CSE 12 – Basic Data Structures and Object-Oriented Design
Lecture 21

Greg Miranda & Paul Cao, Winter 2021

This lecture is being recorded

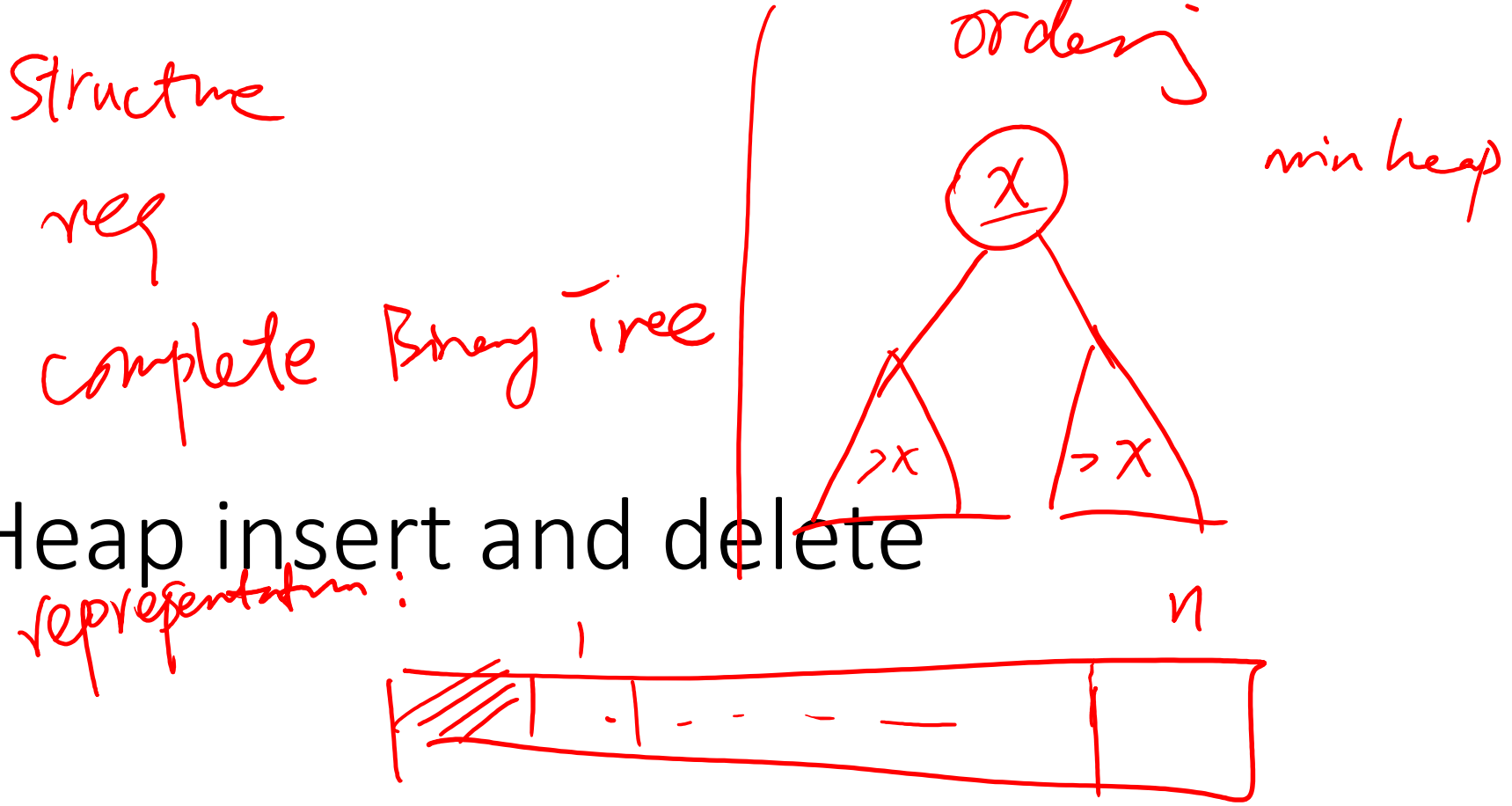
Announcements

- Quiz 21 due Wednesday @ 8am
- Survey 9 due Friday @ 11:59pm
- PA7 due tomorrow @ 11:59pm (Week 9)

Topics

- Heap operations
- Heap applications

Heap insert and delete

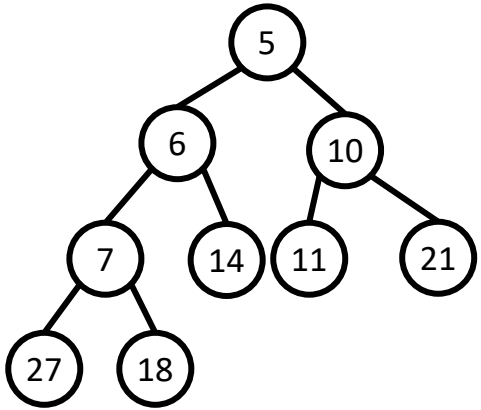


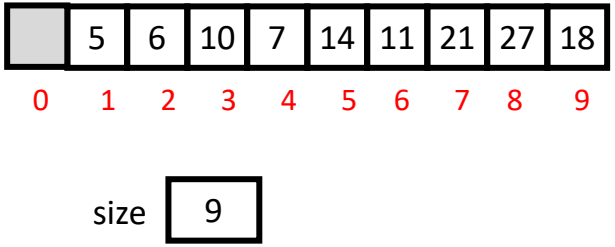
Removing from a heap

- When you remove an element from an array-backed heap as we have discussed them, at what index is the element to remove (the one you will return) located? Assume the variable `size` stores the number of elements currently in the heap, and `arr.length` is the length of the array storing the heap.

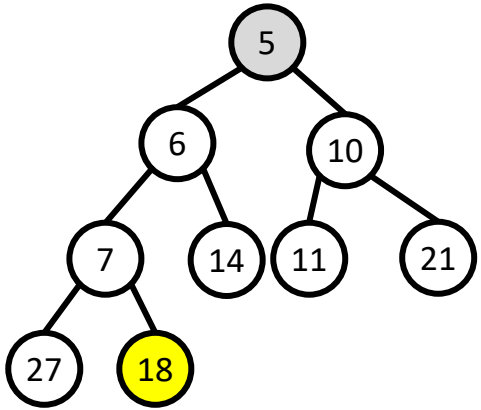
  A. 0
  B. 1
  C. size – 1
  D. arr.length – 1
  E. You can't tell with the information given

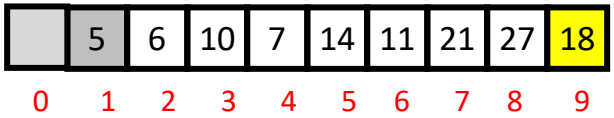
Removing from a heap (poll)





Removing from a heap



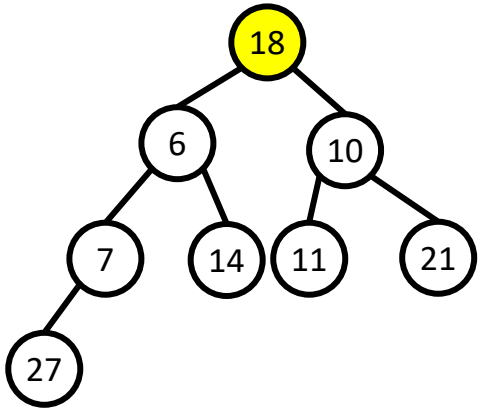


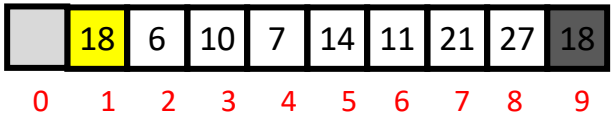
size 9

```
theHeap[1] = theHeap[size];
```

NOTE: The assignment uses ArrayLists not arrays

Removing from a heap





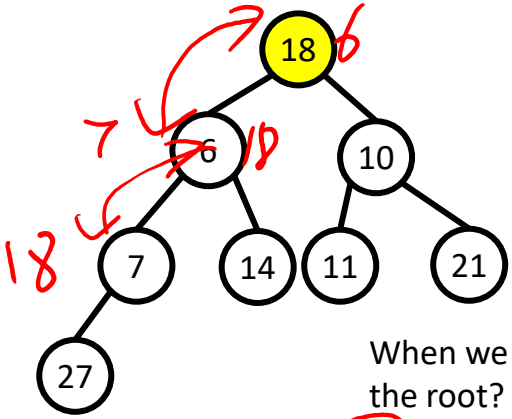
size 8

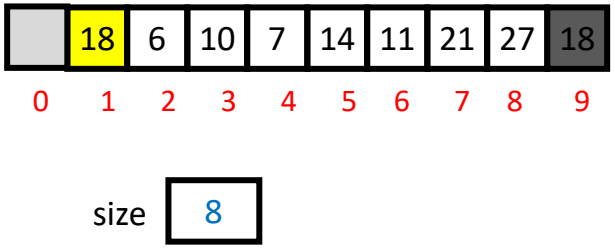
```
theHeap[1] = theHeap[size];
size--;
trickleDown( ____?____ );
```

Now 18 needs to trickle down… this should be a separate method. Can be iterative or recursive, but recursive is easier, really!

TrickleDown (min heap)

This is the main challenge of writing the heap, so I am not going to write it for you. But I will give you some hints and the general idea behind a recursive approach.



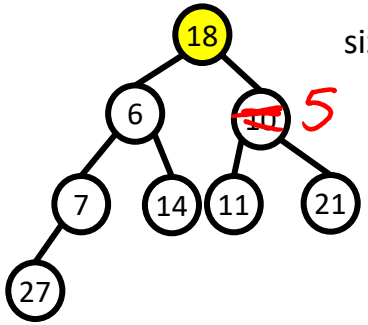


When we trickle 18 down, which value should become the root?

A. 6
B. 10
C. 27
D. 18 should stay there
E. Other

TrickleDown (min heap)

This is the main challenge of writing the heap, so I am not going to write it for you. But I will give you some hints and the general idea behind a recursive approach.



size 8

	18	6	10	7	14	11	21	27	18
0	1	2	3	4	5	6	7	8	9

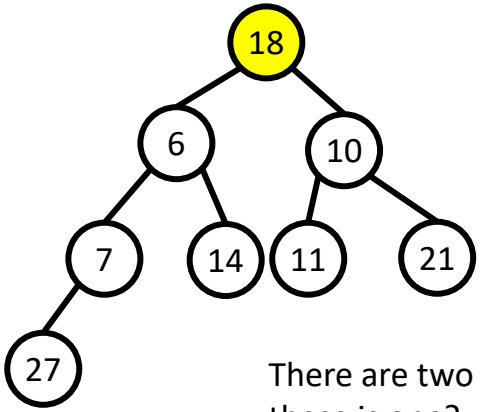
Assume the node to be trickled down is at `index` and that its left and right children are at `lInd` and `rInd`, respectively. Also assume the node at index has two children. Which line correctly completes the code below?

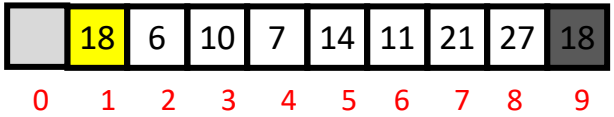
```
if ( ____________________ ) childInd = lInd;
else childInd = rInd;
swap( theHeap, childInd, index );//only swap if child > parent
```

A. `lInd < rInd`
B. `lInd < index`
C. `theHeap[lInd] < theHeap[rInd]`
D. `theHeap[lInd] < theHeap[index]`

TrickleDown (min heap)

This is the main challenge of writing the heap, so I am not going to write it for you. But I will give you some hints and the general idea behind a recursive approach.





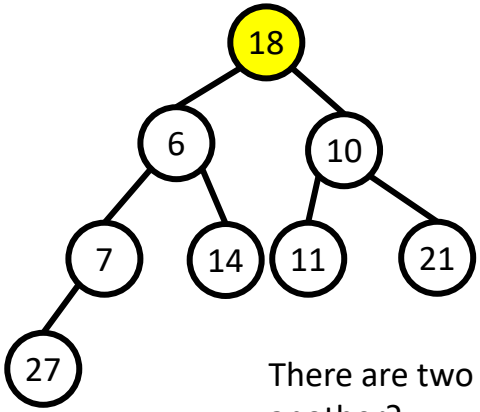
size 8

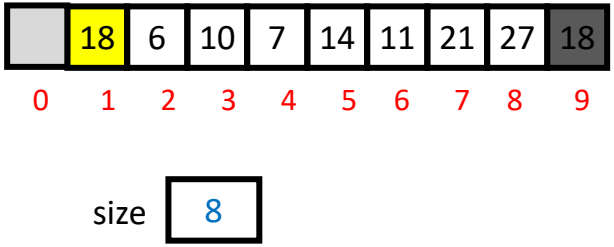
There are two base cases for this method. Which of these is one?

A. 18 is in a leaf node
B. 18 is in a node with one child
C. 18 is a node with 2 children
D. 18 is at the root of the heap

TrickleDown (min heap)

This is the main challenge of writing the heap, so I am not going to write it for you. But I will give you some hints and the general idea behind a recursive approach.



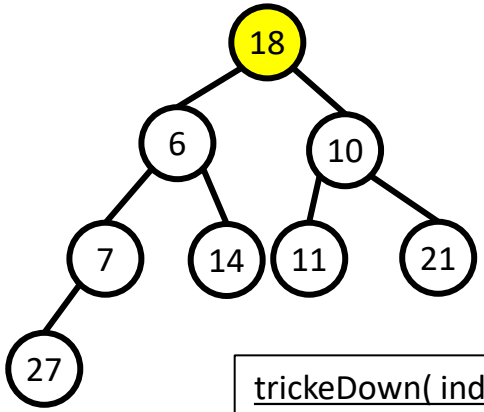


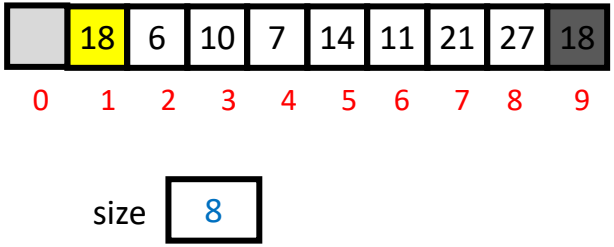
There are two base cases for this method. Which of these is another?

A. 18 has no children less than itself
B. 18 has no more than one child less than itself.
C. 18 has exactly one child, which is greater than or equal to it

TrickleDown (min heap)

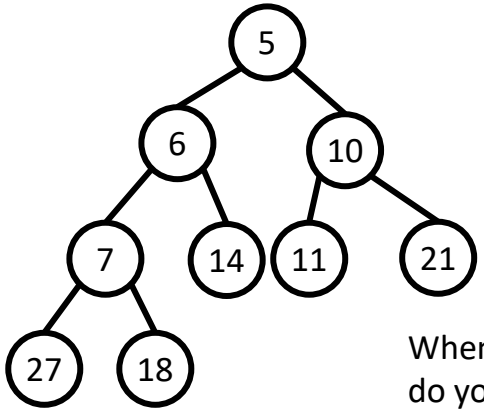
Here's a rough recursive algorithm for trickleDown. It's up to you to translate this to code! And careful, because there are subtleties not mentioned here (e.g., what if the node has only one child?

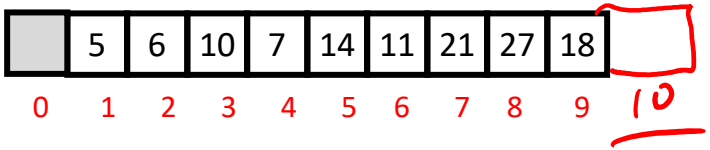




trickeDown(index) **// You might need more arguments**
If value at index is a leaf, return
If value at index has no children less than it, return
Swap value at index with its smaller child (at childInd)
trickleDown(childInd)

Adding to a heap (offer)





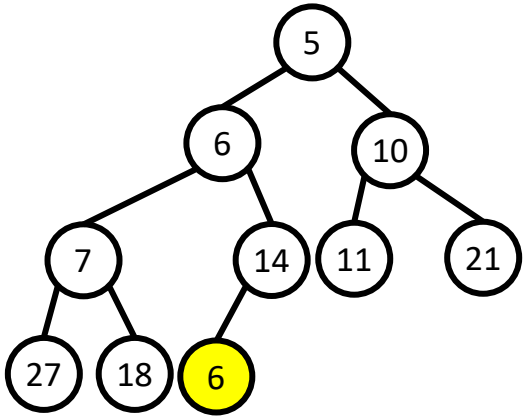
size 9

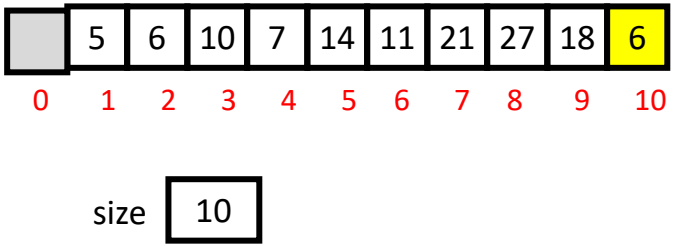
When you add an element to a heap, at what index do you put it initially?

A. 0
B. size – 1
C. size
D. Somewhere else

depending on when size is ++

Adding to a heap (offer)

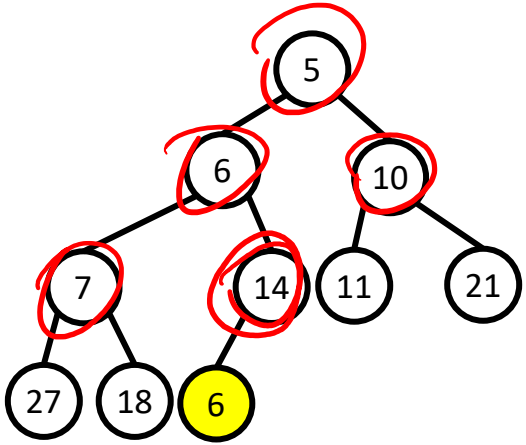


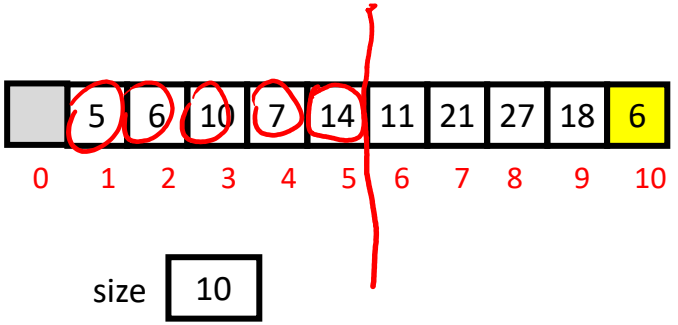


BubbleUp

faster way for heap: heapify every parent (from the lowest to highest)

For offer, you will need a helper method called BubbleUp. we also suggest recursion, and here's a very rough recursive algorithm, though it's up to you to figure out the base case.



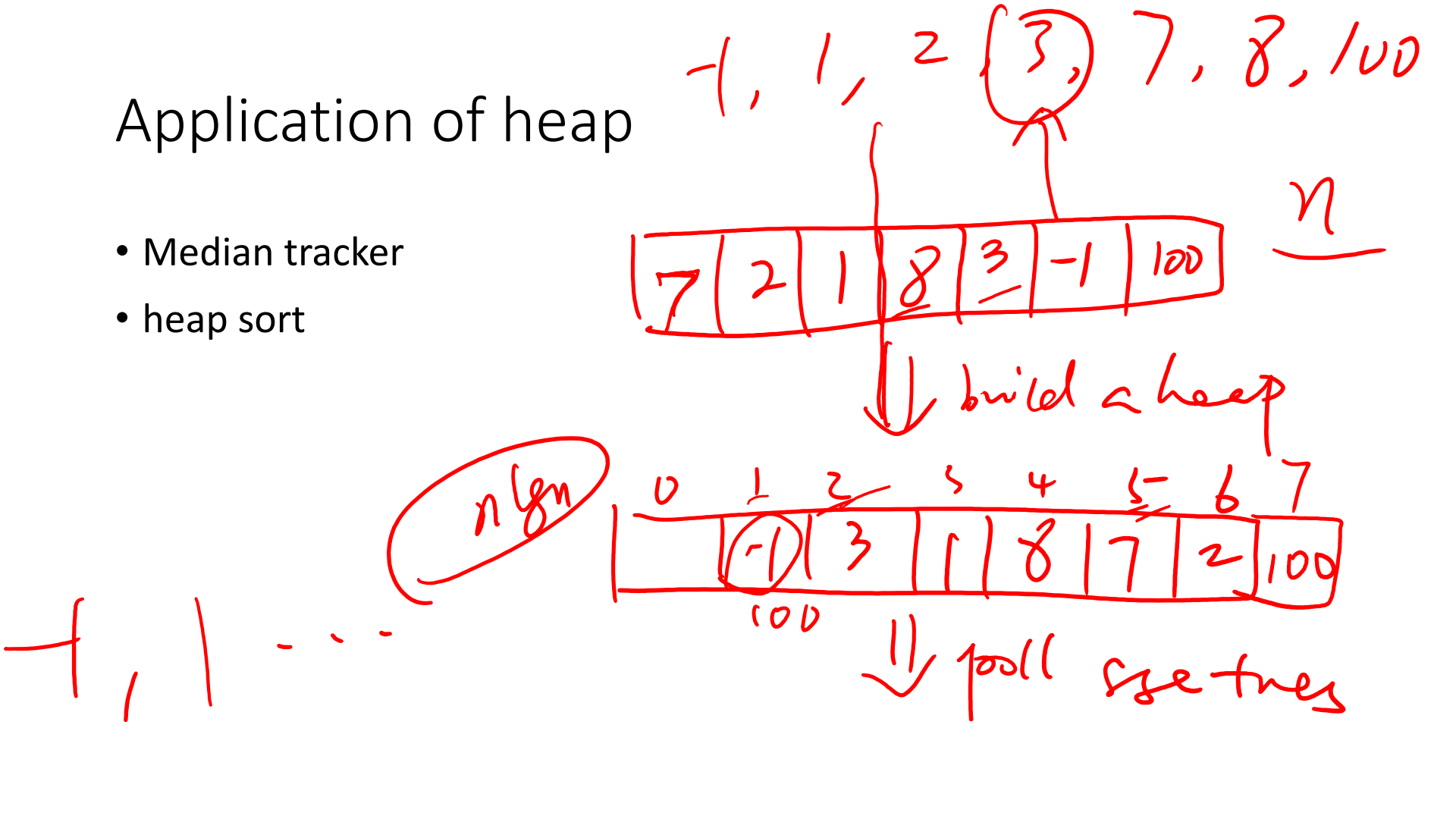



```
bubbleUp( index ) // You might need more arguments
  If value at index shouldn't move any more, return
  Swap value at index with its parent (at parentInd)
  bubbleUp( parentInd )
```

This is the base case for you to figure out.
HINT: There will be more than one

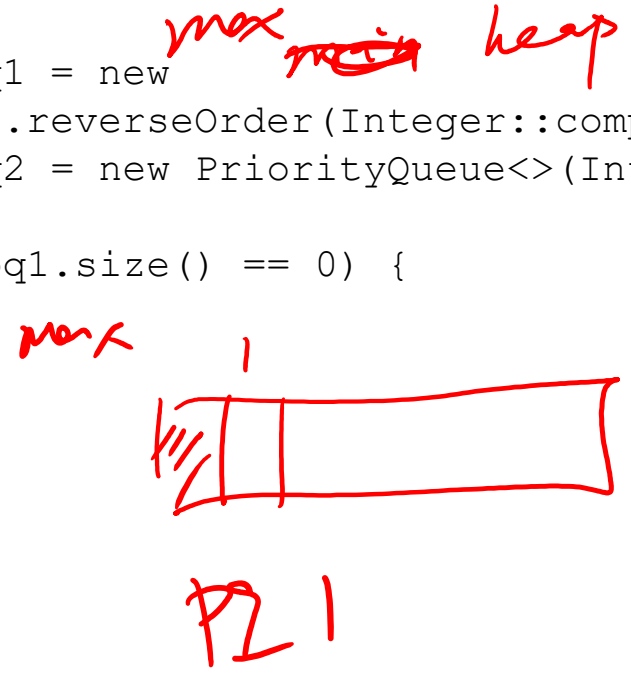
Application of heap

- Median tracker
- heap sort



```
class Tracker {
  PriorityQueue<Integer> pq1 = new
PriorityQueue<>(Collections.reverseOrder(Integer::compare));
  PriorityQueue<Integer> pq2 = new PriorityQueue<>(Integer::compare);
  void add(int n) {
    if(pq2.size() == 0 && pq1.size() == 0) {
      pq2.add(n);
      return;
    }
    int current = get();
    if(n >= current) {
      pq2.add(n);
    }
    else {
      pq1.add(n);
    }
    int sizeDifference = pq2.size() - pq1.size();
    if(sizeDifference > 1) { pq1.add(pq2.poll()); }
    else if(sizeDifference < -1) { pq2.add(pq1.poll()); }
  }

}
```



```
class Tracker {
 PriorityQueue<Integer> pq1 = new     PriorityQueue<>(Collections.reverseOrder(Integer::compare));
 PriorityQueue<Integer> pq2 = new PriorityQueue<>(Integer::compare);
 void add(int n) {
  if(pq2.size() == 0 && pq1.size() == 0) {
   pq2.add(n);
   return;
  }
  int current = get();
  if(n >= current) {
   pq2.add(n);
  }
  else {
   pq1.add(n);
  }
  int sizeDifference = pq2.size() - pq1.size();
  if(sizeDifference > 1) { pq1.add(pq2.poll()); }
  else if(sizeDifference < -1) { pq2.add(pq1.poll()); }
 }

}
```

median

```
int get() {
  if(pq2.size() == pq1.size()) { return (pq2.peek() + pq1.peek()) / 2; }
  if(pq2.size() > pq1.size()) { return pq2.peek(); }
  else { return pq1.peek(); }
 }

 public String toString() {
  return "" + pq1 + " " + this.get() + " " + pq2;
 }
```

MedianTracker (draw the picture and arrays)

- Draw the picture and the arrays for the following:
  - Add the following elements to the MedianTracker (in this order):
    - 5, 10, 15, 20, 25, 30, 35
  - What is the result of the call to get() after adding all the elements?

Questions on Lecture 21?