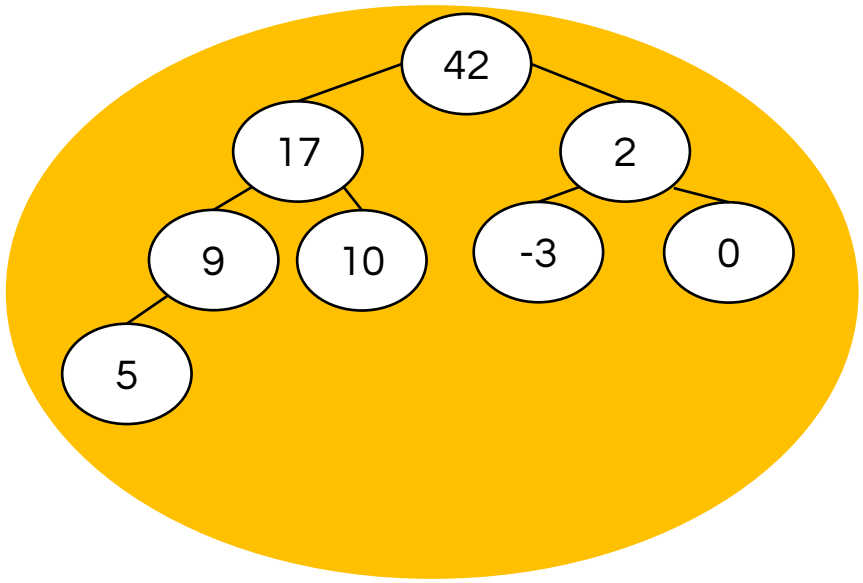
17	42	0	5	10	-3	2	9



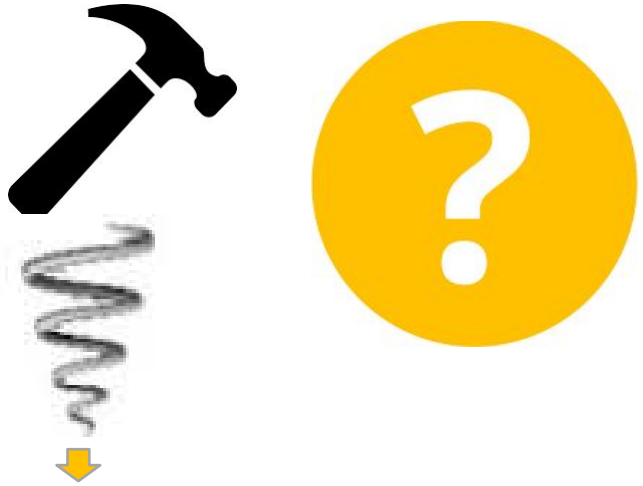
Using max-heap



17	42	0	5	10	-3	2	9

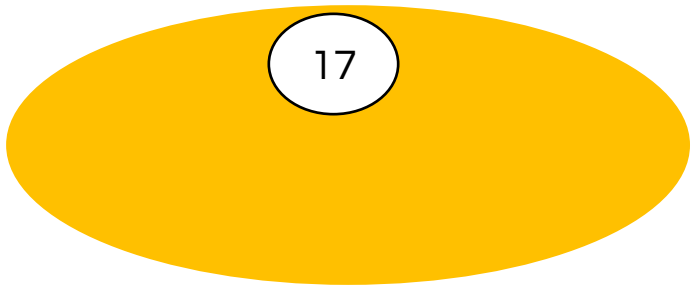


Using max-heap

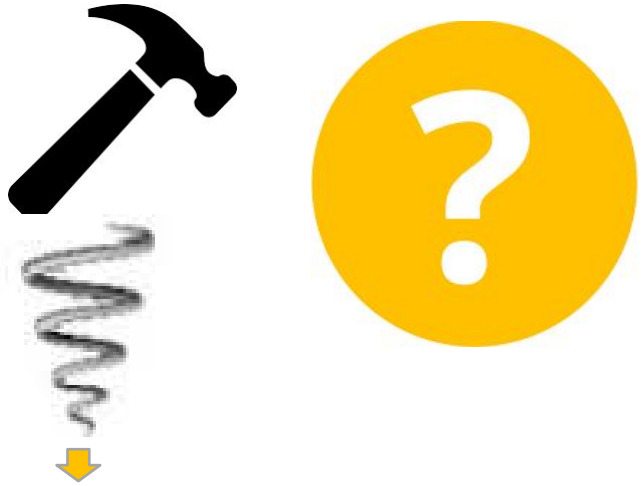


17	42	0	5	10	-3	2	9

k = 3?

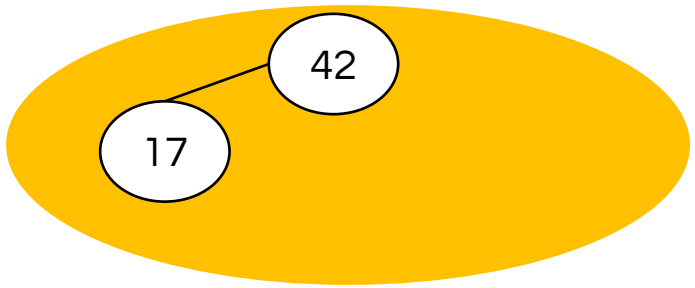


Using max-heap

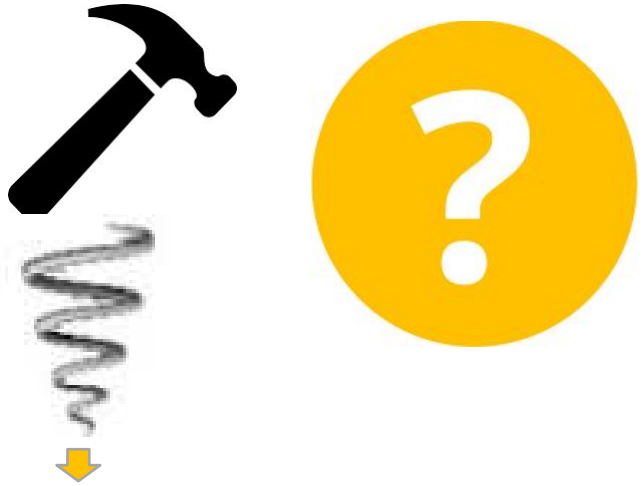


17	42	0	5	10	-3	2	9

k = 3?

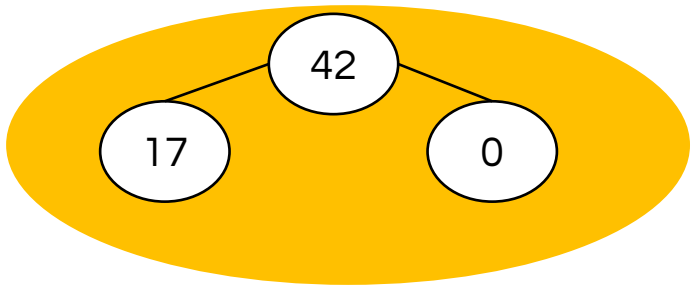


Using max-heap

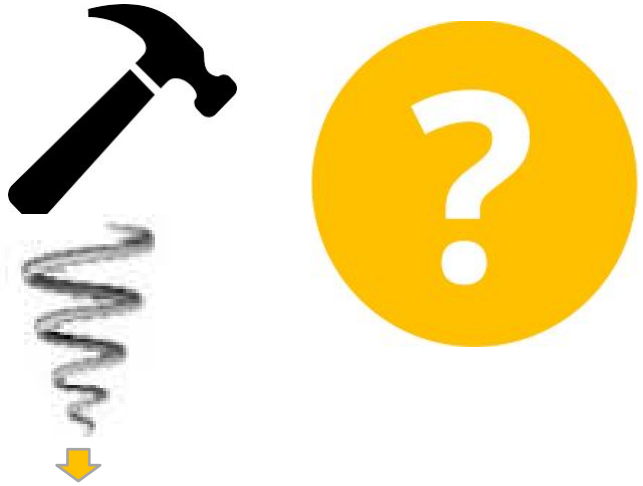


17	42	0	5	10	-3	2	9

k = 3?

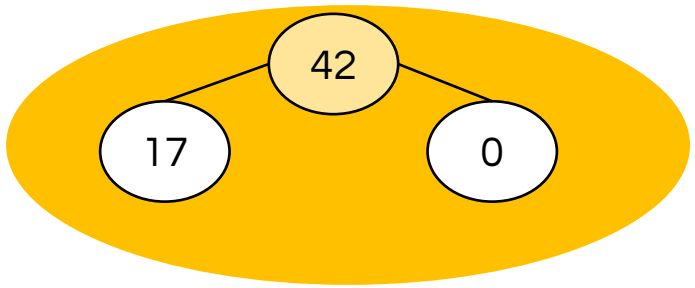


Using max-heap

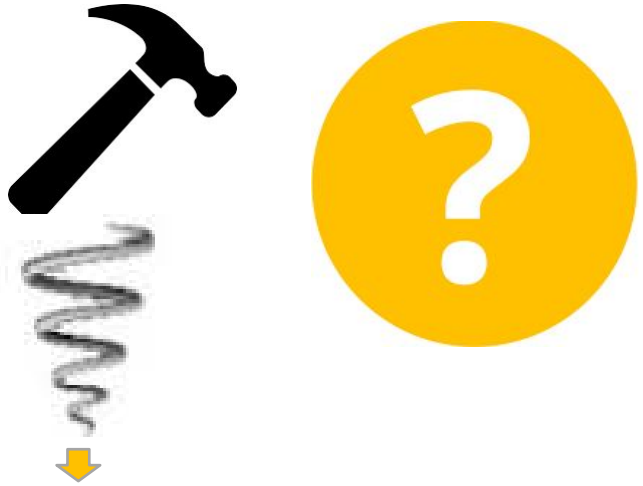


17	42	0	5	10	-3	2	9

k = 3?

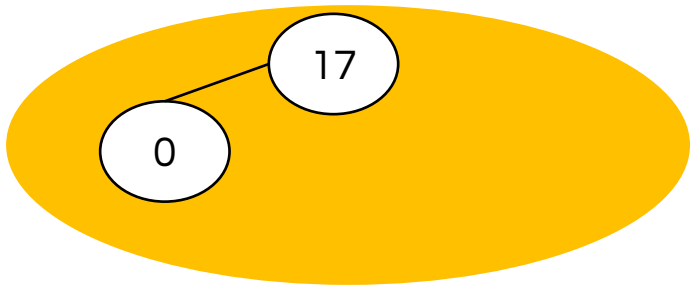


Using max-heap

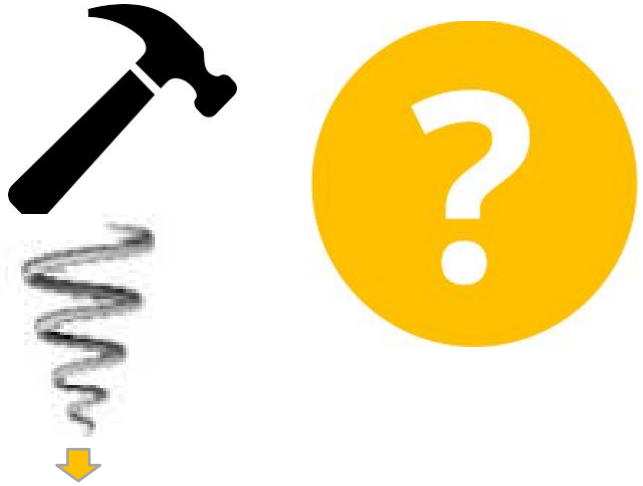


17	42	0	5	10	-3	2	9

k = 3?

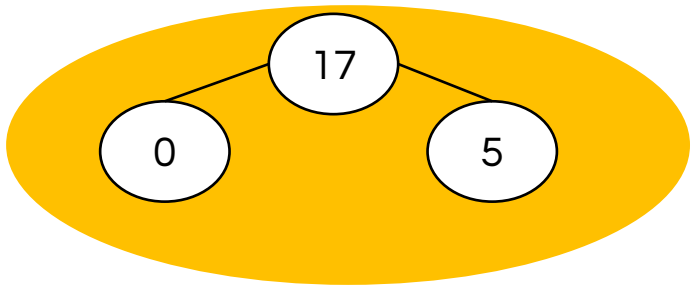


Using max-heap

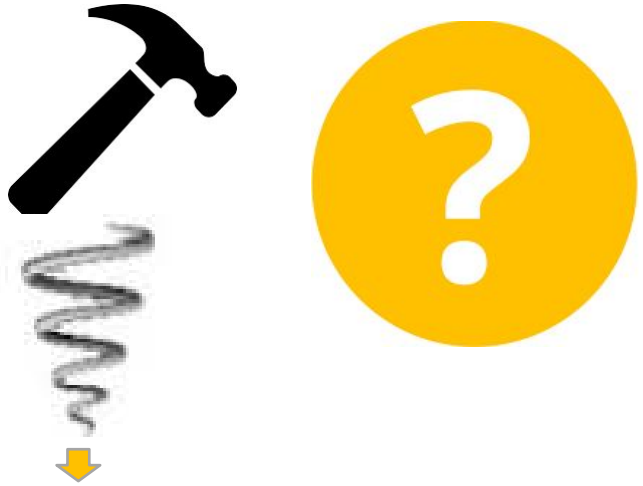


17	42	0	5	10	-3	2	9

k = 3?

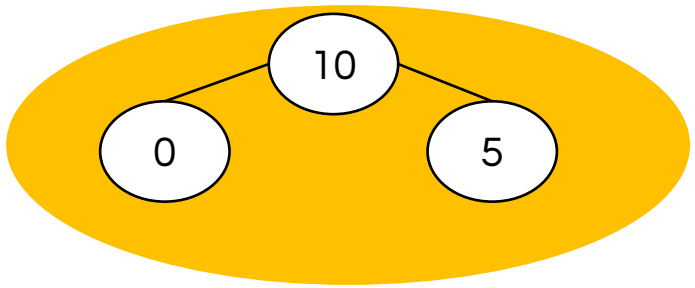


Using max-heap

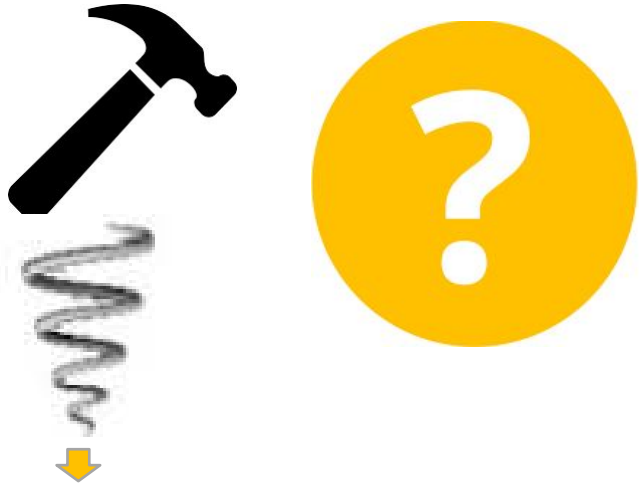


17	42	0	5	10	-3	2	9

k = 3?

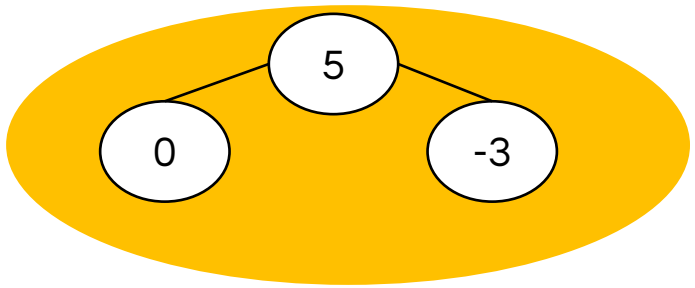


Using max-heap

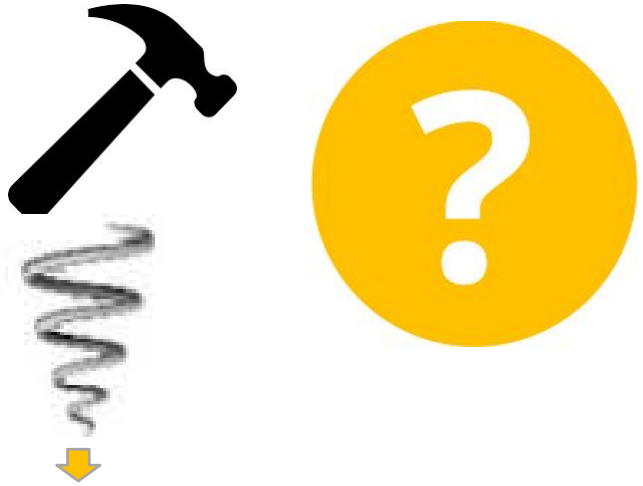


17	42	0	5	10	-3	2	9

k = 3?

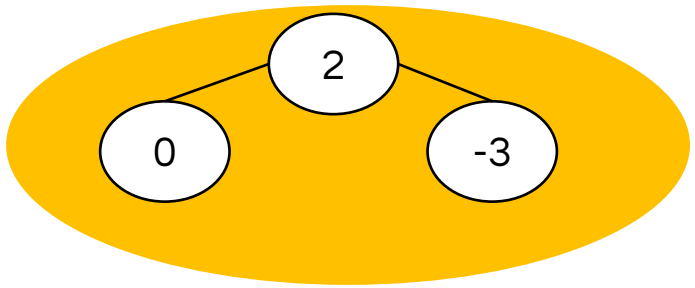


Using max-heap

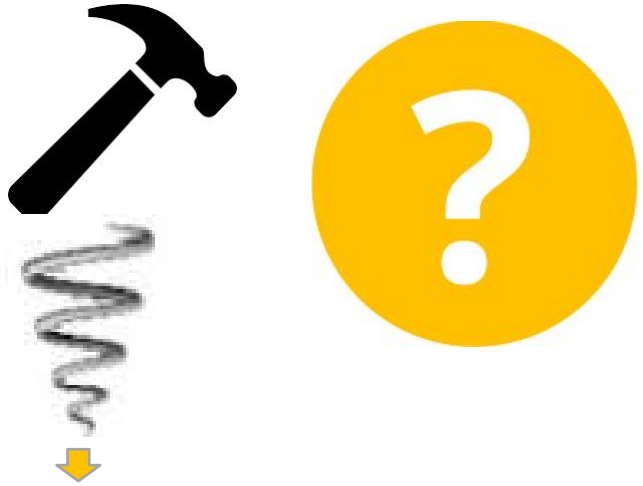


17	42	0	5	10	-3	2	9

k = 3?

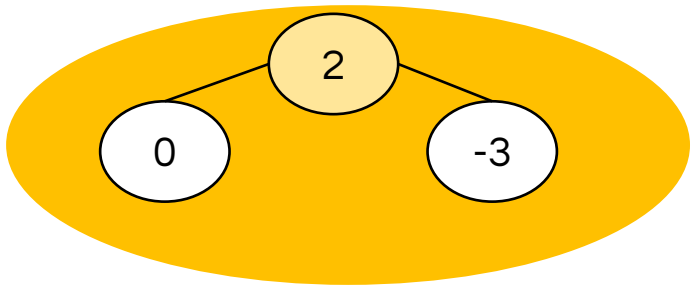


Using max-heap

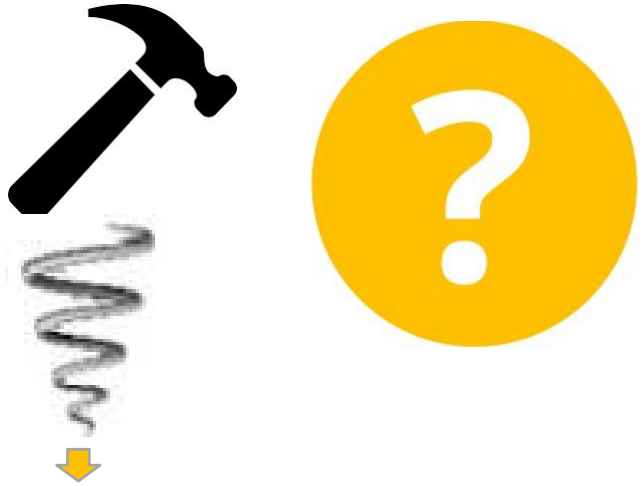


17	42	0	5	10	-3	2	9

k = 3?

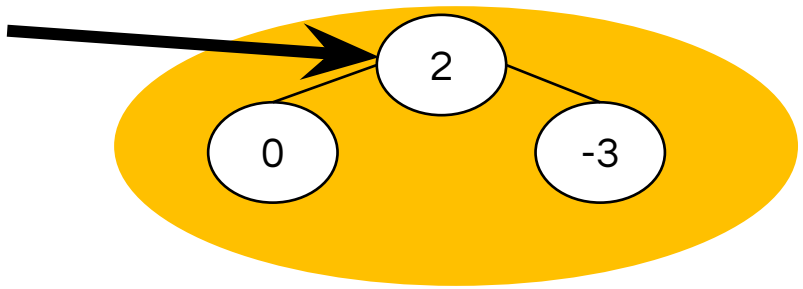


Using max-heap



17	42	0	5	10	-3	2	9

k = 3?



Using max-heap

```
public int kthSmallestViaHeap(int[] array, int k) {
  if (k <= 0 || k > array.length) { throw new IllegalArgumentException(); }
  PriorityQueue<Integer> smallestK = new
       PriorityQueue<Integer>(k, Collections.reverseOrder());
  for (int i = 0; i < Math.min(array.length, k); i++) {
    smallestK.add(array[i]);
  }
  if (k > array.length) {
    for (int j = k; j<array.length; j++) {
      if (array[j] < smallestK.peek()) {
        smallestK.remove();
        smallestK.add(array[j]);
      }
    }
  }
  return smallestK.peek();
}
```

Using max-heap

```
public int kthSmallestViaHeap(int[] array, int k) {
  if (k <= 0 || k > array.length) { throw new IllegalArgumentException(); }
  PriorityQueue<Integer> smallestK = new
       PriorityQueue<Integer>(k, Collections.reverseOrder());
  for (int i = 0; i < Math.min(array.length, k); i++) {
    smallestK.add(array[i]);
  }
  if (k > array.length) {
    for (int j = k; j<array.length; j++) {
      if (array[j] < smallestK.peek()) {
        smallestK.remove();
        smallestK.add(array[j]);
      }
    }
  }
  return smallestK.peek();
}
```

Using max-heap

```
public int kthSmallestViaHeap(int[] array, int k) {
  if (k <= 0 || k > array.length) { throw new IllegalArgumentException(); }
  PriorityQueue<Integer> smallestK = new
       PriorityQueue<Integer>(k, Collections.reverseOrder());
  for (int i = 0; i < Math.min(array.length, k); i++) {
    smallestK.add(array[i]);
  }
  if (k > array.length) {
    for (int j = k; j<array.length; j++) {
      if (array[j] < smallestK.peek()) {
        smallestK.remove();
        smallestK.add(array[j]);
      }
    }
  }
  return smallestK.peek();
}
```

Using max-heap

```
public int kthSmallestViaHeap(int[] array, int k) {
  if (k <= 0 || k > array.length) { throw new IllegalArgumentException(); }
  PriorityQueue<Integer> smallestK = new
       PriorityQueue<Integer>(k, Collections.reverseOrder());
  for (int i = 0; i < Math.min(array.length, k); i++) {
    smallestK.add(array[i]);
  }
  if (k > array.length) {
    for (int j = k; j<array.length; j++) {
      if (array[j] < smallestK.peek()) {
        smallestK.remove();
        smallestK.add(array[j]);
      }
    }
  }
  return smallestK.peek();
}
```

Using max-heap

```
public int kthSmallestViaHeap(int[] array, int k) {
  if (k <= 0 || k > array.length) { throw new IllegalArgumentException(); }
  PriorityQueue<Integer> smallestK = new
       PriorityQueue<Integer>(k, Collections.reverseOrder());
  for (int i = 0; i < Math.min(array.length, k); i++) {
    smallestK.add(array[i]);
  }
  if (k > array.length) {
    for (int j = k; j<array.length; j++) {
      if (array[j] < smallestK.peek()) {
        smallestK.remove();
        smallestK.add(array[j]);
      }
    }
  }
  return smallestK.peek();
}
```

O(n log k)

Or something completely different…

Sort then probe: O(n log n)

Insert in heap: O(n log k)

Or something completely different...

Pivot

17	42	0	5	10	-3	2	9

Or something completely different...

Pivot

17	42	0	5	10	-3	2	9

2	-3	0	5	10	42	9	17

Or something completely different...

Pivot

17	42	0	5	10	-3	2	9

2	-3	0	5	10	42	9	17

Or something completely different…

Pivot

17	42	0	5	10	-3	2	9

2	-3	0	5	10	42	9	17

Max is 3rd smallest element

Or something completely different…

Pivot

17	42	0	5	10	-3	2	9

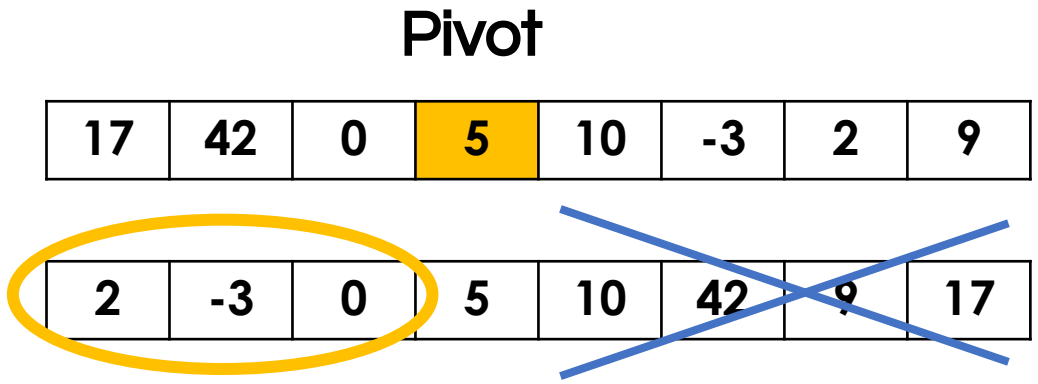
2	-3	0	5	10	42	9	17

Max is 3rd smallest element

```
public int kthSmallestSelectionRank(int[] array, int k) {
  if (k <= 0 || k > array.length) {
    throw new IllegalArgumentException();
  }
  return elementRank(array, 0, array.length - 1, k-1);
}

private int elementRank(int[] array, int left, int right, int k) {
  int pivotIndex = left + rand.nextInt(right+1-left);
  int pivot = array[pivotIndex];
  int sizeSmall = compareToPivot(array, left, right, pivot) - left + 1;
  if (sizeSmall-1 == k) {
    return pivot;
  }
  else if (sizeSmall > k) {
    return elementRank(array,left,left+sizeSmall-1,k);
  }
  else {
    return elementRank(array,left+sizeSmall,right,k-sizeSmall);
  }
}

...
```

```
public int kthSmallestSelectionRank(int[] array, int k) {
  if (k <= 0 || k > array.length) {
    throw new IllegalArgumentException();
  }
  return elementRank(array, 0, array.length - 1, k-1);
}

private int elementRank(int[] array, int left, int right, int k) {
  int pivotIndex = left + rand.nextInt(right+1-left);
  int pivot = array[pivotIndex];
  int sizeSmall = compareToPivot(array, left, right, pivot) - left + 1;
  if (sizeSmall-1 == k) {
    return pivot;
  }
  else if (sizeSmall > k) {
    return elementRank(array,left,left+sizeSmall-1,k);
  }
  else {
    return elementRank(array,left+sizeSmall,right,k-sizeSmall);
  }
}

...
```

Or something completely different…



Selection rank, a random algorithm

Performance: expected O(n) time

The punch line

Always keep going

→ Revisit assumptions
→ Consider performance
→ Iterate solutions

Practice

→ Can data structures help?
→ Apply or modify known algorithms