https://algs4.cs.princeton.edu

2.4 Priority Queues

- *API and elementary implementations*
- *binary heaps*
- *heapsort*
- *event-driven simulation (see videos)*

https://algs4.cs.princeton.edu

2.4 Priority Queues

- *API and elementary implementations*
- *binary heaps*
- *heapsort*
- *event-driven simulation (see videos)*

Collections

A collection is a data type that stores a group of items.

data type	core operations	data structure
stack	PUSH, POP	*linked list, resizing array*
queue	ENQUEUE, DEQUEUE	*linked list, resizing array*
priority queue	INSERT, DELETE-MAX	*binary heap*
symbol table	PUT, GET, DELETE	*binary search tree, hash table*
set	ADD, CONTAINS, DELETE	*binary search tree, hash table*

" Show me your code and conceal your data structures, and I shall continue to be mystified. Show me your data structures, and I won't usually need your code; it'll be obvious." — *Fred Brooks*

Priority queue

Collections allow adding and removing items. Which item to remove?

Stack. Remove the item most recently added.

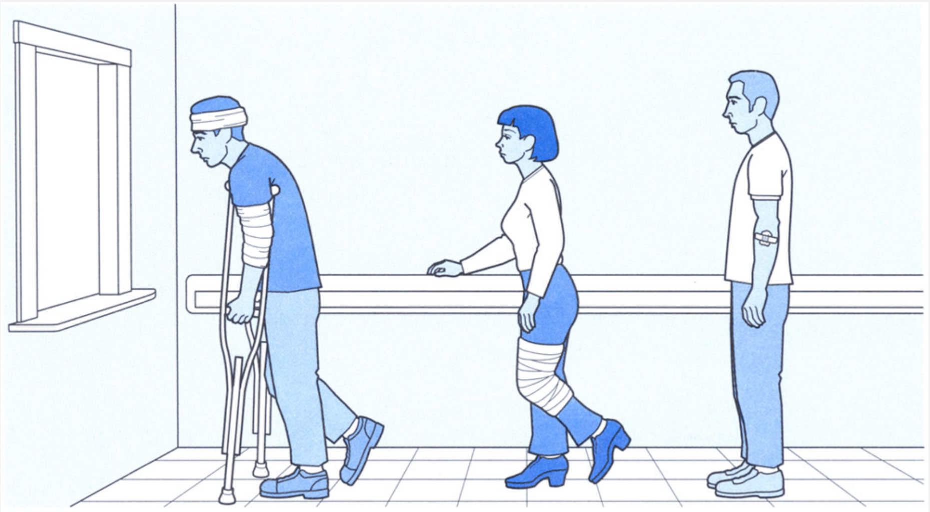
Queue. Remove the item least recently added.

Randomized queue. Remove a random item.

Priority queue. Remove the largest (or smallest) item.

Generalizes: stack, queue, randomized queue.

operation	*argument*	*return value*
insert	P	
insert	Q	
insert	E	
remove max		Q
insert	X	
insert	A	
insert	M	
remove max		X
insert	P	
insert	L	
insert	E	
remove max		P



Priority queue API

Requirement. Keys are generic; they must also be `Comparable`.

Key must be Comparable
("bounded type parameter")

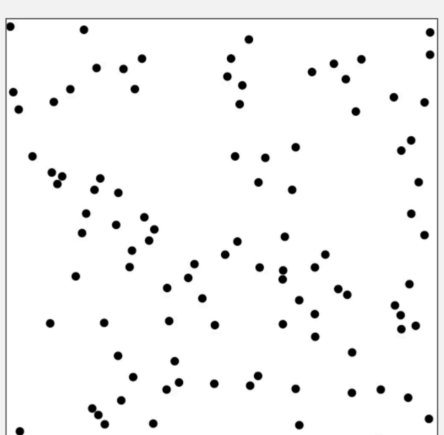
`public class MaxPQ<Key extends Comparable<Key>>`

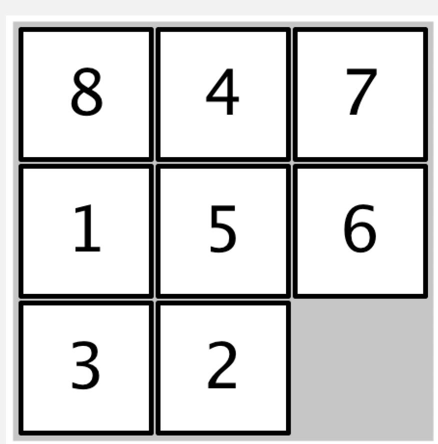
	`MaxPQ()`	*create an empty priority queue*
	`MaxPQ(Key[] a)`	*create a priority queue with given keys*
`void`	`insert(Key v)`	*insert a key into the priority queue*
`Key`	`delMax()`	*return and remove a largest key*
`boolean`	`isEmpty()`	*is the priority queue empty?*
`Key`	`max()`	*return a largest key*
`int`	`size()`	*number of entries in the priority queue*

Note. Duplicate keys allowed; `delMax()` picks any maximum key.

Priority queue: applications

- Event-driven simulation. [customers in a line, colliding particles]
- Numerical computation. [reducing roundoff error]
- Discrete optimization. [bin packing, scheduling]
- Artificial intelligence. [A* search]
- Computer networks. [web cache]
- Data compression. [Huffman codes]
- Operating systems. [load balancing, interrupt handling]
- Graph searching. [Dijkstra's algorithm, Prim's algorithm]
- Number theory. [sum of powers]
- Spam filtering. [Bayesian spam filter]
- Statistics. [online median in data stream]





Priority queue: elementary implementation

Exercise. In the worst case, what are the running times for INSERT and DELETE-MAX for a priority queue implemented with

- an unordered array?
- an ordered array?

operation	argument	return value	size	contents (unordered)							contents (ordered)						
insert	P		1	P							P						
insert	Q		2	P	Q						P	Q					
insert	E		3	P	Q	E					E	P	Q				
remove max		Q	2	P	E						E	P					
insert	X		3	P	E	X					E	P	X				
insert	A		4	P	E	X	A				A	E	P	X			
insert	M		5	P	E	X	A	M			A	E	M	P	X		
remove max		X	4	P	E	M	A				A	E	M	P			
insert	P		5	P	E	M	A	P			A	E	M	P	P		
insert	L		6	P	E	M	A	P	L		A	E	L	M	P	P	
insert	E		7	P	E	M	A	P	L	E	A	E	E	L	M	P	P
remove max		P	6	E	E	M	A	P	L		A	E	E	L	M	P	

Priority queues: quiz 1

In the worst case, what are the running times for INSERT and DELETE-MAX for a priority queue implemented with an ordered array?

ignore array resizing

A. 1 and n

B. 1 and $\log n$

C. $\log n$ and 1

D. n and 1

0	1	2	3	4	5	6	7	8	9	10
A	E	E	L	M	P	X				

Priority queue: implementations cost summary

Challenge. Implement all operations efficiently.

implementation	INSERT	DELETE-MAX	MAX
unordered array	1	n	n
ordered array	n	1	1
goal	$\boxed{\log n}$	$\boxed{\log n}$	$\log n$

order of growth of running time for priority queue with n items

what might this mean?

Solution. "Somewhat-ordered" array.

https://algs4.cs.princeton.edu

2.4 PRIORITY QUEUES

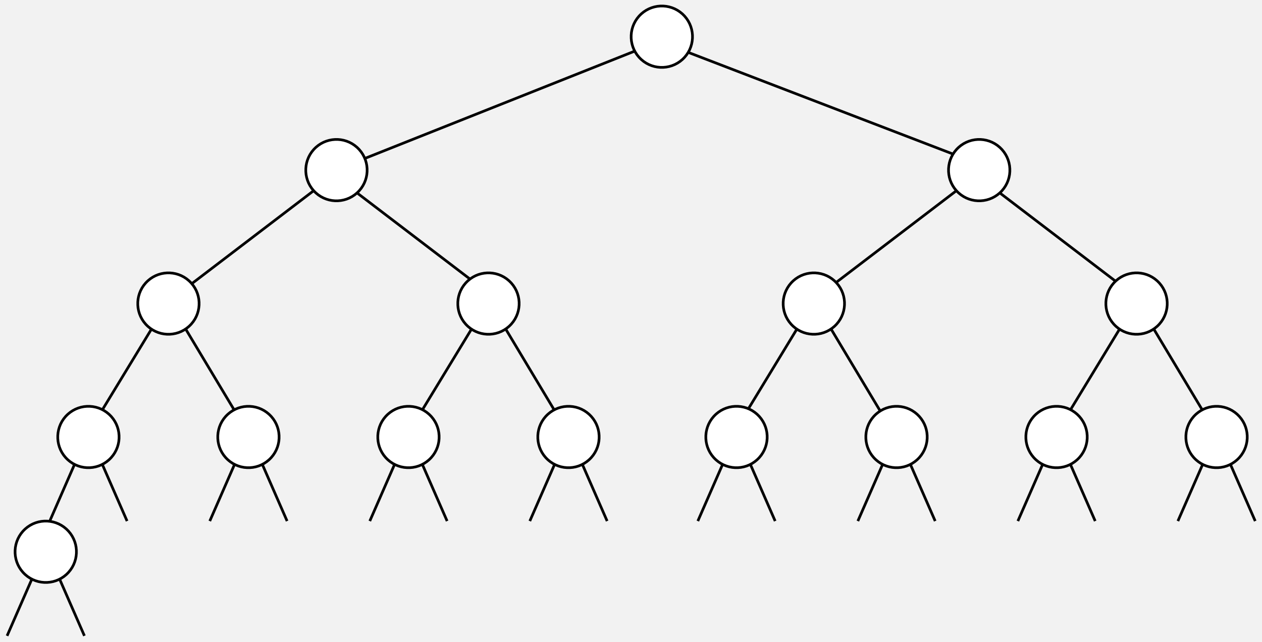
- API and elementary implementations
- binary heaps
- heapsort
- event-driven simulation

Complete binary tree

Binary tree. Empty or node with links to left and right binary trees.

Recursive definition

Complete tree. Every level (except possibly the last) is completely filled; the last level is filled from left to right.



complete binary tree with n = 16 nodes (height = 4)

Property. Height of complete binary tree with n nodes is $\lfloor \lg n \rfloor$.

A complete binary tree in nature

Binary heap: representation

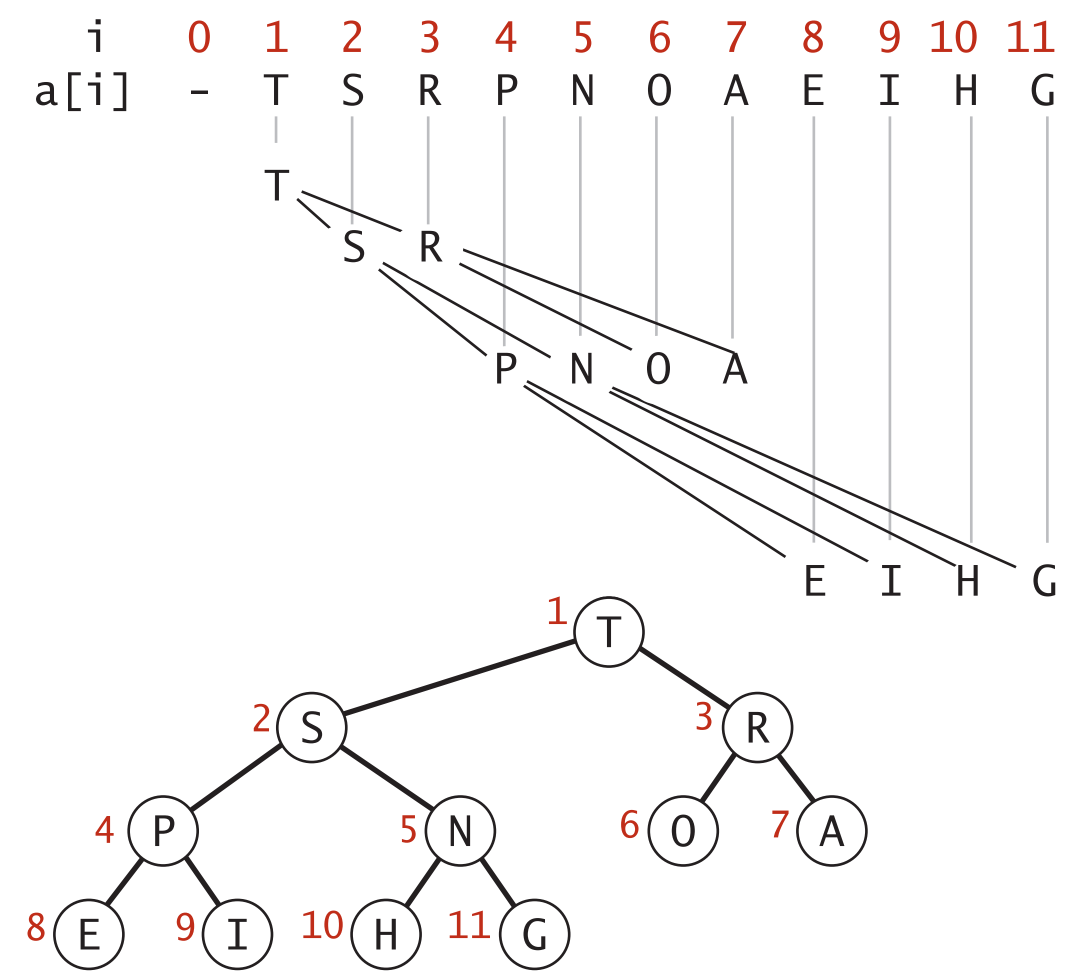
Binary heap. Array representation of a heap-ordered complete binary tree.

Heap-ordered binary tree.

- Keys in nodes.
- Parent's key no smaller than children's keys.

Array representation.

- Indices start at 1.
- Take nodes in level order.
- No explicit links needed!



Heap representations

Priority queues: quiz 2

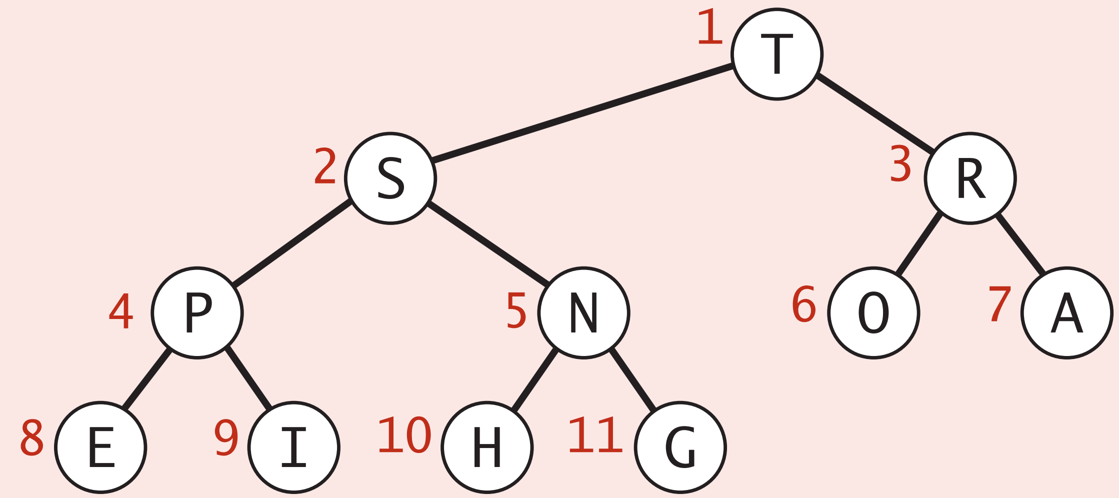
Which is the index of the parent of the item at index k in a binary heap?

A. `k/2 - 1`

B. `k/2`

C. `k/2 + 1`

D. `2*k`



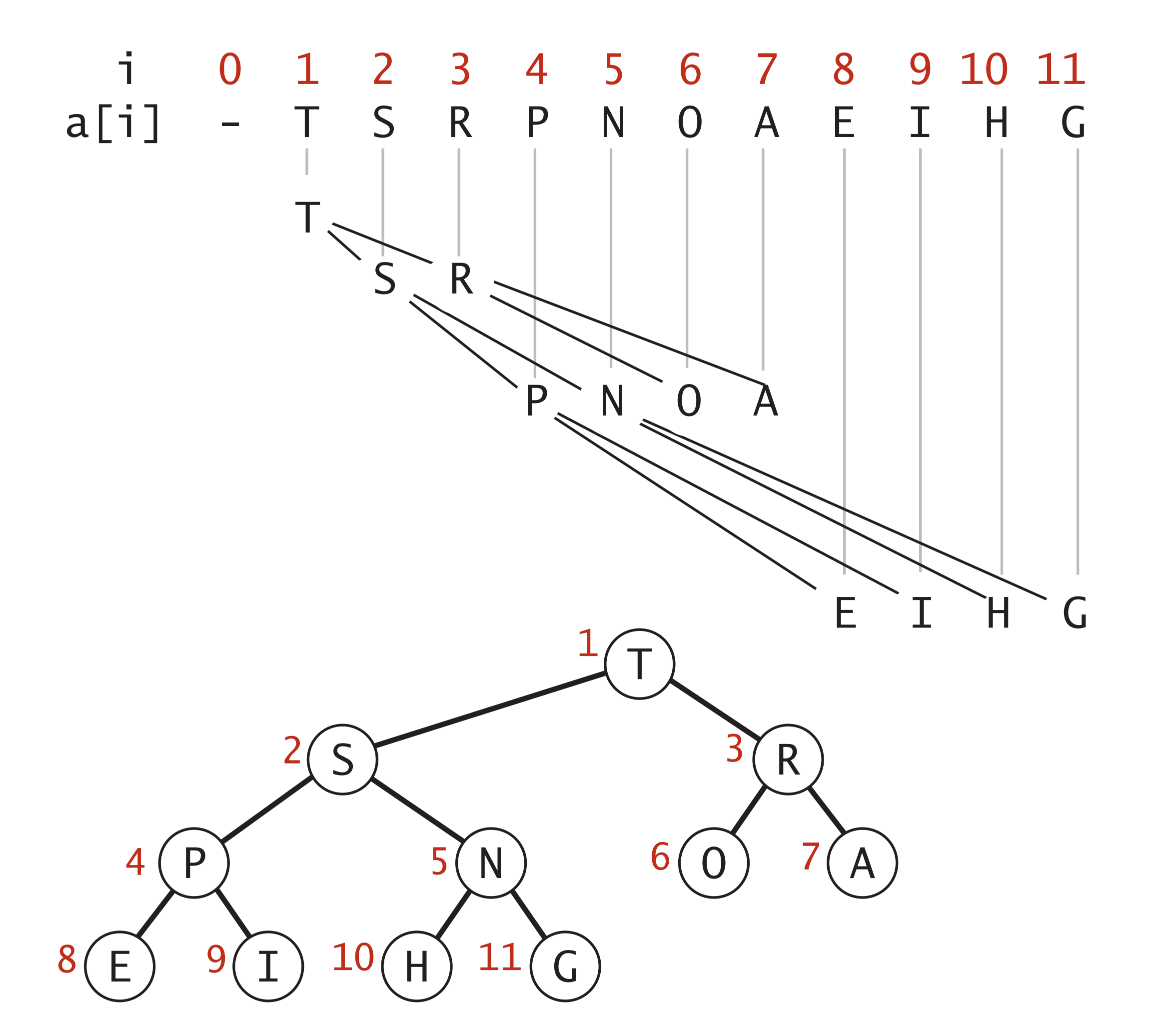
i	0	1	2	3	4	5	6	7	8	9	10	11
a[i]	-	T	S	R	P	N	O	A	E	I	H	G

Binary heap: properties

Proposition. Largest key is `a[1]`, which is root of binary tree.

Proposition. Can use array indices to move through tree.

- Parent of node at `k` is at `k/2`.
- Children of node at `k` are at `2k` and `2k+1`.



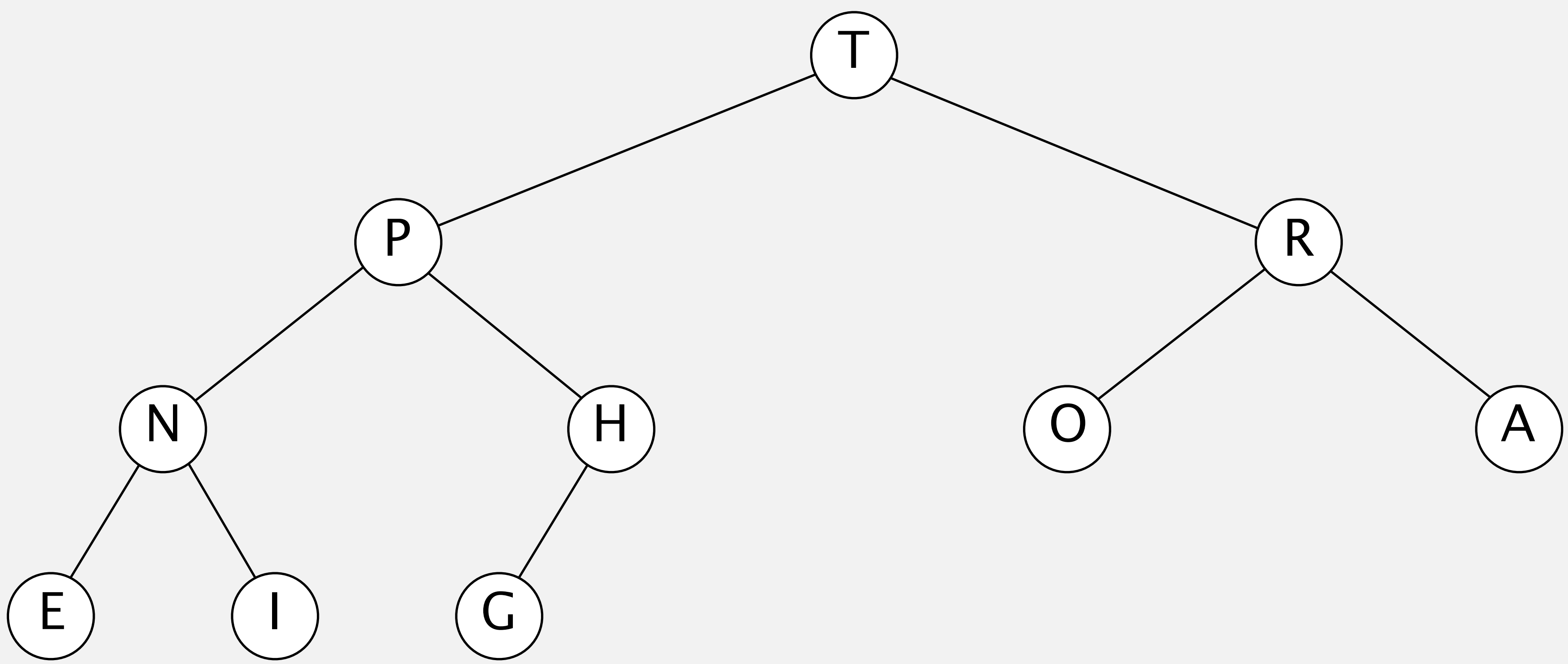
Heap representations

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

heap ordered



T	P	R	N	H	O	A	E	I	G	

Binary heap: swim / promotion

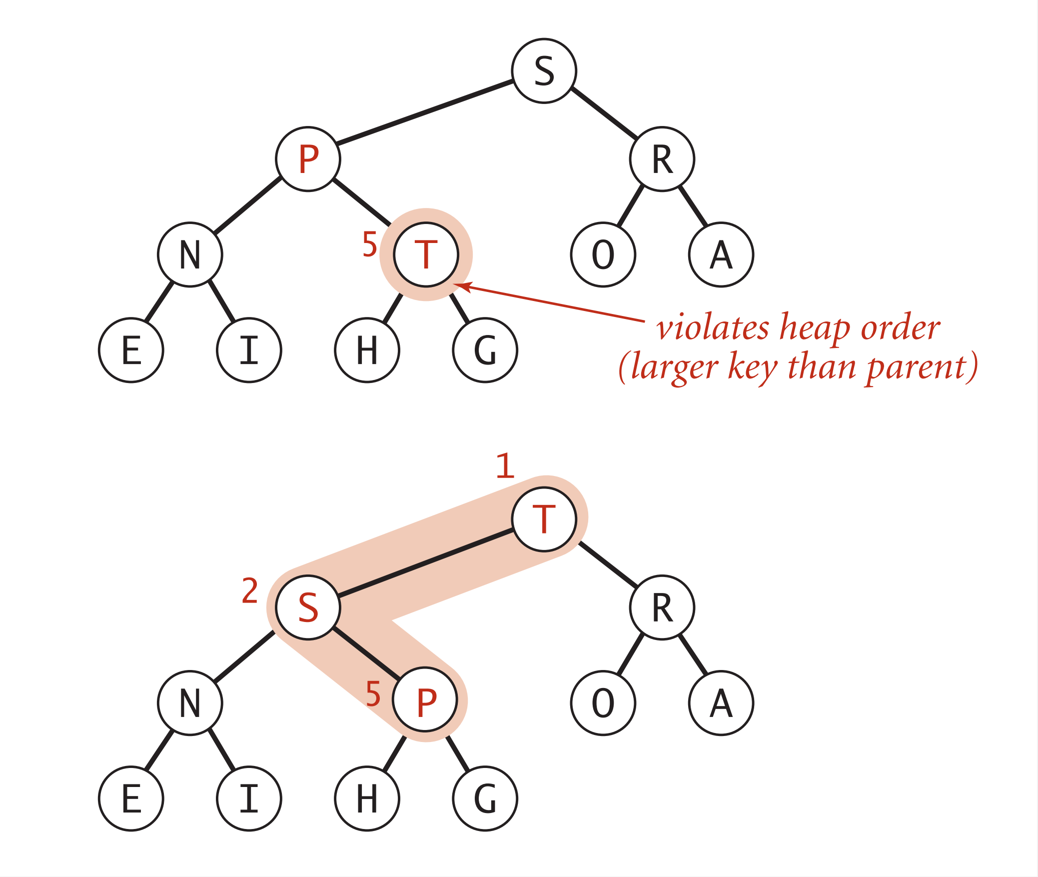
Scenario. A key becomes larger than its parent's key.

To eliminate the violation:

- Exchange key in child with key in parent.
- Repeat until heap order restored.

```
private void swim(int k)
{
   while (k > 1 && less(k/2, k))
   {
      exch(k, k/2);
      k = k/2;
   }
}
```

parent of node at k is at k/2



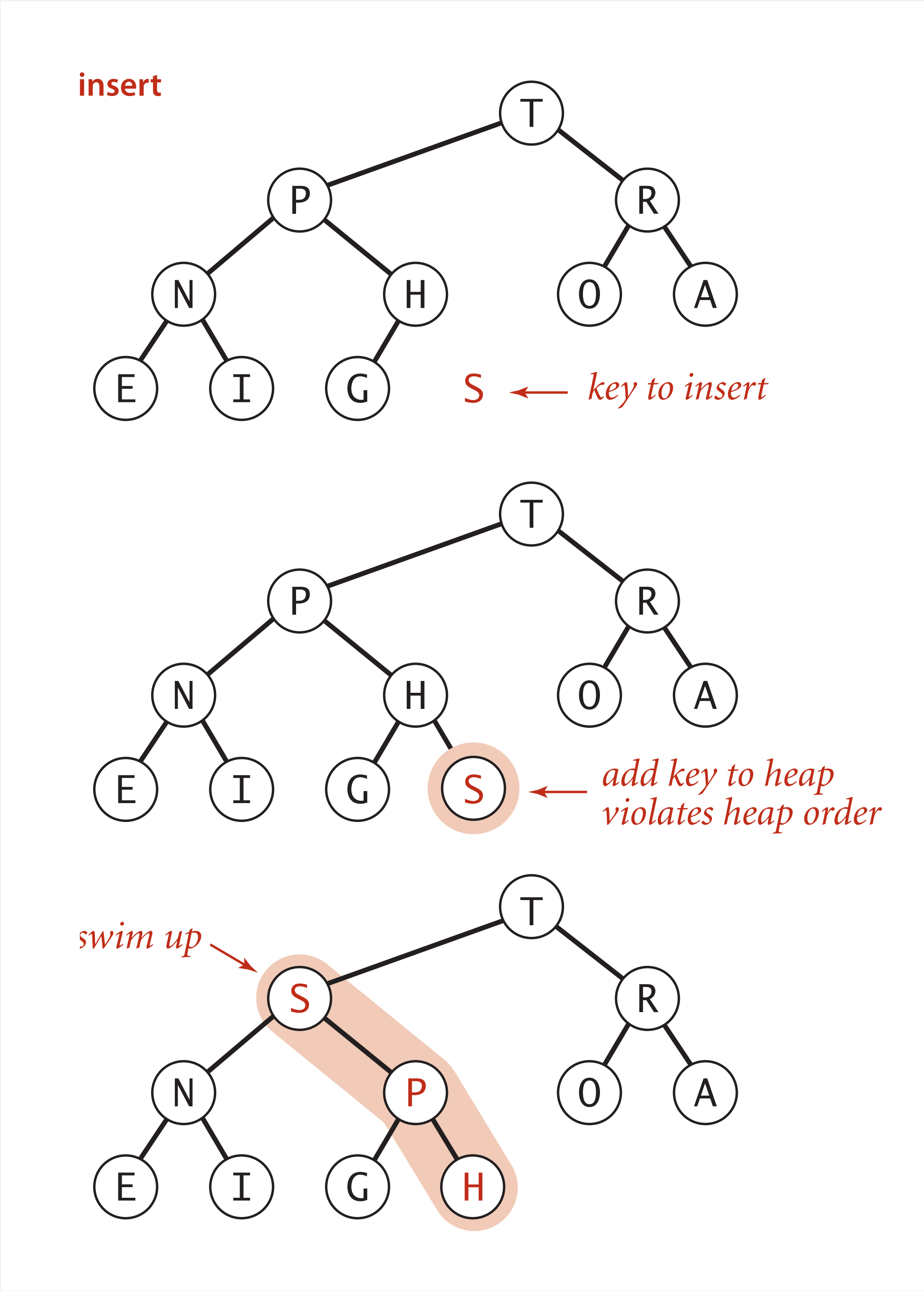
Peter principle. Node promoted to level of incompetence.

Binary heap: insertion

Insert. Add node at end in bottom level; then, swim it up.

Cost. At most $1 + \lg n$ compares.

```
public void insert(Key x)
{
   pq[++n] = x;
   swim(n);
}
```



Binary heap: sink / demotion

Scenario. A key becomes smaller than one (or both) of its children's.

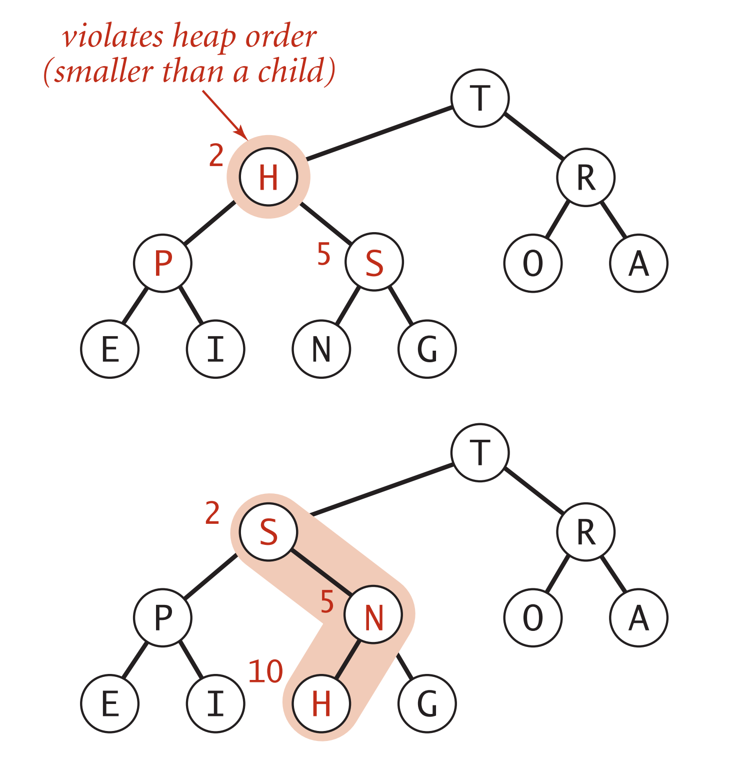
To eliminate the violation:

- Exchange key in parent with key in larger child.
- Repeat until heap order restored.

why not smaller child?

```
private void sink(int k)
{
   while (2*k <= n)
   {
      int j = 2*k;
      if (j < n && less(j, j+1)) j++;
      if (!less(k, j)) break;
      exch(k, j);
      k = j;
   }
}
```

children of node at k are at 2*k and 2*k+1



Top-down reheapify (sink)

Power struggle. Better subordinate promoted.

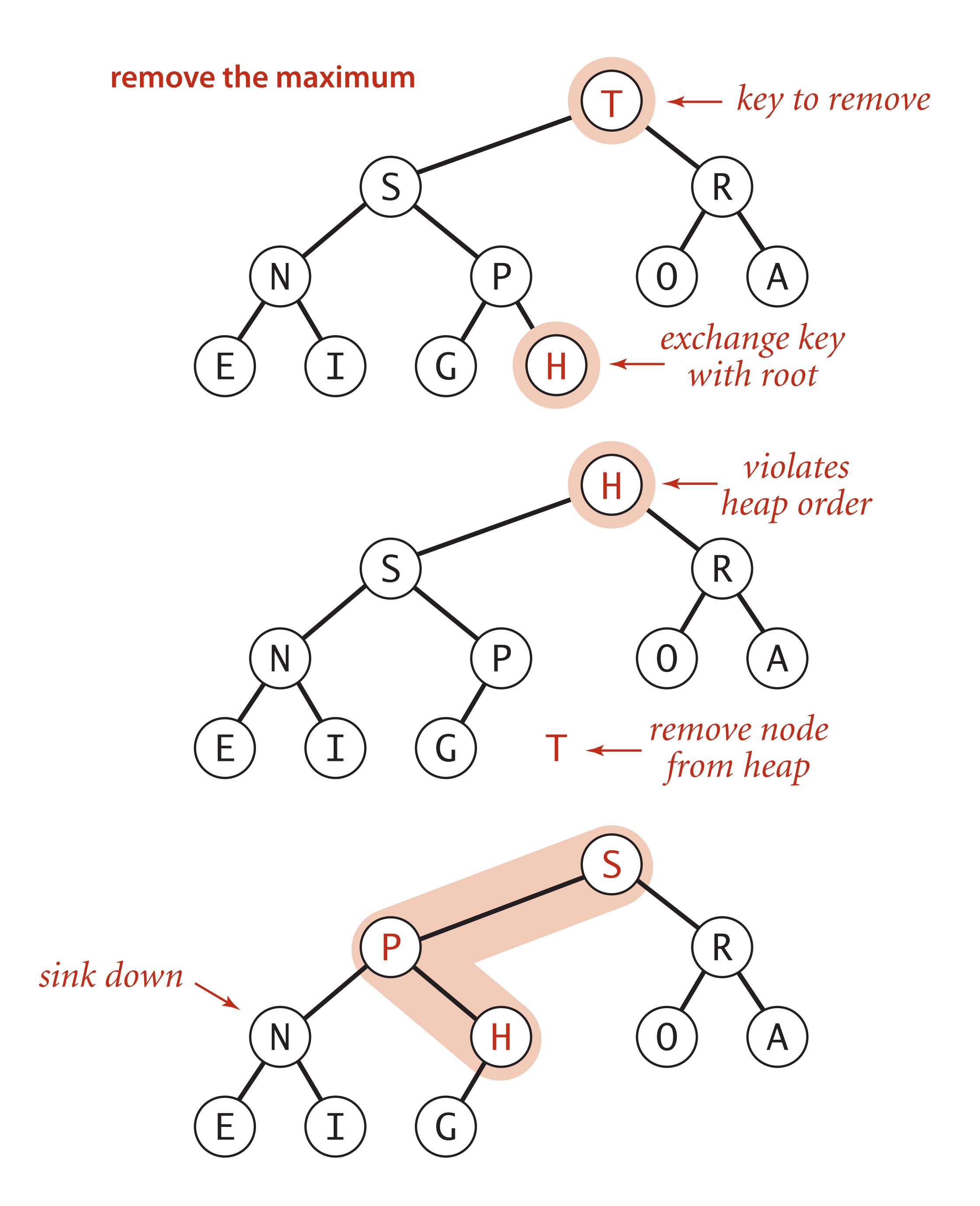
Binary heap: delete the maximum

Delete max. Exchange root with node at end; then, sink it down.

Cost. At most $2 \lg n$ compares.

```
public Key delMax()
{
   Key max = pq[1];
   exch(1, n--);
   sink(1);
   pq[n+1] = null;
   return max;
}
```

prevent loitering



Binary heap: Java implementation

```
public class MaxPQ<Key extends Comparable<Key>>
{
   private Key[] pq;
   private int n;

   public MaxPQ(int capacity)
   {  pq = (Key[]) new Comparable[capacity+1];  }

   public boolean isEmpty()
   {  return n == 0;  }
   public void insert(Key key)   // see previous code
   public Key delMax()           // see previous code

   private void swim(int k)      // see previous code
   private void sink(int k)      // see previous code

   private boolean less(int i, int j)
   {  return pq[i].compareTo(pq[j]) < 0;  }

   private void exch(int i, int j)
   {  Key t = pq[i]; pq[i] = pq[j]; pq[j] = t;  }
}
```

https://algs4.cs.princeton.edu/24pq/MaxPQ.java.html

fixed capacity (for simplicity)

PQ ops

heap helper functions

array helper functions

Priority queue: implementations cost summary

implementation	INSERT	DELETE-MAX	MAX
unordered array	1	n	n
ordered array	n	1	1
binary heap	$\log n$	$\log n$	1

order of growth of running time for priority queue with n items

Binary heap: considerations

Underflow and overflow.

- Underflow: throw exception if deleting from empty PQ.
- Overflow: add no-arg constructor and use resizing array.

leads to $\log n$
amortized time per op
(how to make worst case?)

Minimum-oriented priority queue.

- Replace `less()` with `greater()`.
- Implement `greater()`.

Other operations.

- Remove an arbitrary item.
- Change the priority of an item.

can implement efficiently with `sink()` and `swim()`
[stay tuned for Prim/Dijkstra]

Immutability of keys.

- Assumption: client does not change keys while they're on the PQ.
- Best practice: use immutable keys.

Immutability: implementing in Java

Data type. Set of values and operations on those values.

Immutable data type. Can't change the data type value once created.

```
public final class Vector {
   private final int n;
   private final double[] data;

   public Vector(double[] data) {
      this.n = data.length;
      this.data = new double[n];
      for (int i = 0; i < n; i++)
         this.data[i] = data[i];
   }

   ⋮

}
```

instance variables `private` and `final` (neither necessary nor sufficient, but good programming practice)

defensive copy of mutable instance variables

instance methods don't change instance variables

Immutable in Java. `String`, `Integer`, `Double`, `Color`, `File`, …

Mutable in Java. `StringBuilder`, `Stack`, `URL`, arrays, …

Immutability: properties

Data type. Set of values and operations on those values.

Immutable data type. Can't change the data type value once created.

Advantages.

- Simplifies debugging.
- Simplifies concurrent programming.
- More secure in presence of hostile code.
- Safe to use as key in priority queue or symbol table.

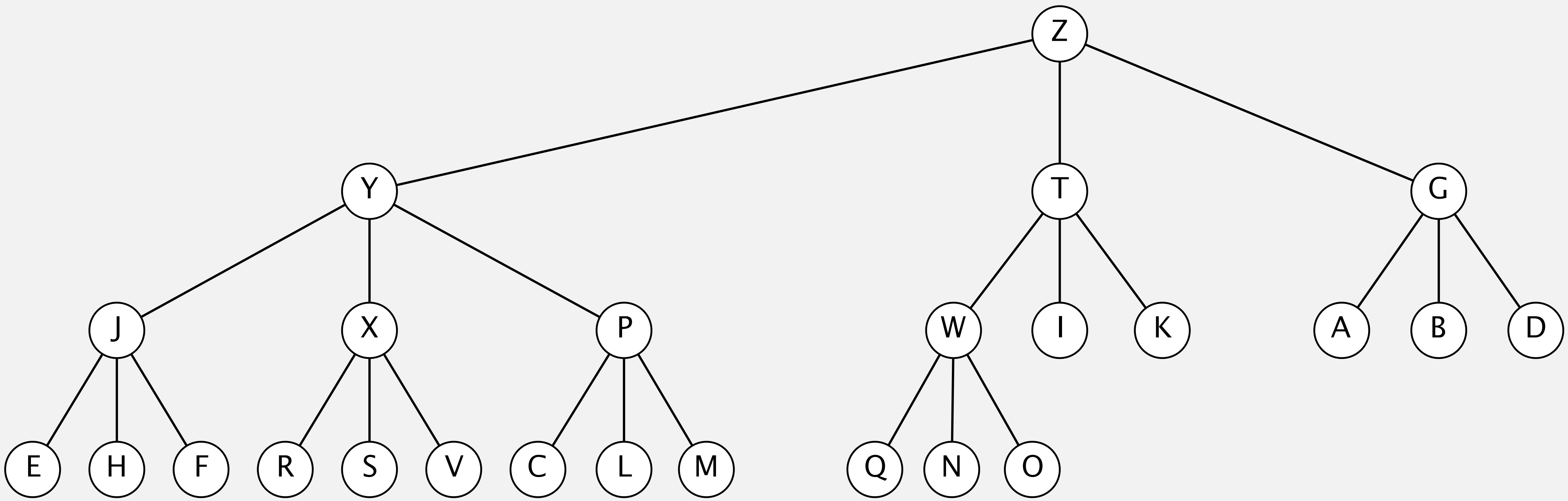
Disadvantage. Must create new object for each data-type value.

Binary heap: practical improvement

Multiway heaps.

- Complete d-way tree.
- Parent's key no smaller than its children's keys.

Fact. Height of complete d-way tree on n nodes is $\sim \log_d n$.



3-way heap

Priority queues: quiz 3

In the worst case, how many compares to INSERT and DELETE-MAX in a d-way heap?

A. $\sim \log_d n$ and $\sim \log_d n$

B. $\sim \log_d n$ and $\sim d \log_d n$

C. $\sim d \log_d n$ and $\sim \log_d n$

D. $\sim d \log_d n$ and $\sim d \log_d n$

Priority queue: implementation cost summary

implementation	INSERT	DELETE-MAX	MAX
unordered array	1	n	n
ordered array	n	1	1
binary heap	$\log n$	$\log n$	1
d-ary heap	$\log_d n$	$d \log_d n$	1
Fibonacci	1	$\log n$ †	1
Brodal queue	1	$\log n$	1
impossible	1	1	1

← sweet spot: $d = 4$

← why impossible?

† amortized

order-of-growth of running time for priority queue with n items

Impossibility of priority queue with constant-time INSERT & DELETE-MAX

Exercise.

- Assume there is a priority queue which makes a constant number of compares in the worst case for both INSERT and DELETE-MAX.
- Design a sorting algorithm that uses this priority queue.
- How many compares does it perform in the worst case?

https://algs4.cs.princeton.edu

2.4 Priority Queues

- API and elementary implementations
- binary heaps
- ***heapsort***
- event-driven simulation

Priority queues: quiz 4

What are the properties of this sorting algorithm?

```
public void sort(String[] a)
{
    int n = a.length;
    MaxPQ<String> pq = new MaxPQ<String>();
    for (int i = 0; i < n; i++)
        pq.insert(a[i]);
    for (int i = n-1; i >= 0; i--)
        a[i] = pq.delMax();
}
```

A. $n \log n$ compares in the worst case.

B. In-place.

C. Stable.

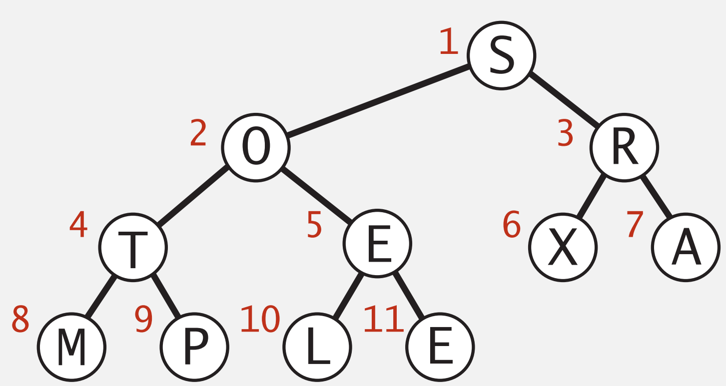
D. *All of the above.*

Heapsort

Basic plan for in-place sort.

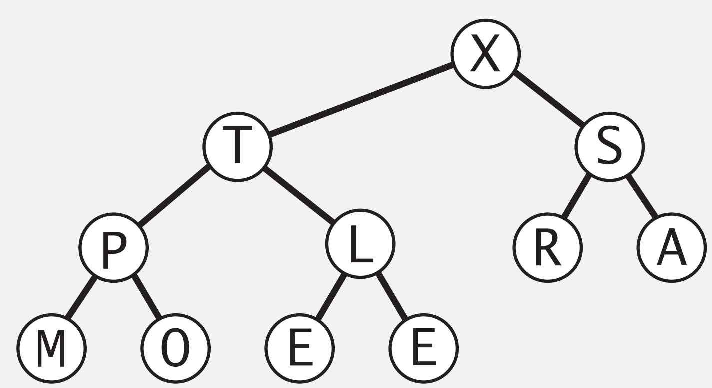
- View input array as a complete binary tree.
- Heap construction: build a max-heap with all n keys.
- Sortdown: repeatedly remove the maximum key.

keys in arbitrary order



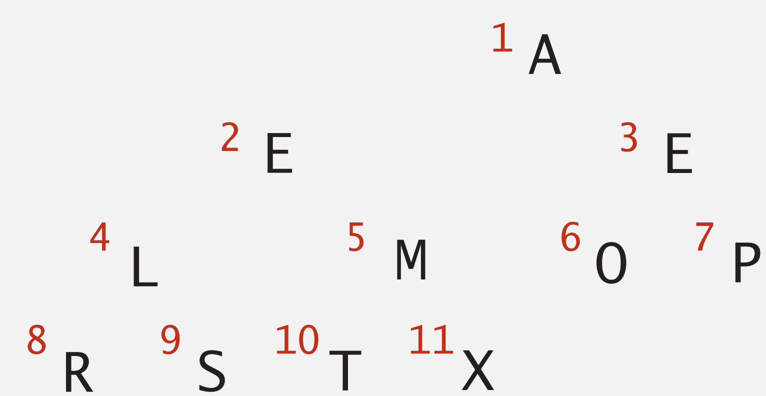
1	2	3	4	5	6	7	8	9	10	11
S	O	R	T	E	X	A	M	P	L	E

build max heap (in place)



1	2	3	4	5	6	7	8	9	10	11
X	T	S	P	L	R	A	M	O	E	E

sorted result (in place)



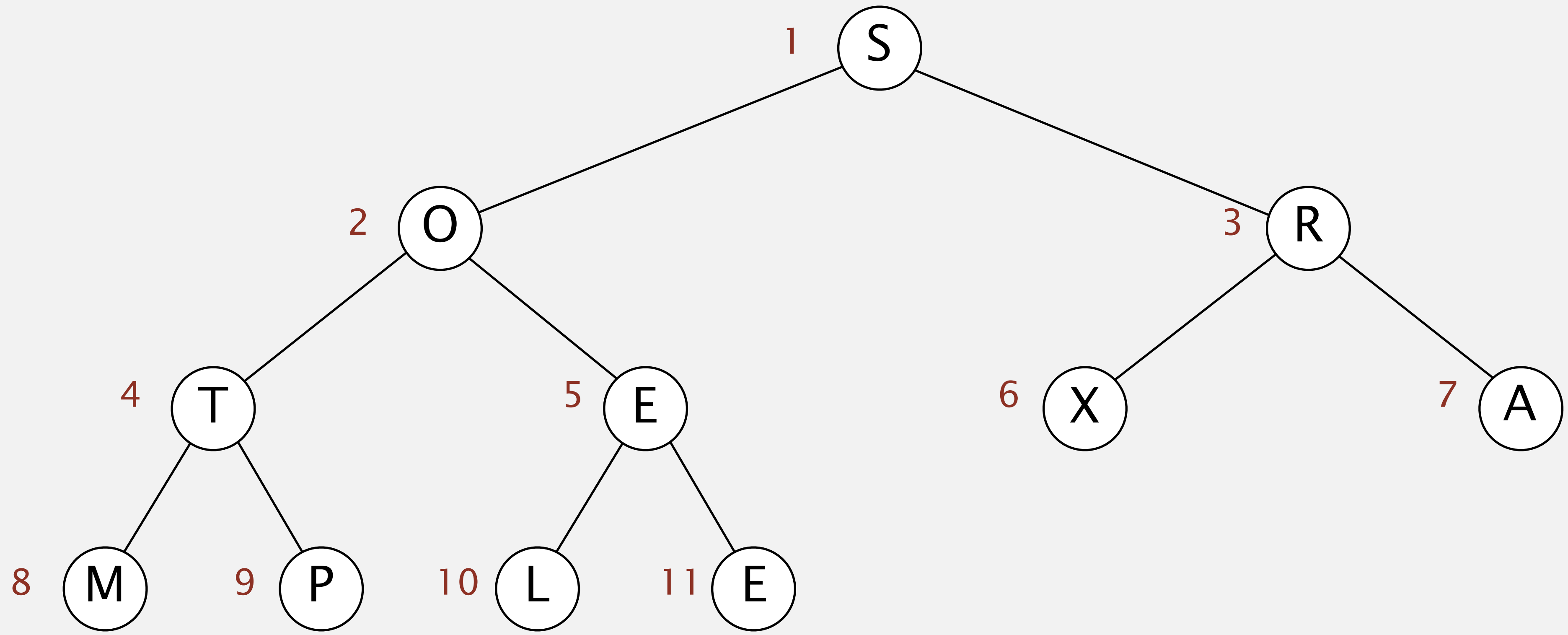
1	2	3	4	5	6	7	8	9	10	11
A	E	E	L	M	O	P	R	S	T	X

Heapsort demo

Heap construction. Build max heap using bottom-up method.

for now, assume array entries are indexed 1 to n

array in arbitrary order



S	O	R	T	E	X	A	M	P	L	E
1	2	3	4	5	6	7	8	9	10	11

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

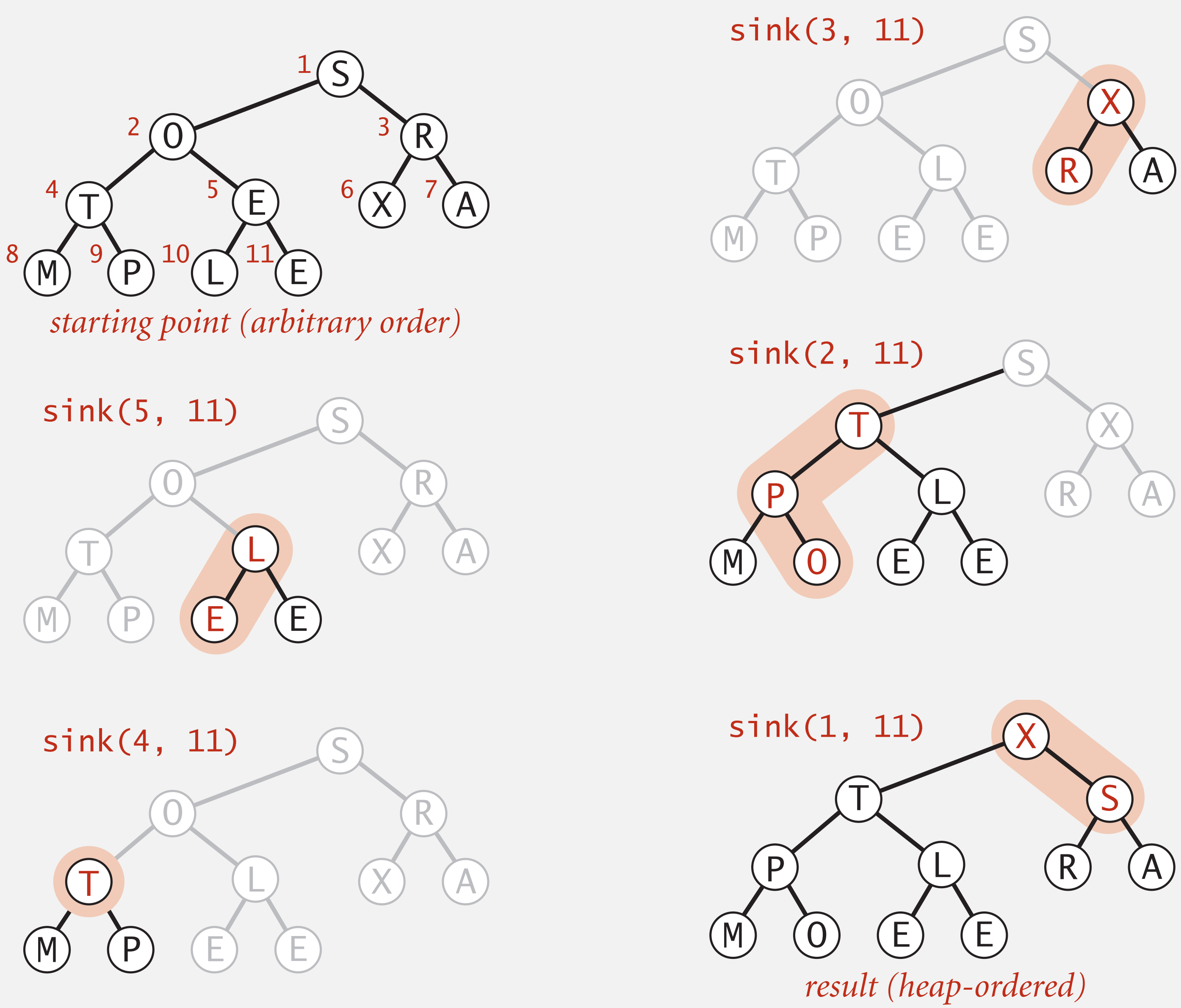
array in sorted order

A	E	E	L	M	O	P	R	S	T	X
1	2	3	4	5	6	7	8	9	10	11

Heapsort: heap construction

First pass. Build heap using bottom-up method.

```
for (int k = n/2; k >= 1; k--)
   sink(a, k, n);
```



Key insight.
After `sink(a, k, n)` completes, the subtree rooted at k is a heap.

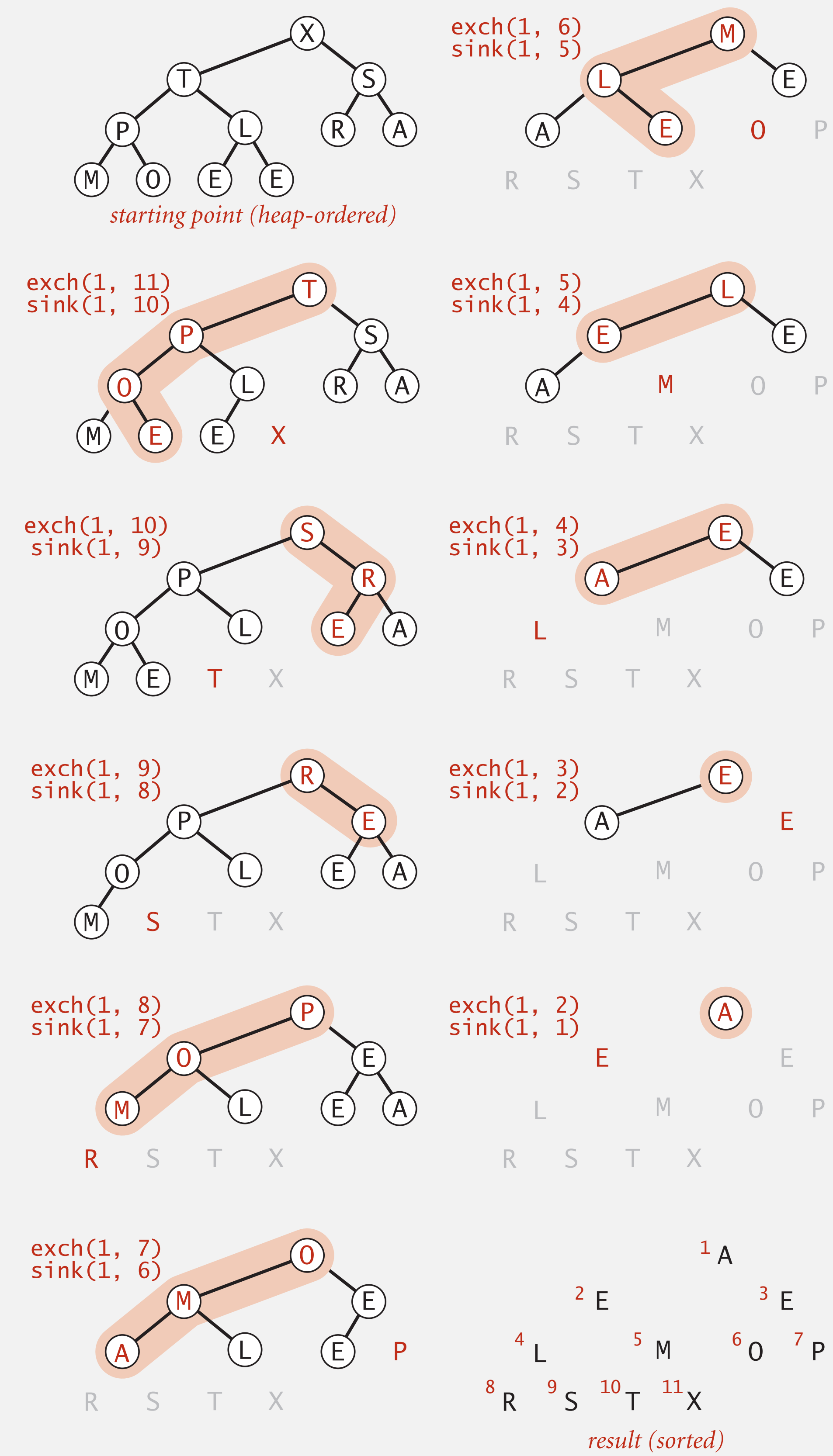
Heapsort: sortdown

Second pass.

- Remove the maximum, one at a time.
- Leave in array, instead of nulling out.

```
while (n > 1)
{
   exch(a, 1, n--);
   sink(a, 1, n);
}
```

Key insight.
After each iteration, the array consists of a heap-ordered subarray followed by a sub-array in final order



Heapsort: Java implementation

```
public class Heap
{
   public static void sort(Comparable[] a)
   {
      int n = a.length;
      for (int k = n/2; k >= 1; k--)
         sink(a, k, n);
      while (n > 1)
      {
         exch(a, 1, n);
         sink(a, 1, --n);
      }
   }

   private static void sink(Comparable[] a, int k, int n)
   {  /* as before */  }

   private static boolean less(Comparable[] a, int i, int j)
   {  /* as before */  }

   private static void exch(Object[] a, int i, int j)
   {  /* as before */  }

}
```

but make static (and pass arguments)

but convert from 1-based indexing to 0-base indexing

https://algs4.cs.princeton.edu/24pq/Heap.java.html

Heapsort: trace

N	k	a[i] 0	1	2	3	4	5	6	7	8	9	10	11
initial values			S	O	R	T	E	X	A	M	P	L	E
11	5		S	O	R	T	L	X	A	M	P	E	E
11	4		S	O	R	T	L	X	A	M	P	E	E
11	3		S	O	X	T	L	R	A	M	P	E	E
11	2		S	T	X	P	L	R	A	M	O	E	E
11	1		X	T	S	P	L	R	A	M	O	E	E
heap-ordered			X	T	S	P	L	R	A	M	O	E	E
10	1		T	P	S	O	L	R	A	M	E	E	X
9	1		S	P	R	O	L	E	A	M	E	T	X
8	1		R	P	E	O	L	E	A	M	S	T	X
7	1		P	O	E	M	L	E	A	R	S	T	X
6	1		O	M	E	A	L	E	P	R	S	T	X
5	1		M	L	E	A	E	O	P	R	S	T	X
4	1		L	E	E	A	M	O	P	R	S	T	X
3	1		E	A	E	L	M	O	P	R	S	T	X
2	1		E	A	E	L	M	O	P	R	S	T	X
1	1		A	E	E	L	M	O	P	R	S	T	X
sorted result			A	E	E	L	M	O	P	R	S	T	X

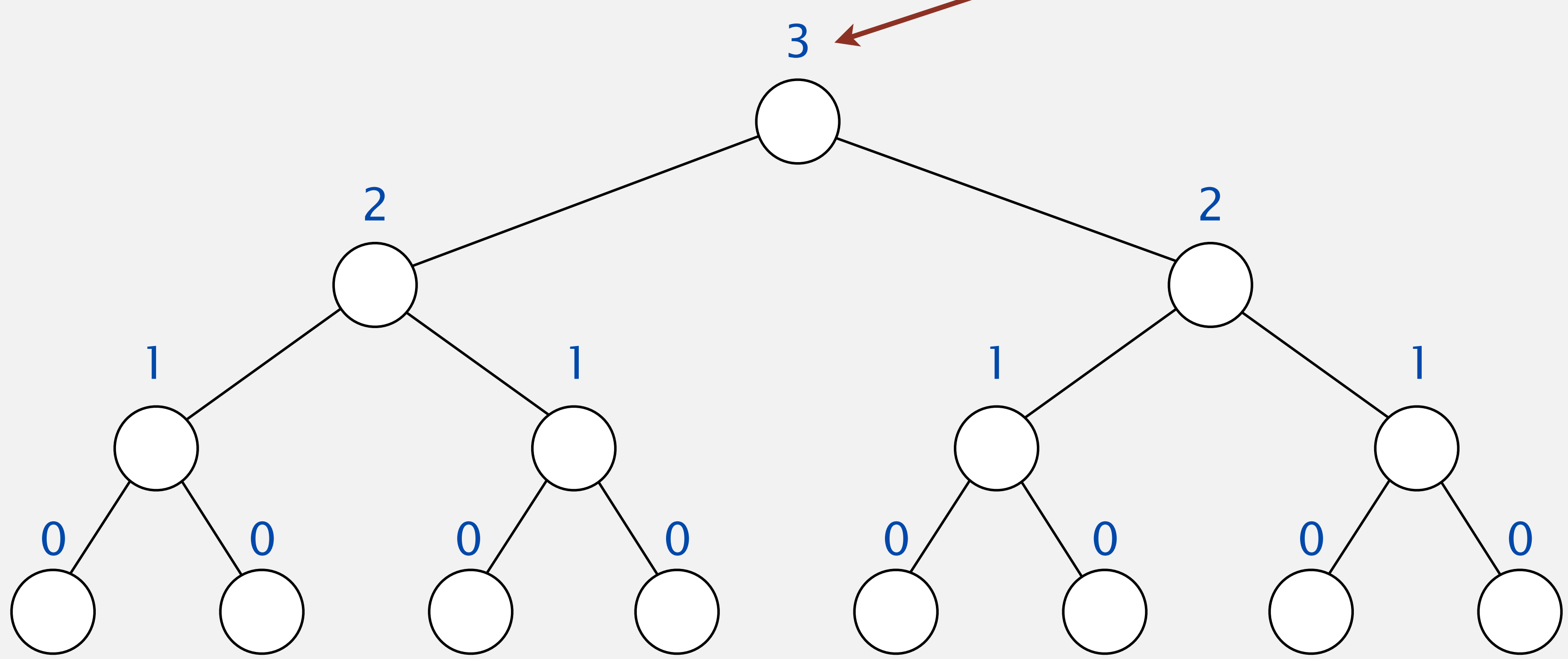
Heapsort trace (array contents just after each sink)

Heapsort: mathematical analysis

Proposition. Heap construction makes $\leq n$ exchanges and $\leq 2\,n$ compares.

Pf sketch. [assume $n = 2^{h+1} - 1$]

max number of exchanges to sink node



binary heap of height h = 3

a tricky sum (see COS 340)

$$\begin{aligned} h + 2(h-1) + 4(h-2) + 8(h-3) + \ldots + 2^h(0) &= 2^{h+1} - h - 2 \\ &= n - (h-1) \\ &\leq n \end{aligned}$$

Heapsort: mathematical analysis

Proposition. Heap construction makes $\leq n$ exchanges and $\leq 2\,n$ compares.

Proposition. Heapsort uses $\leq 2\,n \lg n$ compares and exchanges.

algorithm can be improved to $\sim n \lg n$
(but no such variant is known to be practical)

Significance. In-place sorting algorithm with $n \log n$ worst-case.

- Mergesort: no, linear extra space. ← in-place merge possible, not practical
- Quicksort: no, quadratic time in worst case. ← $n \log n$ worst-case quicksort possible, not practical
- Heapsort: yes!

Bottom line. Heapsort is optimal for both time and space, but:

- Inner loop longer than quicksort's.
- Makes poor use of cache.
- Not stable.

can be improved using
advanced caching tricks

Sorting algorithms: summary

	inplace?	stable?	best	average	worst	remarks
selection	✔		$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	n exchanges
insertion	✔	✔	n	$\frac{1}{4} n^2$	$\frac{1}{2} n^2$	use for small n or partially ordered
merge		✔	$\frac{1}{2} n \lg n$	$n \lg n$	$n \lg n$	$n \log n$ guarantee; stable
quick	✔		$n \lg n$	$2 n \ln n$	$\frac{1}{2} n^2$	$n \log n$ probabilistic guarantee; fastest in practice
3-way quick	✔		n	$2 n \ln n$	$\frac{1}{2} n^2$	improves quicksort when duplicate keys
heap	✔		$3 n$	$2 n \lg n$	$2 n \lg n$	$n \log n$ guarantee; in-place
?	✔	✔	n	$n \lg n$	$n \lg n$	holy sorting grail