Chapter 3, Case Studies:

- Linear-time sorting algorithms (last week Monday and Wednesday)
- Disjoint sets and array forests (last week Friday)
- Priority queues (**Monday and Wednesday**)
- *N*-sets and bit vectors (Thursday in lab)
- (Start graphs Friday)

Today:

- Recent HW and quiz problems
- Worklist algorithms
- Priority queue ADT (problem statement)
- Inefficient solutions
- Abstractions for the heap data structure
- Heap implementation details, part 1
- Excursion: heap sort
- Heap implementation details, part 2
- Analysis and optimization

```
static Node arrayToList1(int[] array) {
    Node toReturn = new Node(array[0], null);
    for (int i = 1; i < array.length; i++) {
        Node current = toReturn;
        while (current.next() != null)
            current = current.next();
        current.setNext(new Node(array[i], null));
    }
    return toReturn;
}

Node arrayToList2(int[] array) {
    Node toReturn = null;
    for (int i = array.length - 1; i >= 0; i--)
        toReturn = new Node(array[i], toReturn);
    return toReturn;
}

static int[] listToArray(Node head) {
    int size = 0;
    for (Node current = head; current != null; current = current.next())
        size++;
    int[] toReturn = new int[size];
    int i = 0;
    for (Node current = head; current != null; current = current.next())
        toReturn[i++] = current.datum();
    return toReturn;
}
```

```
public class ArrayList<E>
                implements List<E> {
    private E[] internal;
    private int size;
...
}
```

```
public class ArrayStack<E>
                implements Stack<E> {
    private E[] internal;
    private int size;
...
}
```

Informal: The positions of `internal` in range $[0, \texttt{size})$ contain the elements in the conceptual list in order.

Formal:

(a) $\texttt{size} \leq \texttt{internal.length}$

(b) `size` equals the number of calls to `insert()` and `add()` minus the number of (non-empty) calls to `remove()`.

Informal: `size` is the number of items in the conceptual stack, which appear in order of arrival in range $[0, \texttt{size})$

Formal:

(a) $\texttt{size} \leq \texttt{internal.length}$

(b) `size` equals the number of calls to `push()` minus the number of (non-empty) calls to `pop()`.

(c) For all $i \in [0, \texttt{size} - 1)$, the item in $\texttt{internal}[i]$ was pushed before the item in $\texttt{internal}[i + 1]$.

```
private class PlainFind implements FindStrategy {
    public int find(int p) {
        while (p != parents[p]) p = parents[p];
        return p;
    }
}
```

Let p_{orig} be the original value of p. Let i be the number of iterations completed.

(a) $0 \leq p < N$.

(b) $leader(p) = leader(p_{orig})$

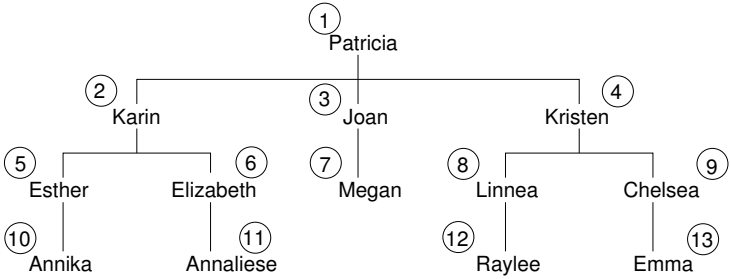
(c) $p = \overbrace{p[\ \ ]}^{i \text{ times}} p_{orig}$

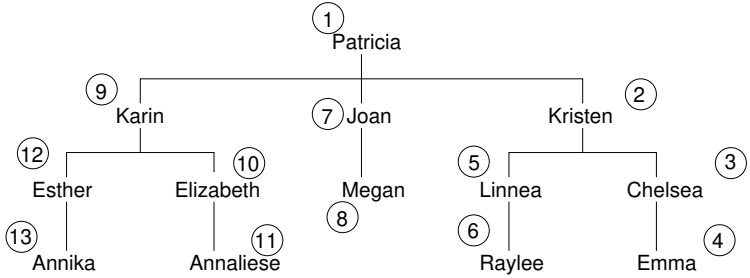
- An array forest is an example of what kind of data structure?
  It contains both array-based and linked concepts.

- For each, is it best considered a data structure, abstraction, or abstract data type?

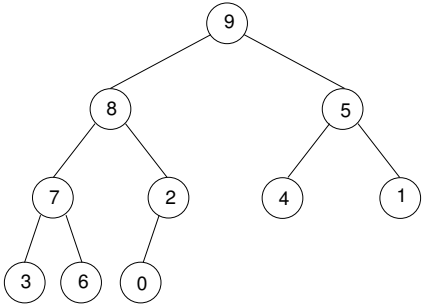
Array	Data structure (simple)
Array Forest	Data structures (hybrid/advanced)
Disjoint set	Abstract data type
Forest	Abstraction

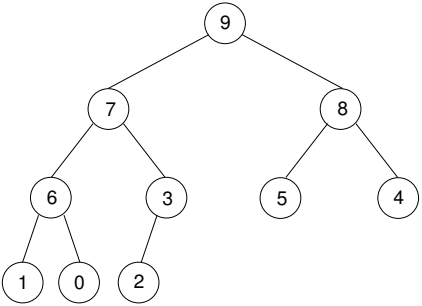
- What design pattern was (explicitly) used in the implementation of disjoint sets?
  Strategy

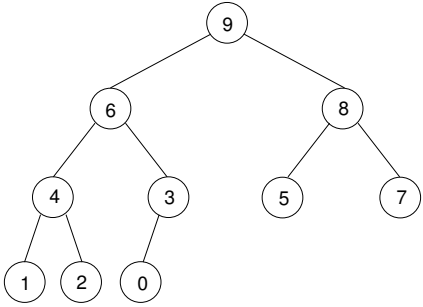

1
Patricia
2
Karin
3
Joan
4
Kristen
5
Esther
6
Elizabeth
7
Megan
8
Linnea
9
Chelsea
10
Annika
11
Annaliese
12
Raylee
13
Emma

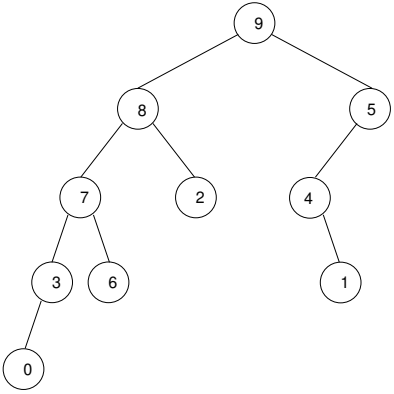

1
Patricia
9
Karin
7
Joan
2
Kristen
12
Esther
10
Elizabeth
Megan
8
5
Linnea
3
Chelsea
13
Annika
11
Annaliese
6
Raylee
4
Emma

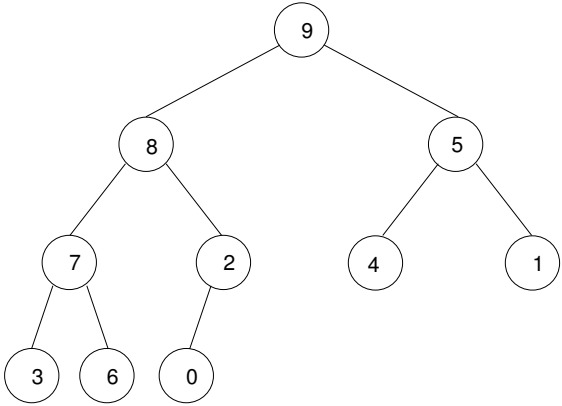
	`ListPriorityQueue`	`SortedListPriorityQueue`
Initialize empty	$\Theta(1)$	$\Theta(1)$
Initialize populated	$\Theta(n)$	$\Theta(n^2)$
`insert`	$\Theta(1)$	$\Theta(n)$
`max`	$\Theta(n)$	$\Theta(1)$
`extractMax`	$\Theta(n)$	$\Theta(1)$
`contains`	$\Theta(n)$	$\Theta(n)$
`increaseKey`	$\Theta(1)$	$\Theta(n)$
`decreaseKey`	$\Theta(1)$	$\Theta(n)$


9
8
5
7
2
4
1
3
6
0

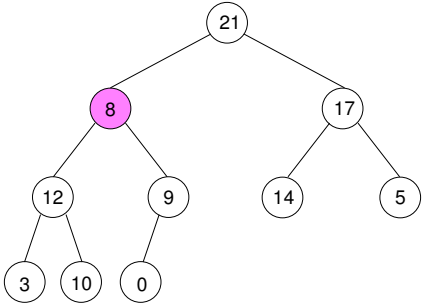

9
7
8
6
3
5
4
1
0
2

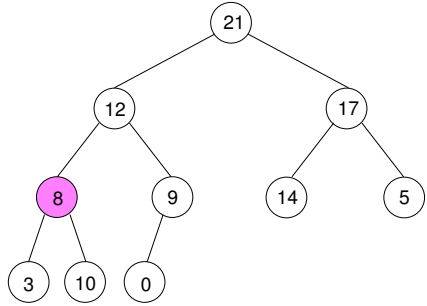

9
6
8
4
3
5
7
1
2
0

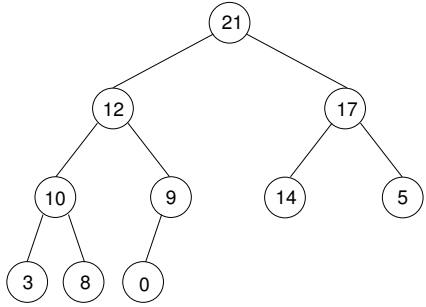

9
8
5
7
2
4
3
6
1
0

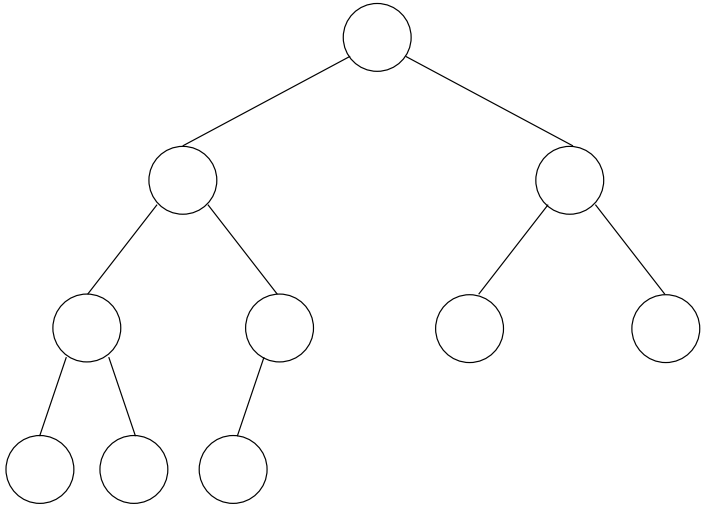


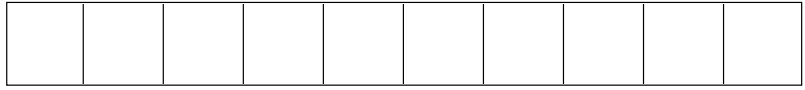
9	8	5	7	2	4	1	3	6	0


21
8
17
12
9
14
5
3
10
0

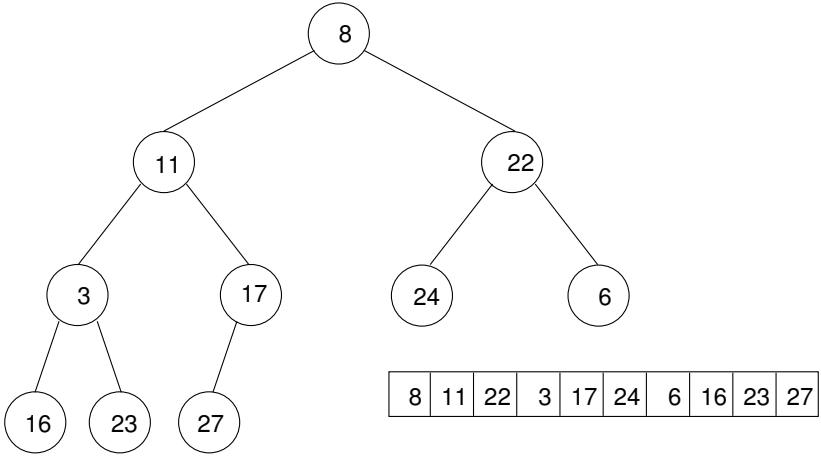

21
12
17
8
9
14
5
3
10
0

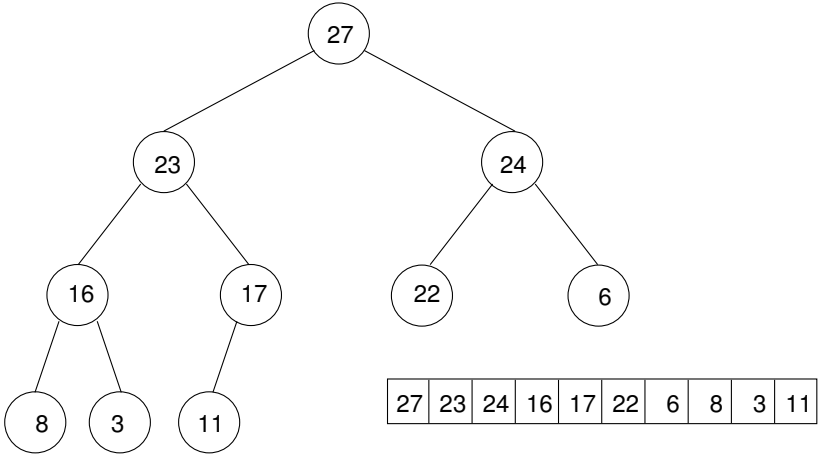

21
12
17
10
9
14
5
3
8
0



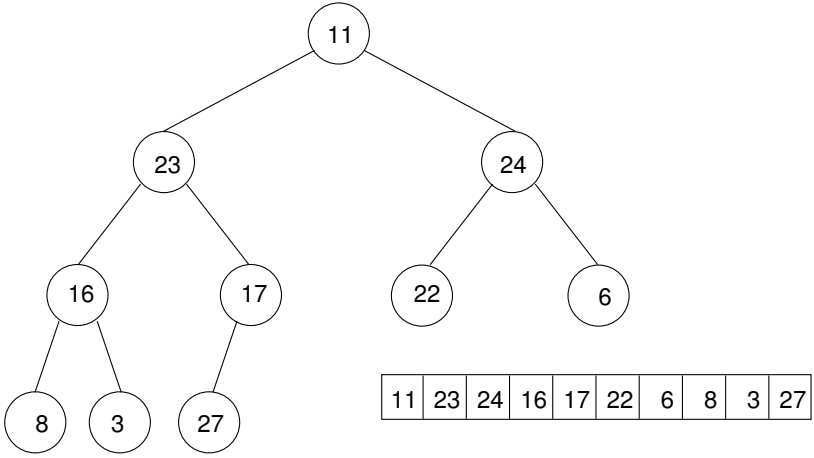


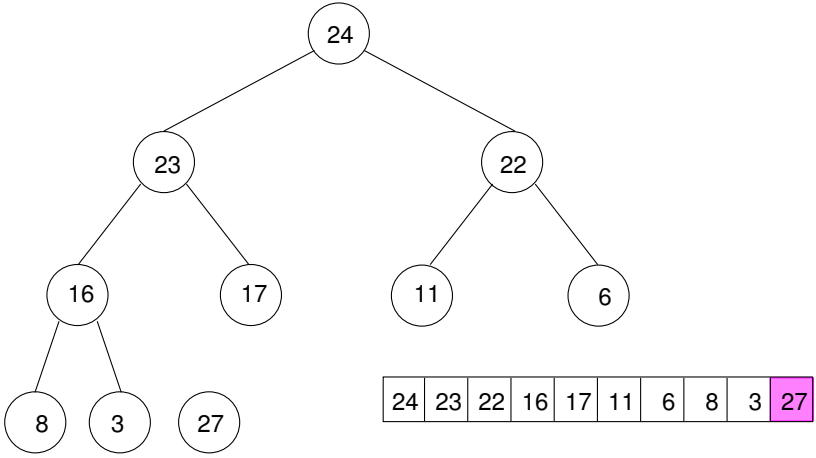
$$
\begin{aligned}
\sum_{i=0}^{h-1} 2^i(h-1-i) &= (h-1)\sum_{i=0}^{h-1} 2^i \;-\; \sum_{i=0}^{h-1} i2^i \\
&= (h-1)(2^h-1) - 2 - (h-2)2^h \\
&= h2^h - 2^h - h + 1 - 2 - h2^h + 2\cdot 2^h \\
&= 2^h - h - 1 \\
&= 2^{\lg(n+1)} - \lg(n+1) - 1 \\
&= n + 1 - \lg(n+1) - 1 \\
&= n - \lg(n+1)
\end{aligned}
$$

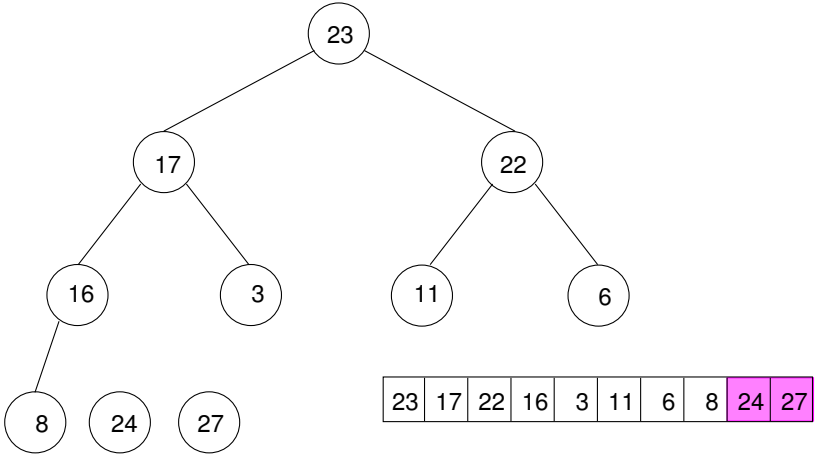

8
11
22
3
17
24
6
16
23
27
8 11 22 3 17 24 6 16 23 27

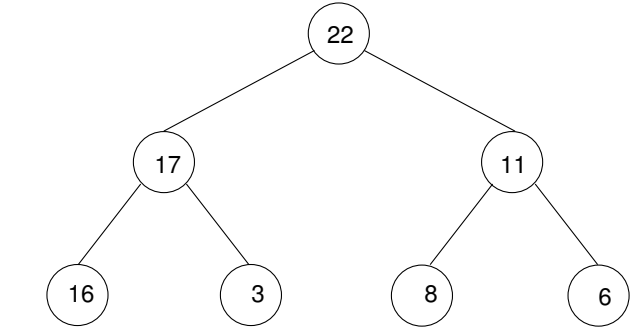


27	23	24	16	17	22	6	8	3	11

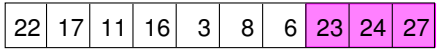

11
23
24
16
17
22
6
8
3
27
11 23 24 16 17 22 6 8 3 27

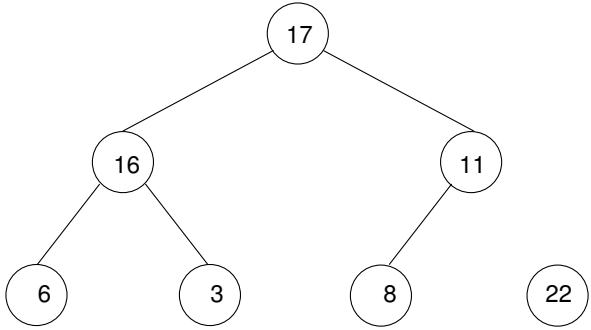

24
23
22
16
17
11
6
8
3
27
24 23 22 16 17 11 6 8 3 27


23
17
22
16
3
11
6
8
24
27
23 17 22 16 3 11 6 8 24 27

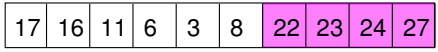

22
17
11
16
3
8
6

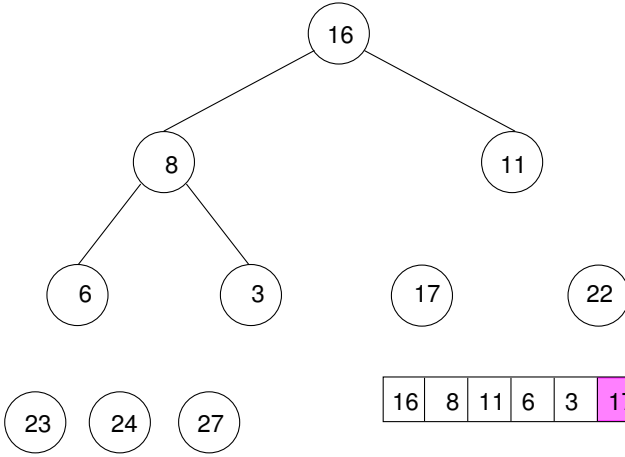

23
24
27


22
17
11
16
3
8
6
23
24
27

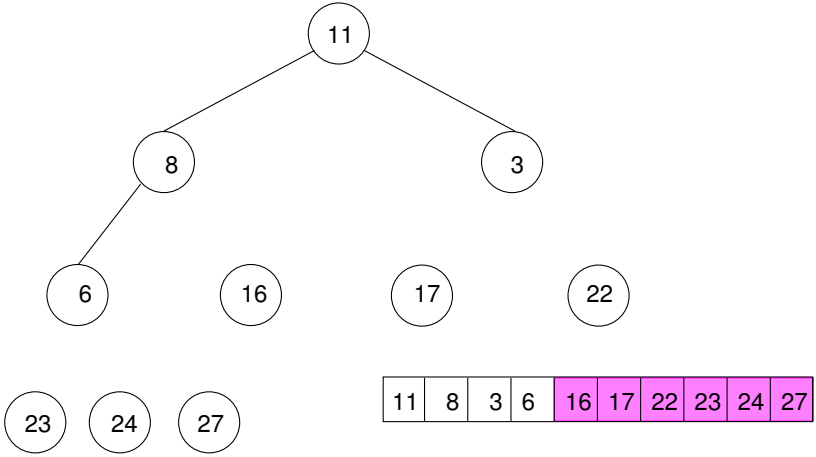

17
16
11
6
3
8
22

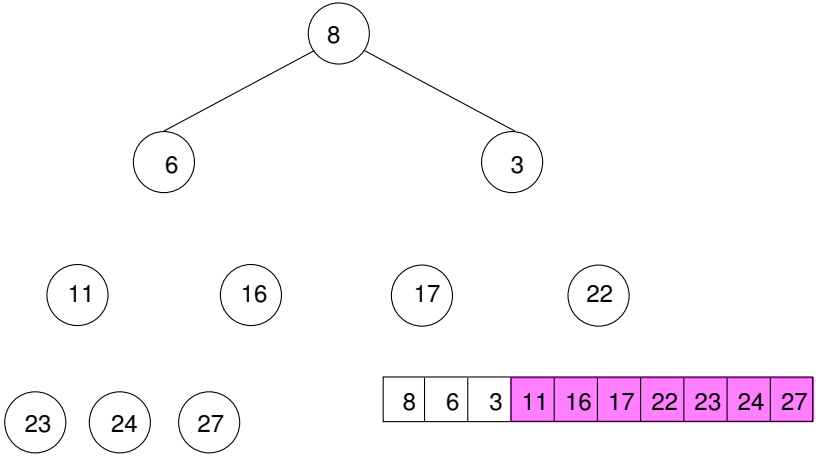

23
24
27

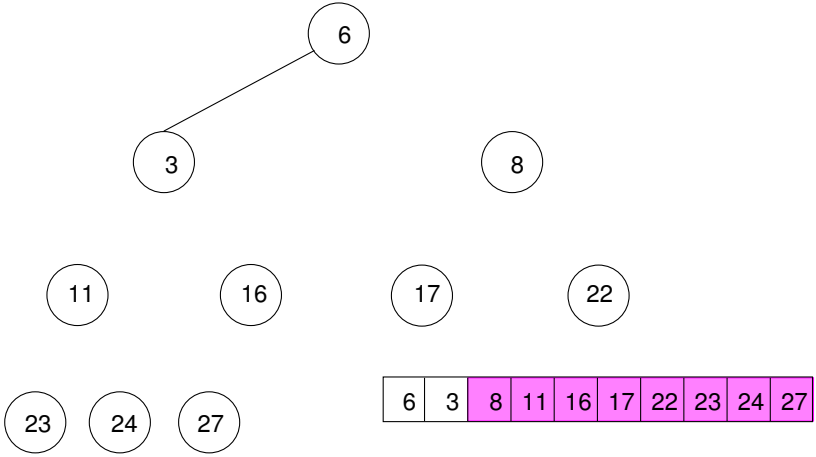

17
16
11
6
3
8
22
23
24
27

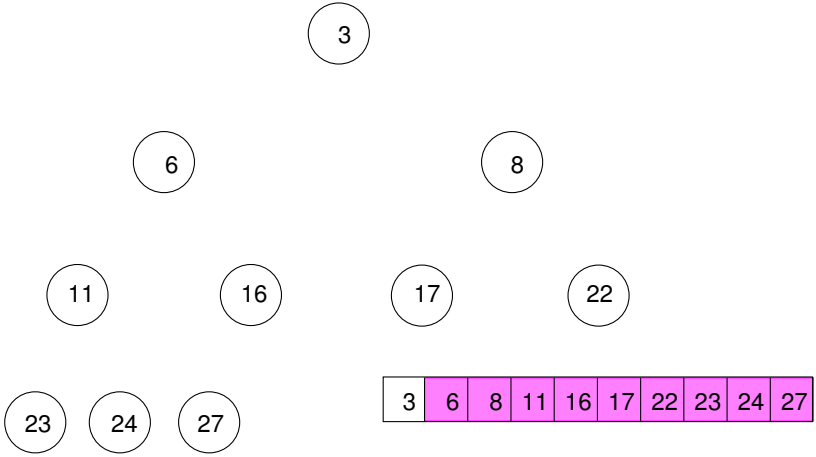


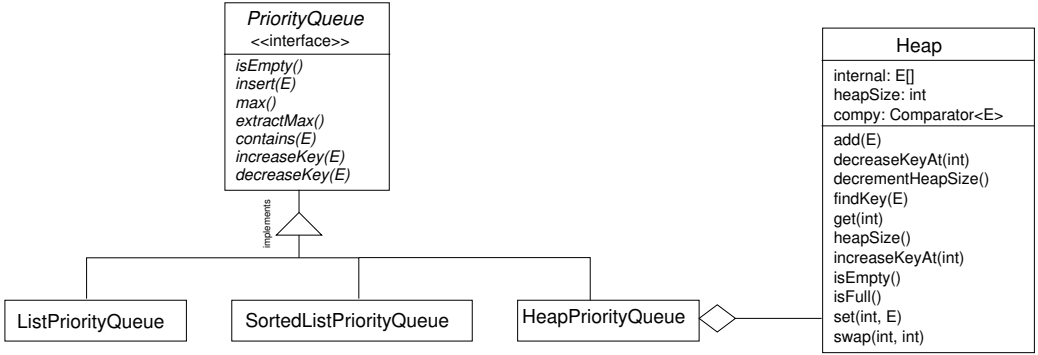
16	8	11	6	3	17	22	23	24	27


11
8
3
6
16
17
22
23
24
27
11 8 3 6 16 17 22 23 24 27


8
6
3
11
16
17
22
23
24
27
8 6 3 11 16 17 22 23 24 27


6
3
8
11
16
17
22
23
24
27
6
3
8
11
16
17
22
23
24
27


3
6
8
11
16
17
22
23
24
27
3 6 8 11 16 17 22 23 24 27


PriorityQueue
<<interface>>
isEmpty()
insert(E)
max()
extractMax()
contains(E)
increaseKey(E)
decreaseKey(E)
implements
Heap
internal: E[]
heapSize: int
compy: Comparator<E>
add(E)
decreaseKeyAt(int)
decrementHeapSize()
findKey(E)
get(int)
heapSize()
increaseKeyAt(int)
isEmpty()
isFull()
set(int, E)
swap(int, int)
ListPriorityQueue
SortedListPriorityQueue
HeapPriorityQueue

	`ListPriorityQueue`	`SortedPriorityQueue`	`HeapPriorityQueue`
Initialize empty	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Initialize populated	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n)$
`insert`	$\Theta(1)$	$\Theta(n)$	$\Theta(\lg n)$
`max`	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$
`extractMax`	$\Theta(n)$	$\Theta(1)$	$\Theta(\lg n)$
`contains`	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
`increaseKey`	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
`decreaseKey`	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$

Coming up: (all end-of-day)

Do **linear sorting** *project (due today, Mon, Feb 10)*
Do **heaps and priority queue** *project (due next Wed, Feb 19)*

Due **Wed, Feb 12***:*
Read Section 3.3 (heaps and priority queues)
Take heap/pq quiz

Due **Thurs, Feb 13***:*
Read Section 3.4 (N-sets and bit vectors)
Do Exercises 3.(26 & 27).
Take N-sets quiz