INTRODUCTION TO ALGORITHMS

Lecture 7: Heap Sort Algorithm

Yao-Chung Fan
yfan@nchu.edu.tw

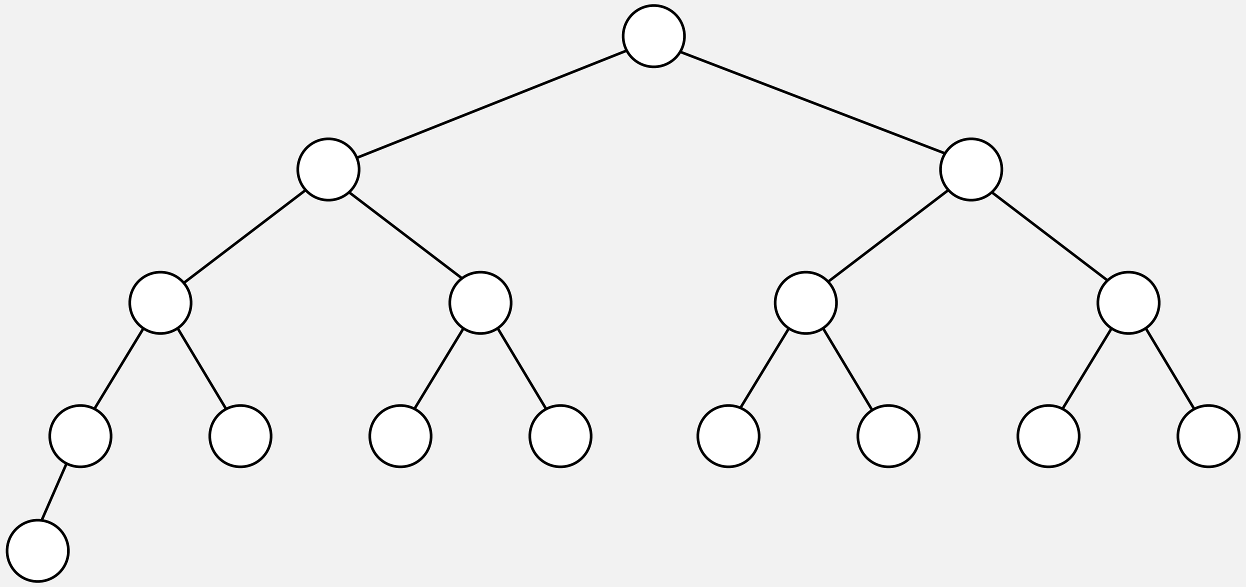
HEAPSORT

- *binary heap*
- *In-place heapsort*

Complete binary tree

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

Complete tree. Perfectly balanced, except for bottom level.



complete tree with N = 16 nodes (height = 4)

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

Pf. Height increases only when N is a power of 2.

Binary heap representations

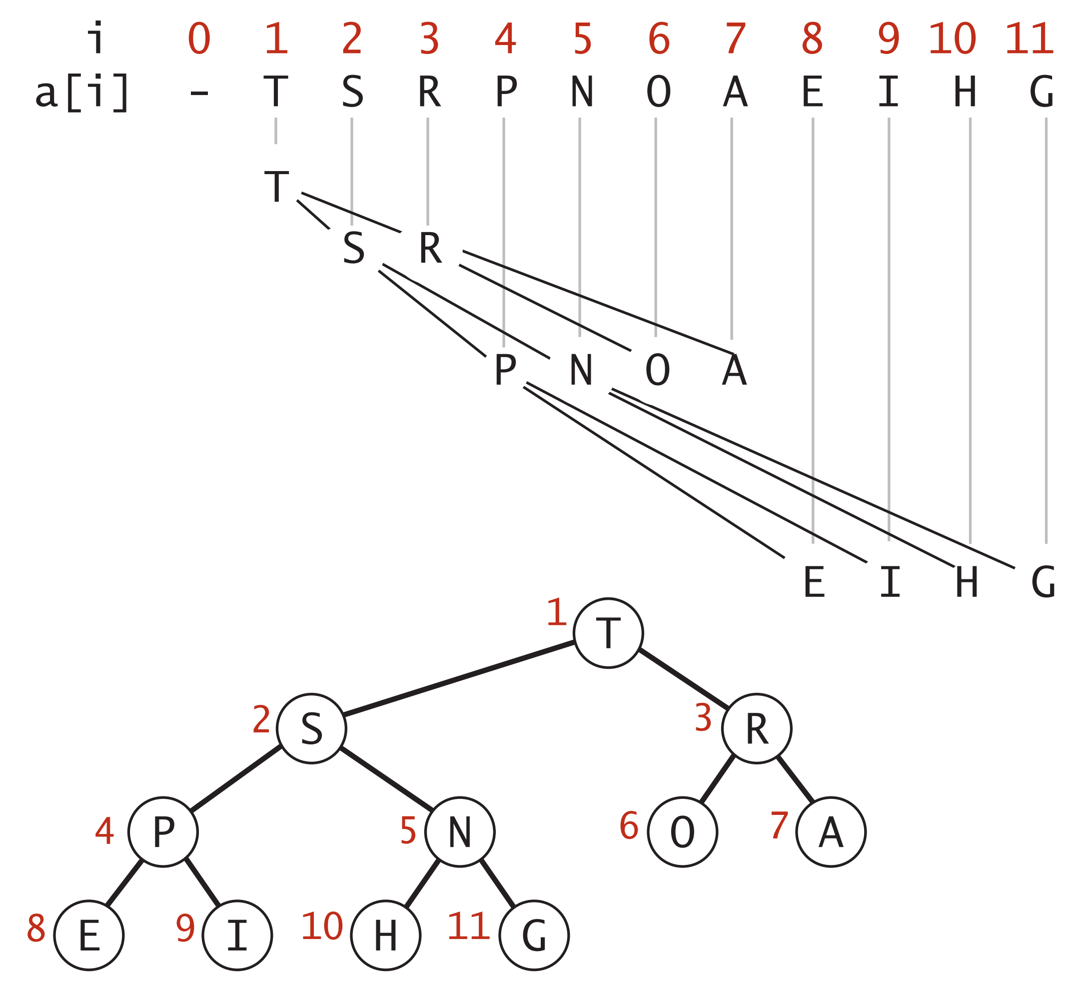
Binary heap. Array representation of a heap-ordered complete binary tree.

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!



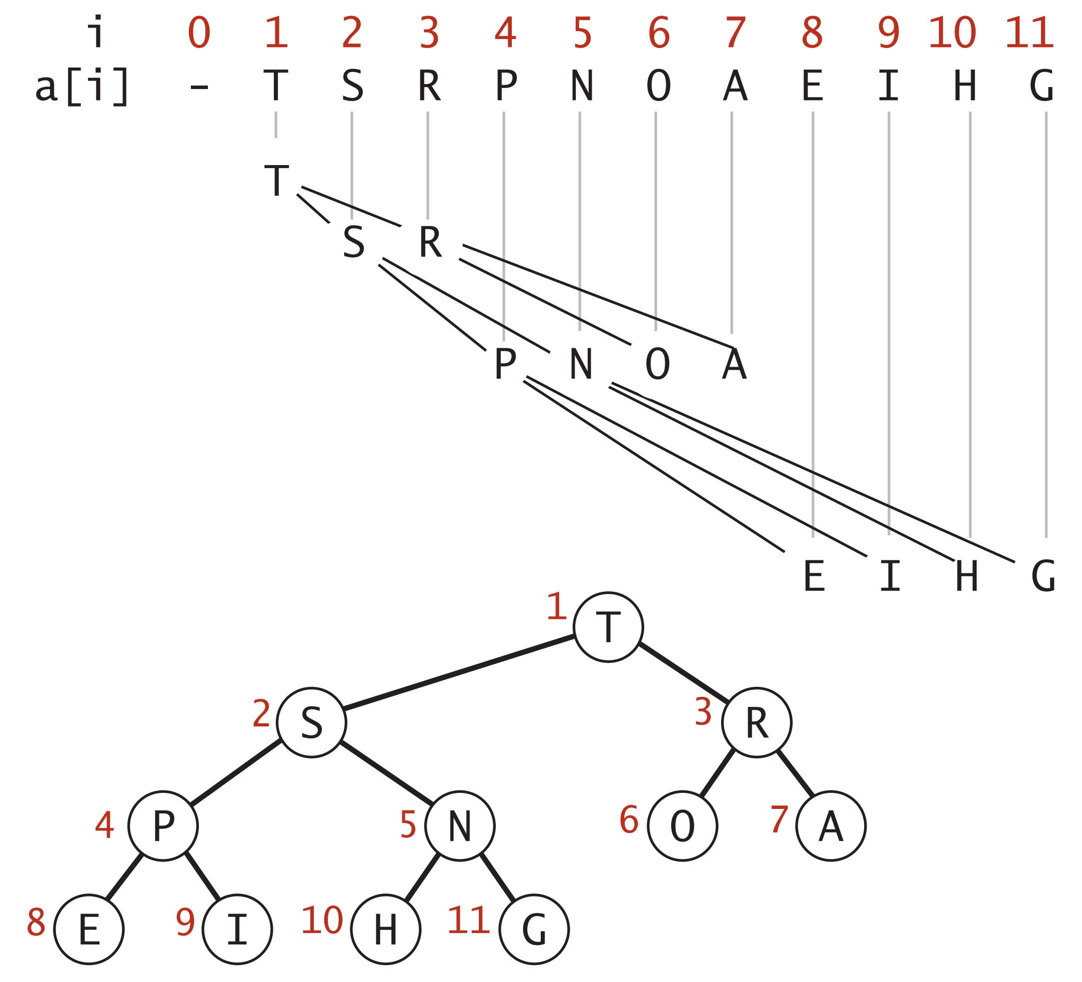
Heap representations

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`.



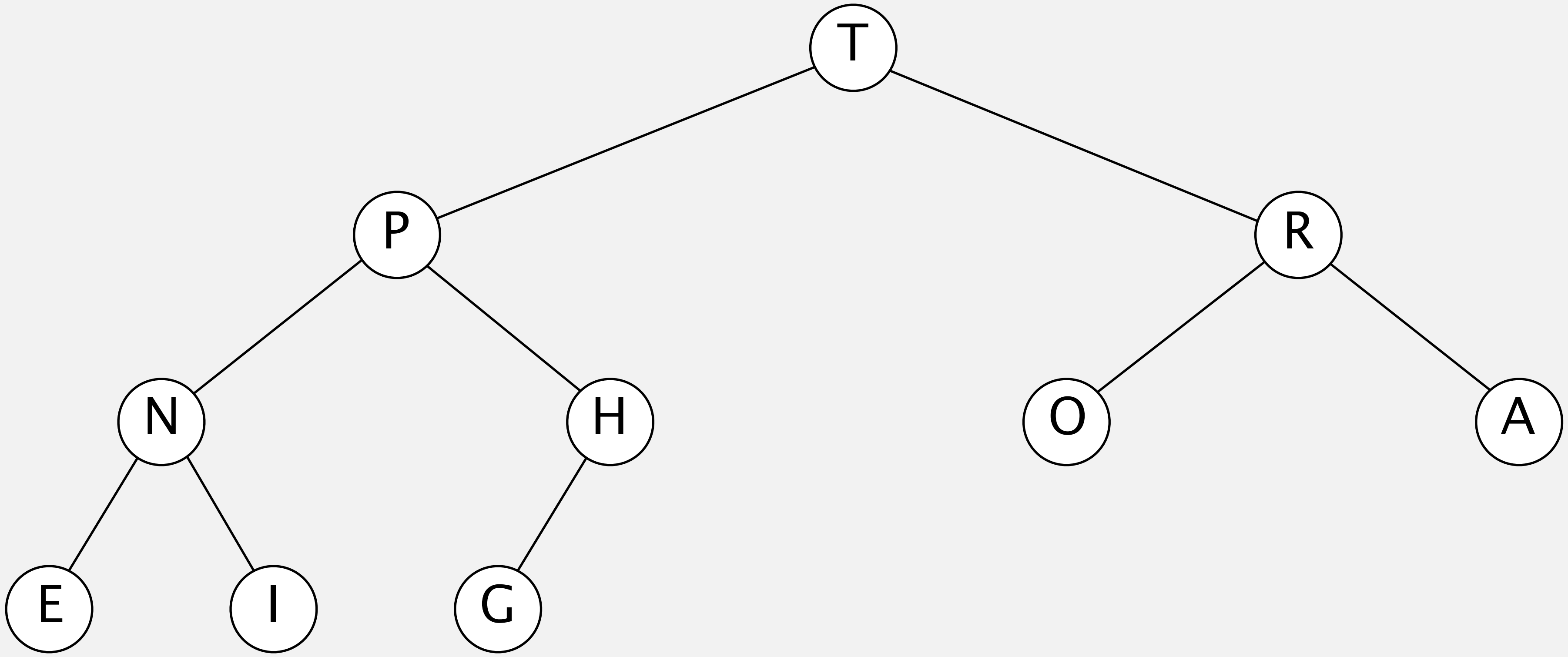
Heap representations

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	

Promotion in a heap

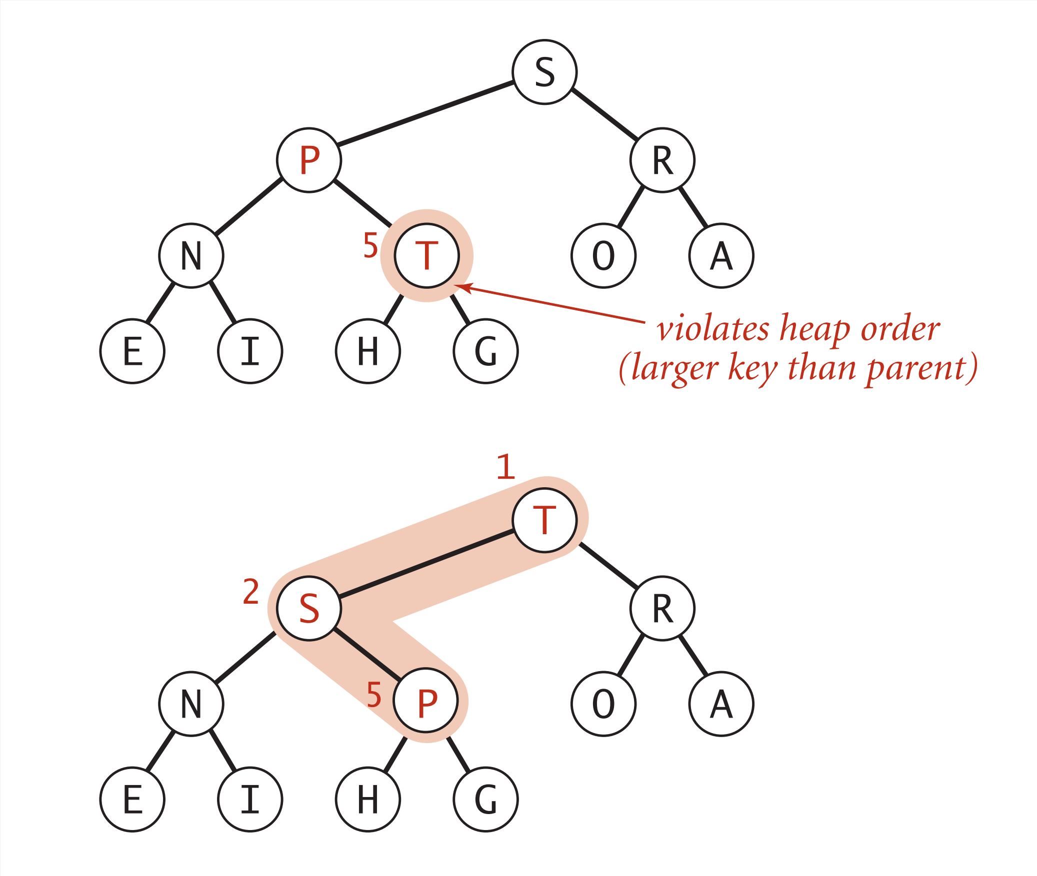
Scenario. Child's key becomes larger key than its parent's key.

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



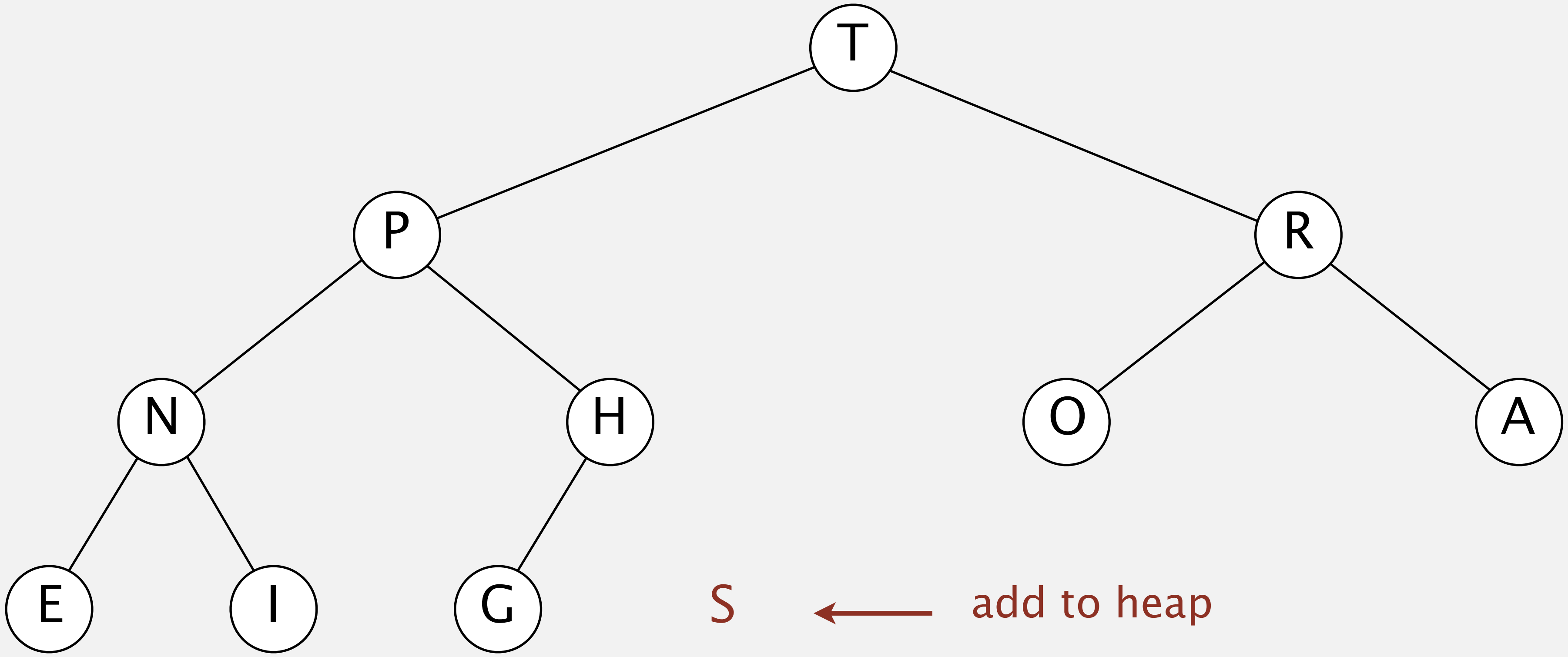
Peter principle. Node promoted to level of incompetence.

Binary heap demo (insert example)

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

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

insert S



add to heap

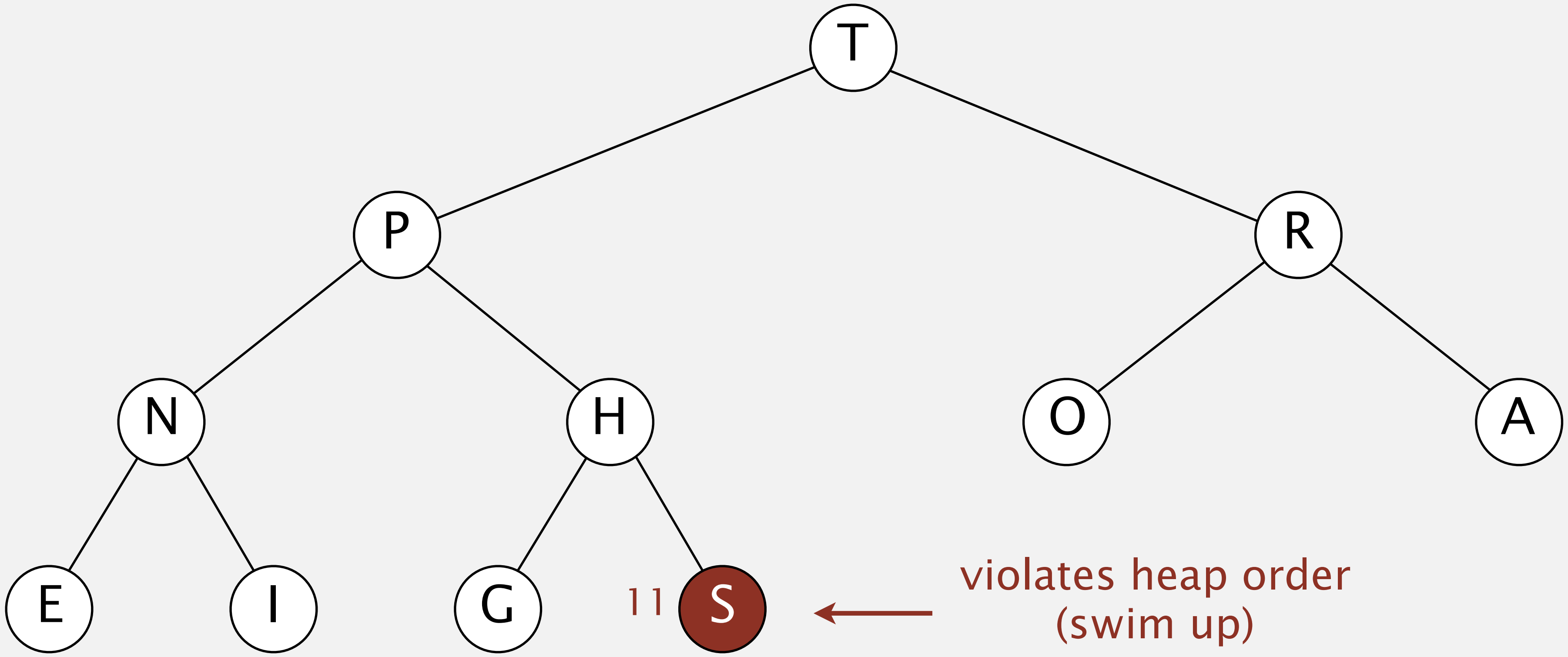
T	P	R	N	H	O	A	E	I	G	

Binary heap demo (insert example)

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

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

insert S



violates heap order
(swim up)

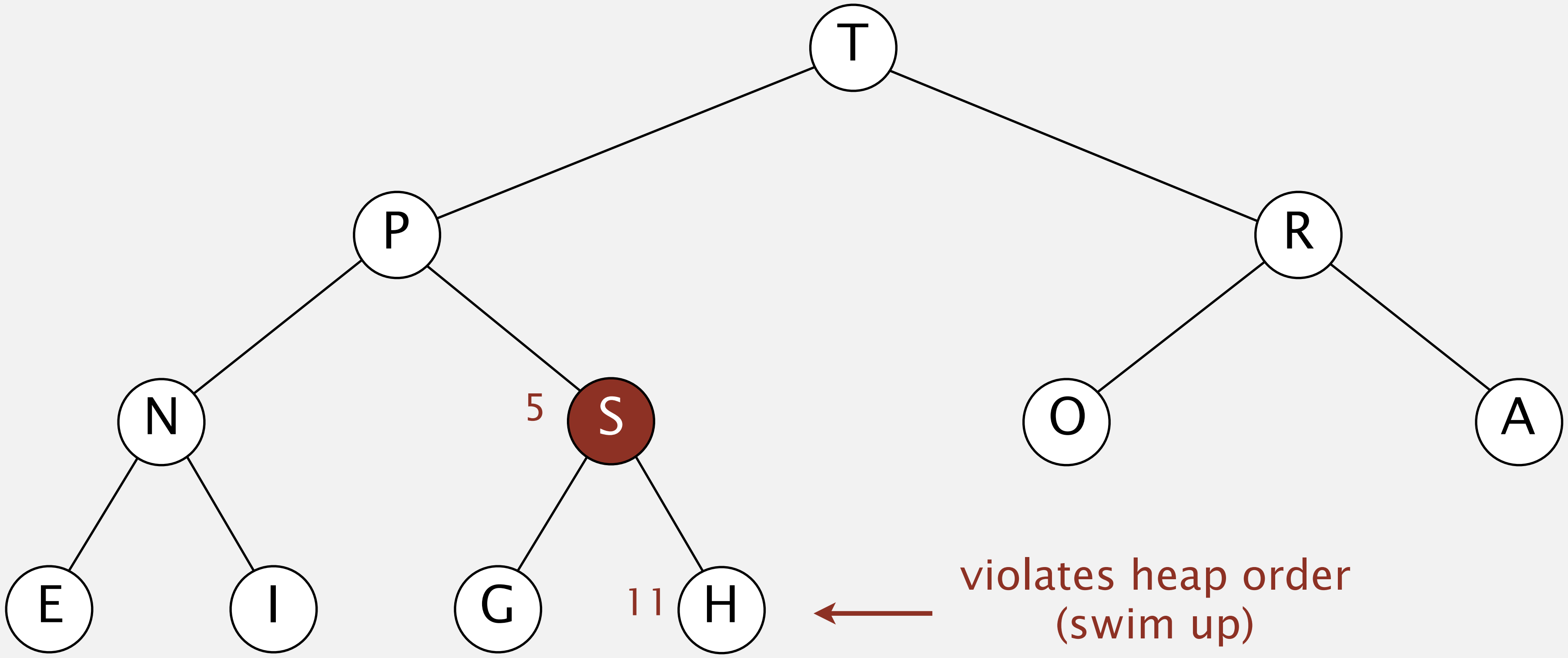
T	P	R	N	H	O	A	E	I	G	S
										11

Binary heap demo (insert example)

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

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

insert S



violates heap order
(swim up)

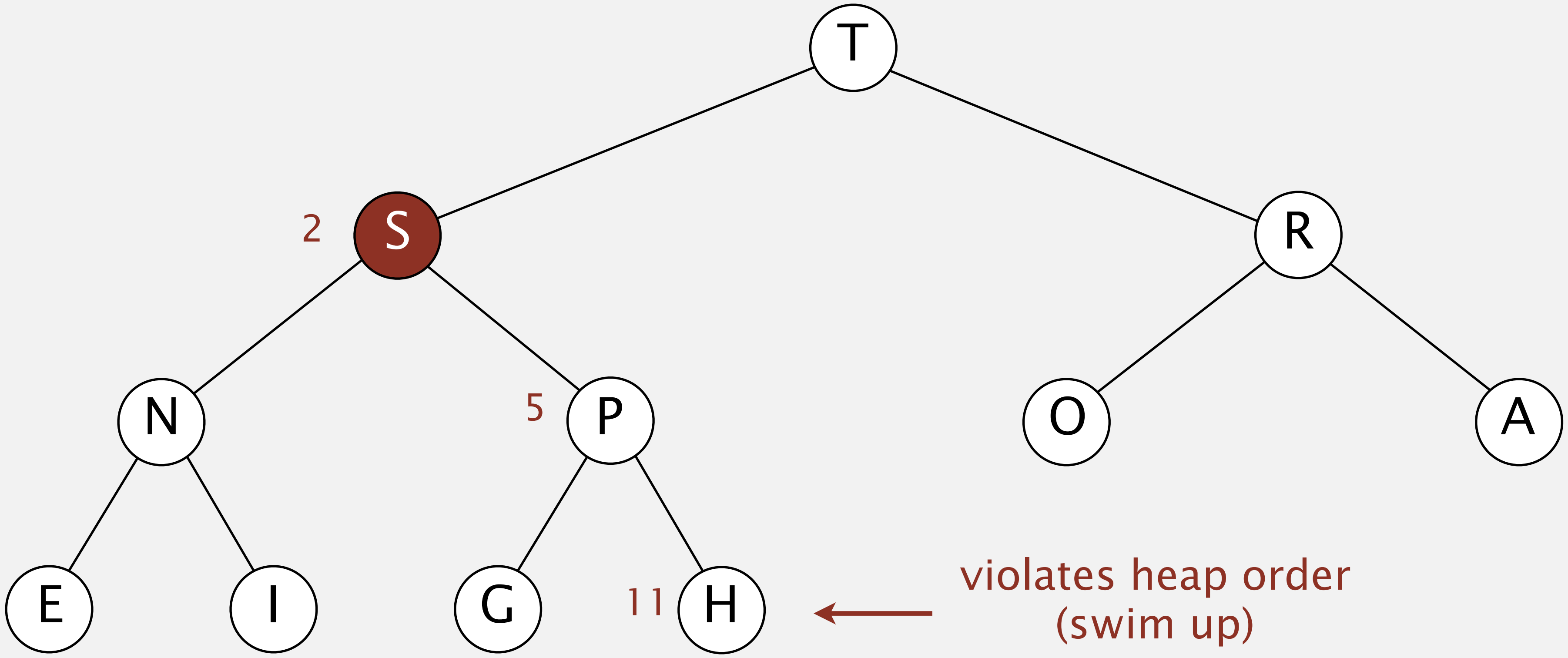
T	P	R	N	S	O	A	E	I	G	H
				5						11

Binary heap demo (insert example)

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

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

insert S



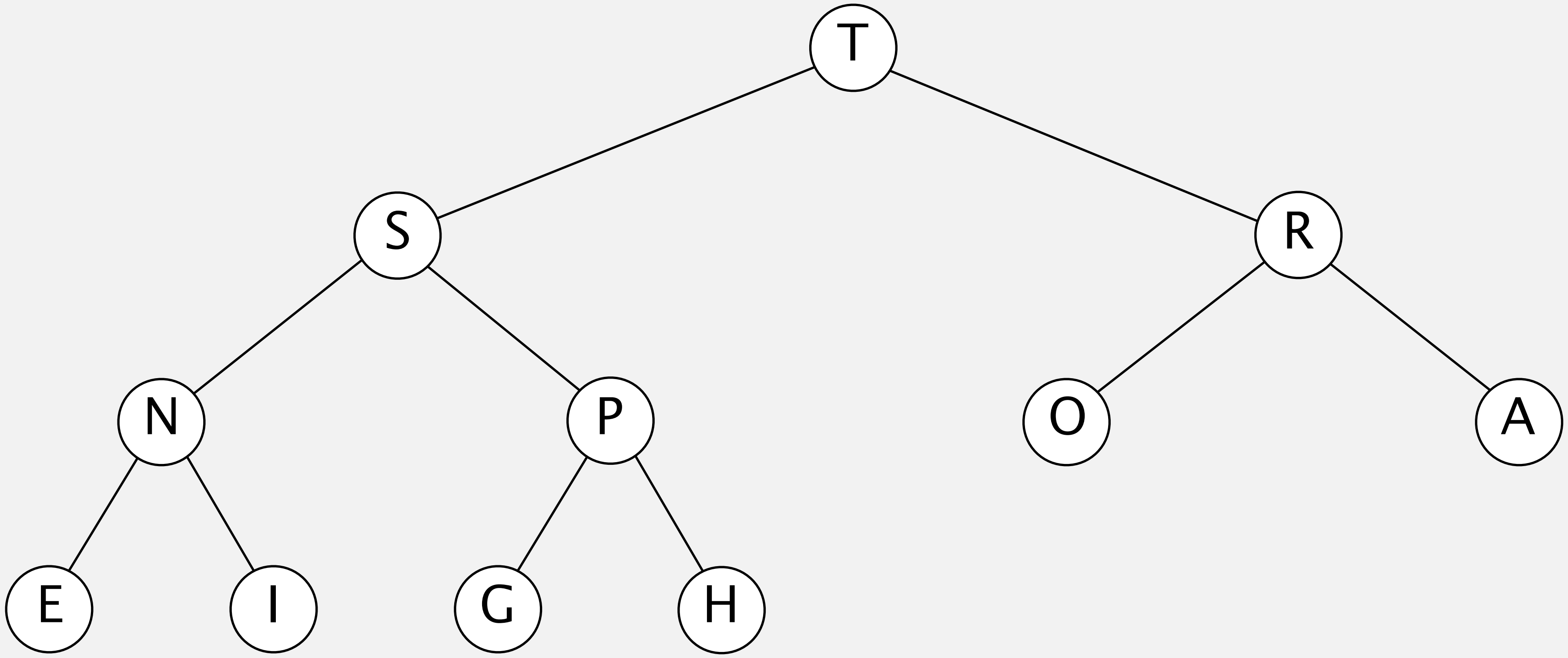
violates heap order (swim up)

T	S	R	N	P	O	A	E	I	G	H
	2			5						11

Binary heap demo (insert example)

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

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



heap ordered

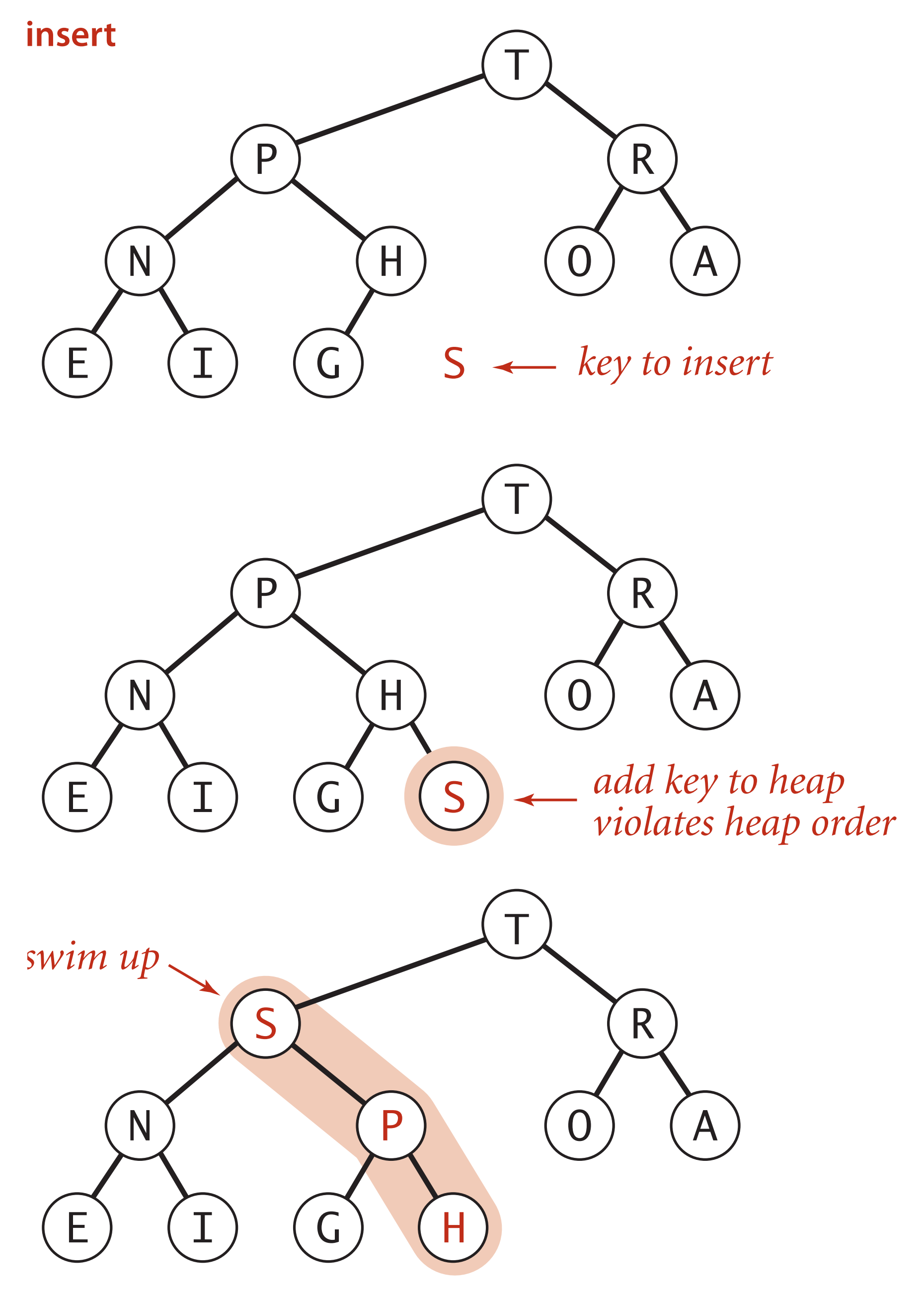
T	S	R	N	P	O	A	E	I	G	H

Insertion in a heap

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

Cost. At most $\sim \lg N$ compares.

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

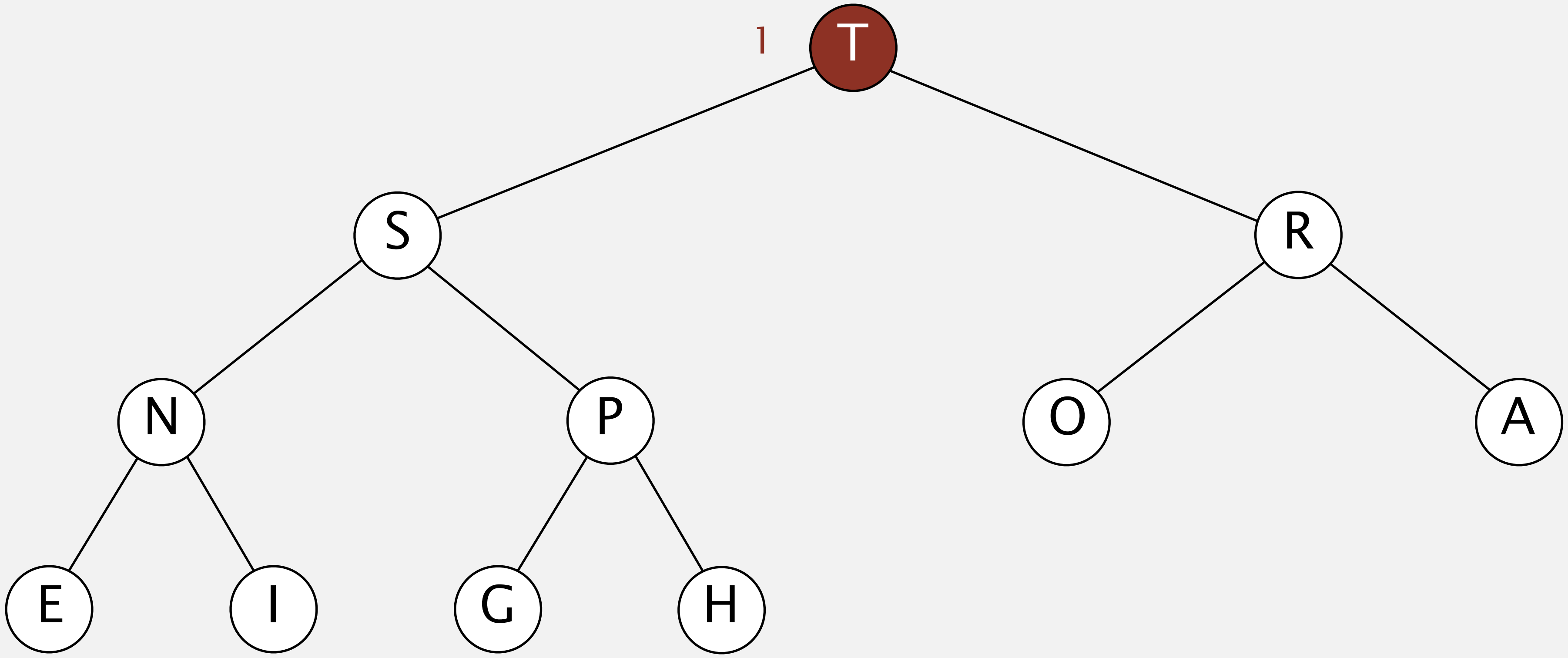


Binary heap demo (remove example)

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

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

remove the maximum



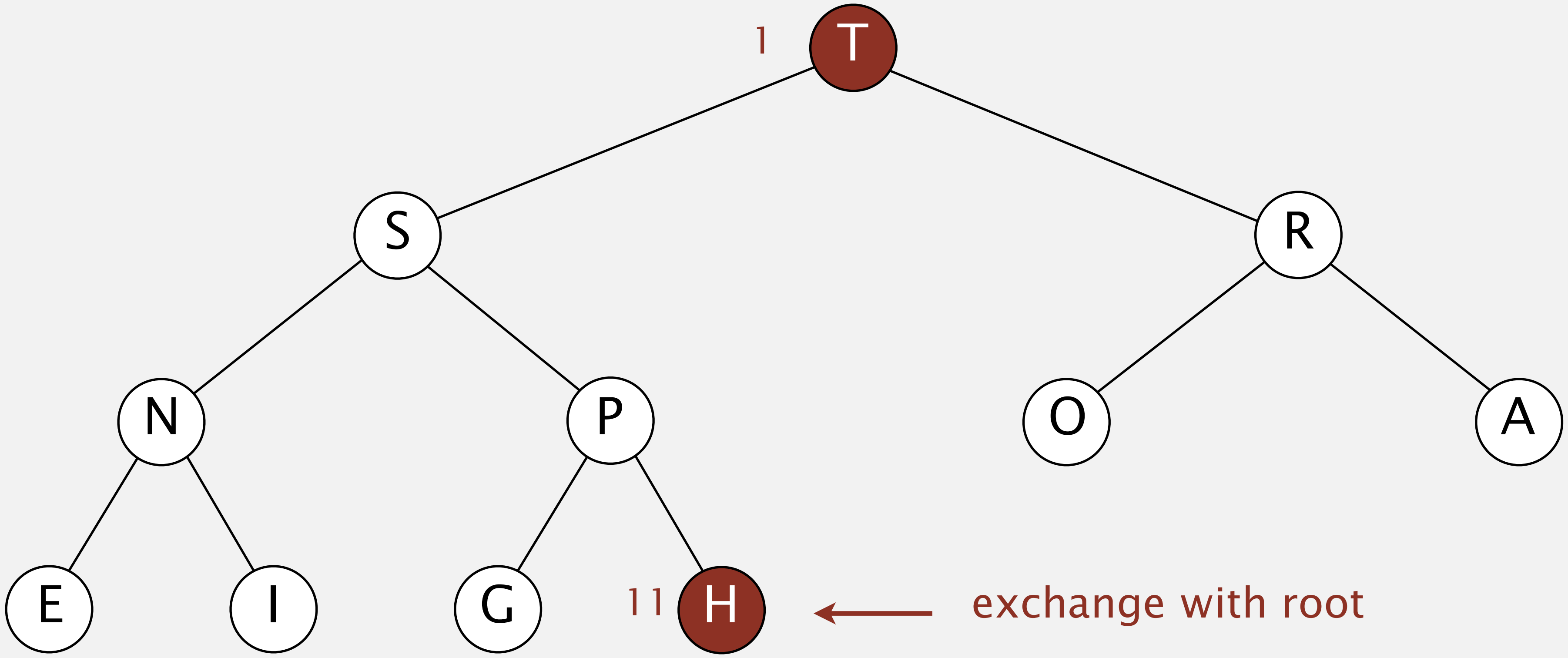
T	S	R	N	P	O	A	E	I	G	H
1										

Binary heap demo (remove example)

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

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

remove the maximum



exchange with root

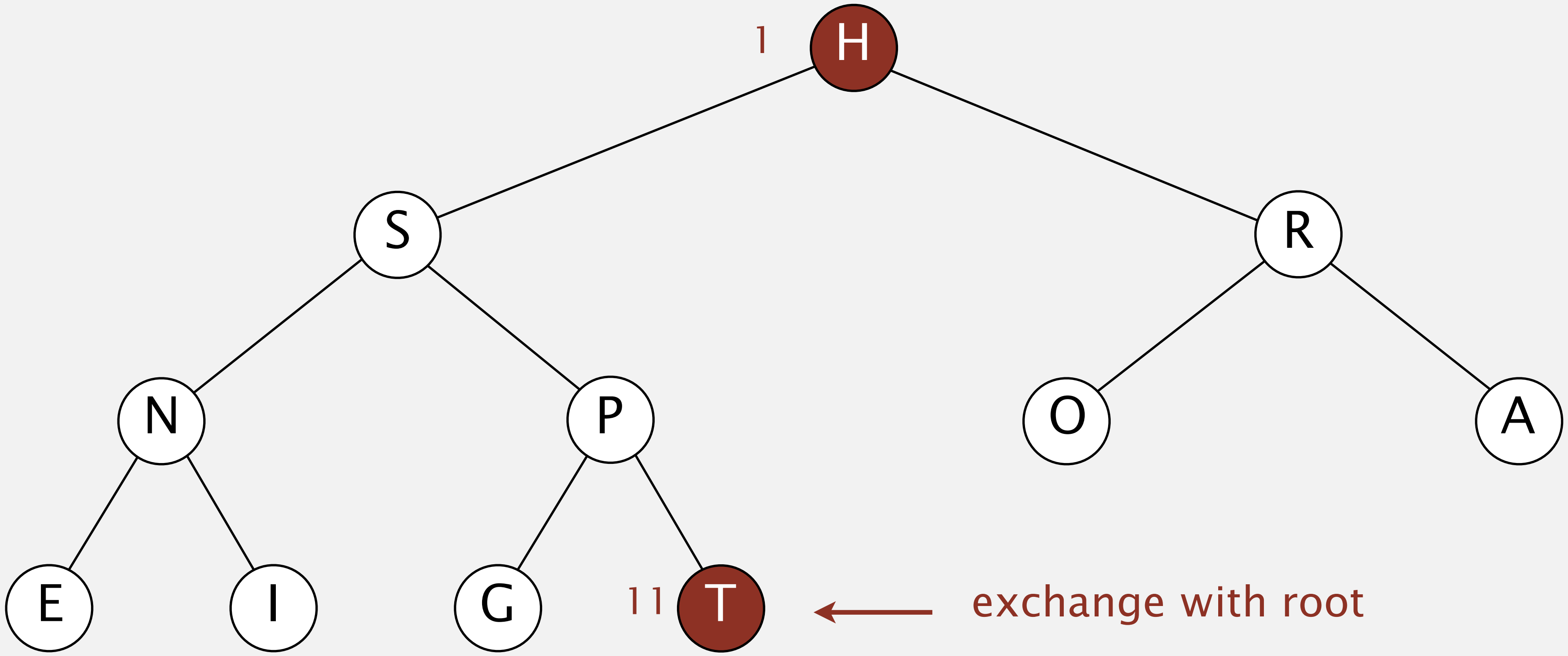
T	S	R	N	P	O	A	E	I	G	H
1										11

Binary heap demo (remove example)

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

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

remove the maximum



exchange with root

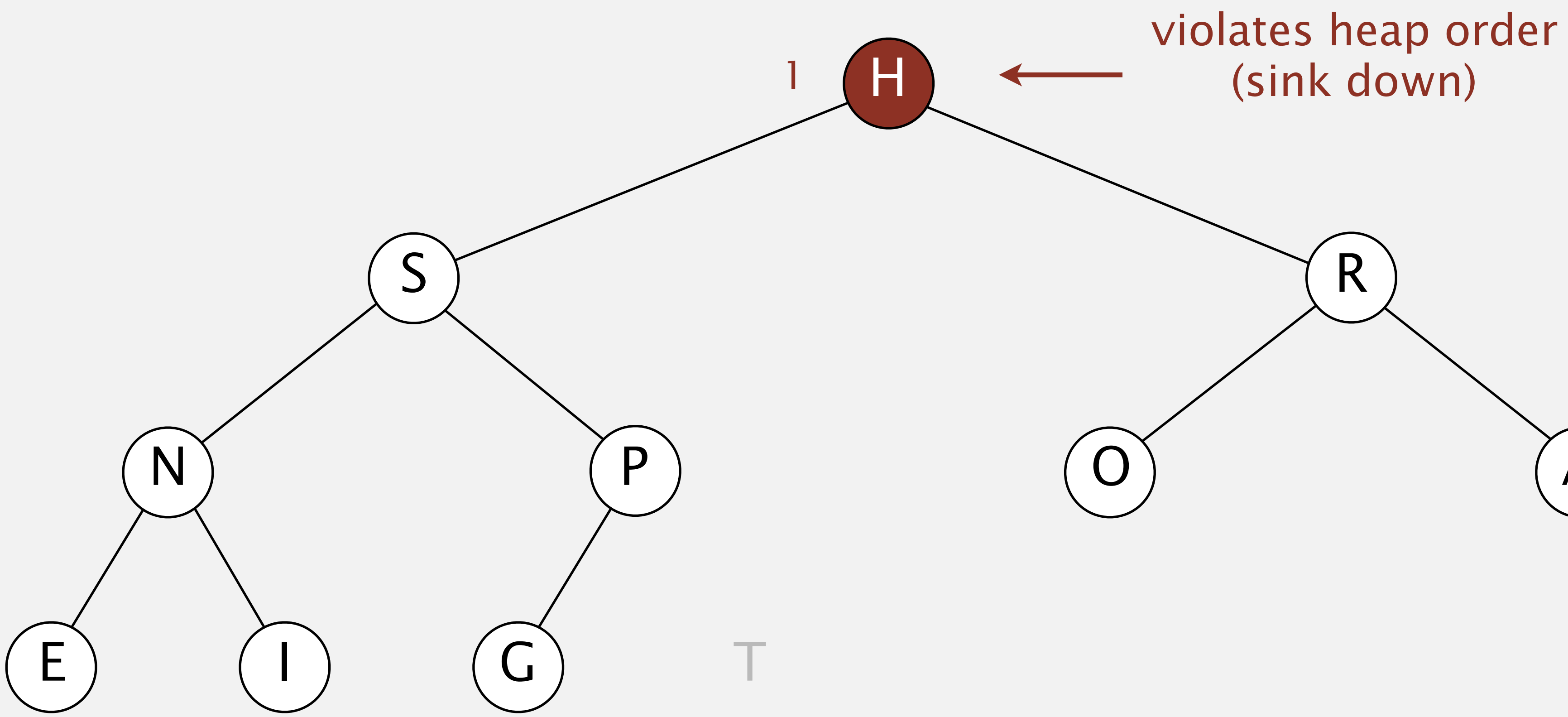
H	S	R	N	P	O	A	E	I	G	T
1										11

Binary heap demo (remove example)

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

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

remove the maximum



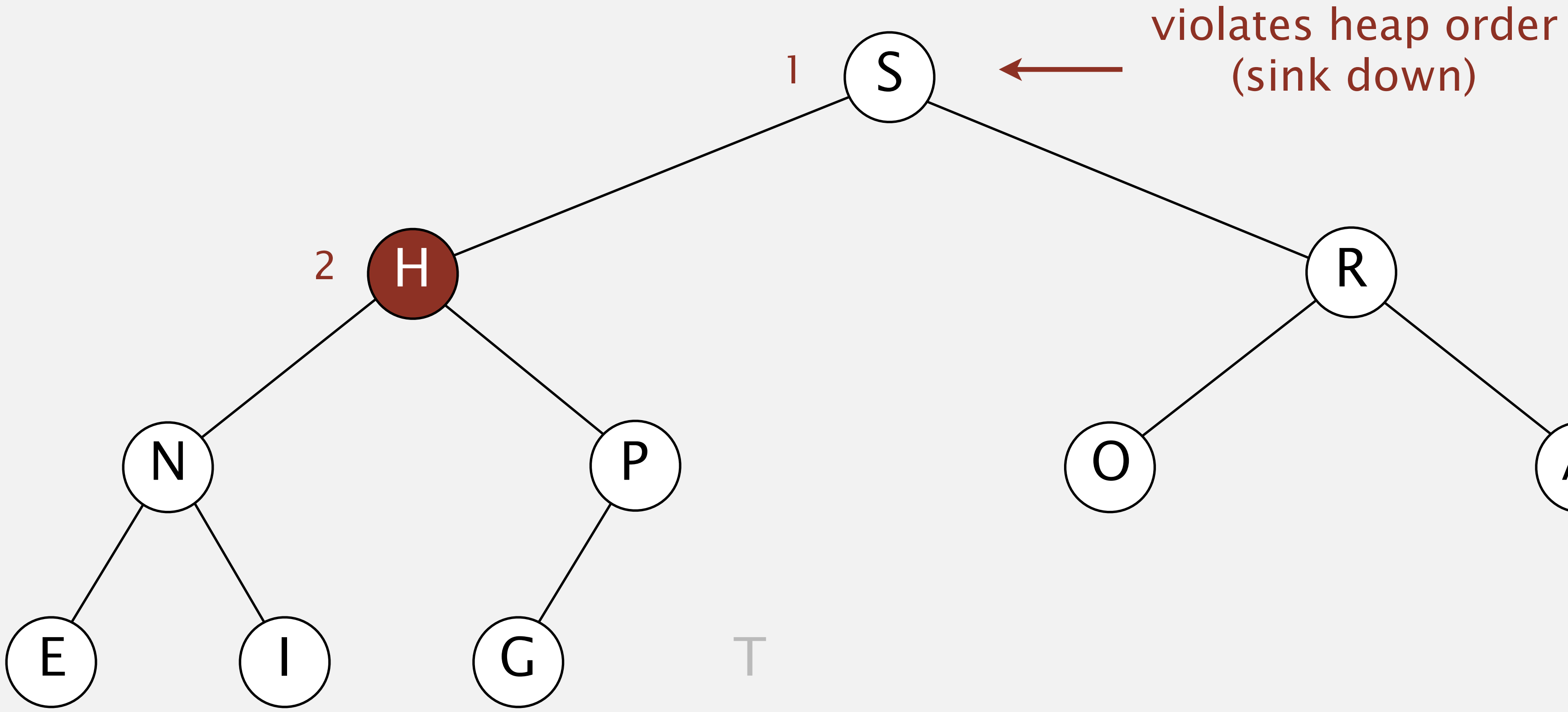
H	S	R	N	P	O	A	E	I	G	T
1										

Binary heap demo (remove example)

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

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

remove the maximum



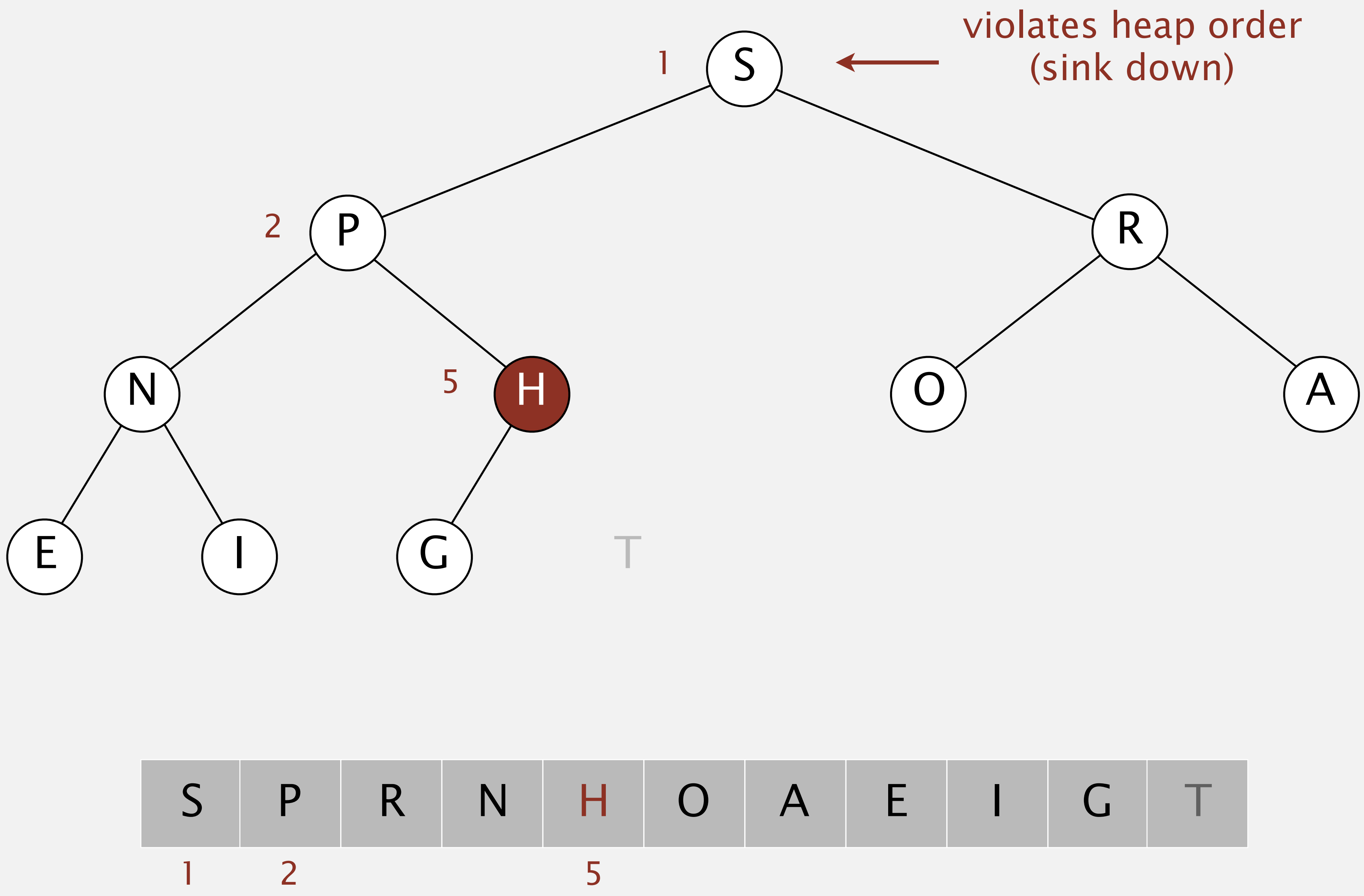
S	H	R	N	P	O	A	E	I	G	T
1	2									

Binary heap demo (remove example)

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

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

remove the maximum

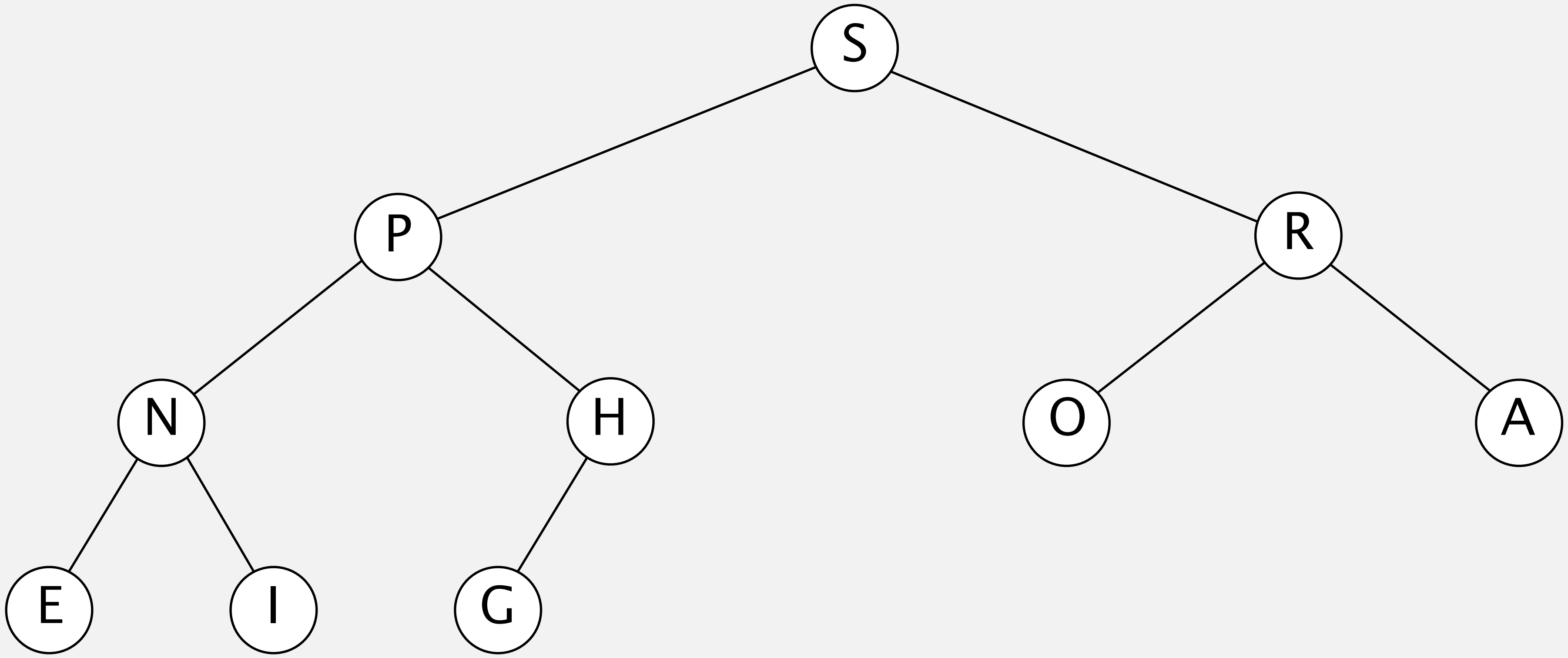


Binary heap demo (remove example)

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

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

heap ordered



S	P	R	N	H	O	A	E	I	G	

Demotion in a heap

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

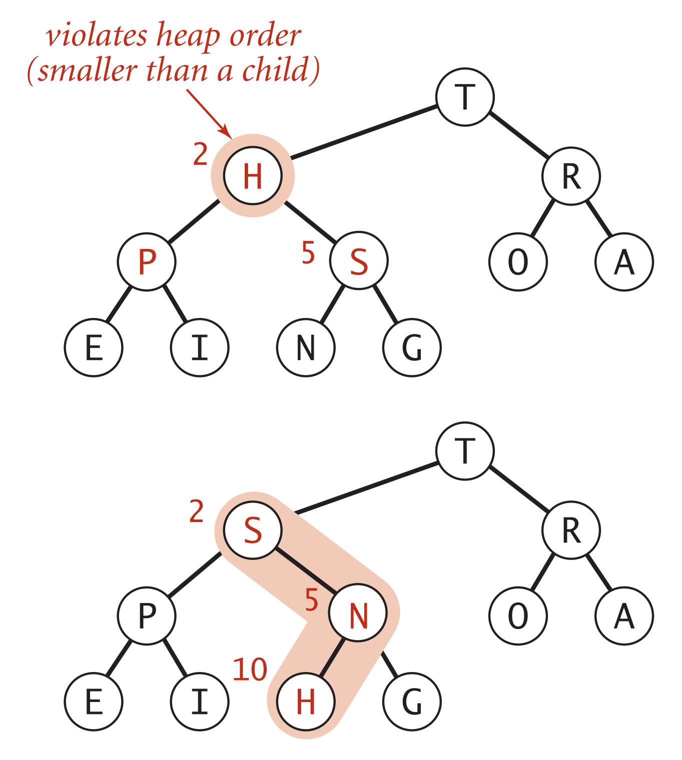
To eliminate the violation:

why not smaller child?

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

```
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 2k and 2k+1



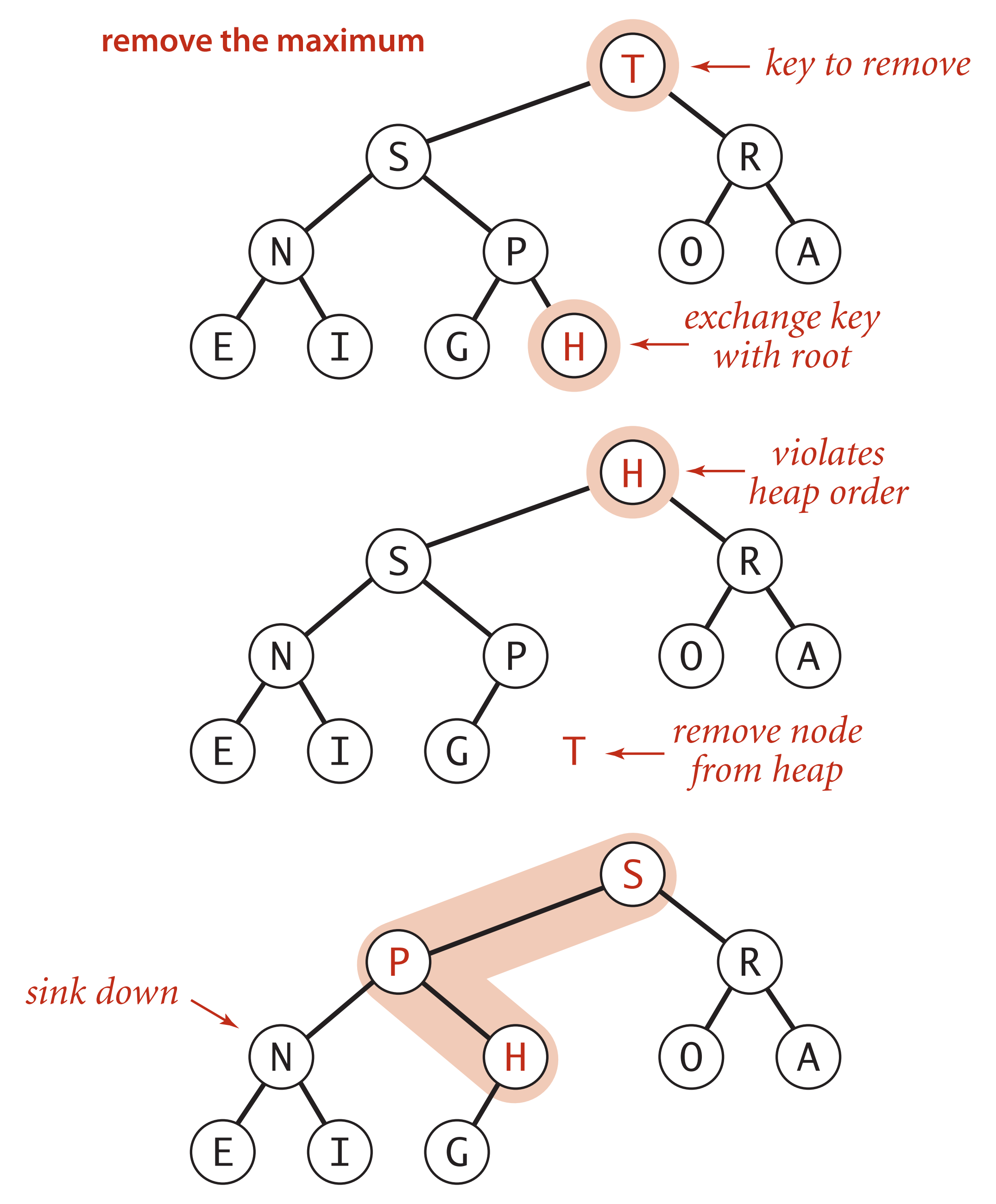
Top-down reheapify (sink)

Delete the maximum in a heap

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);
   return max;
}
```

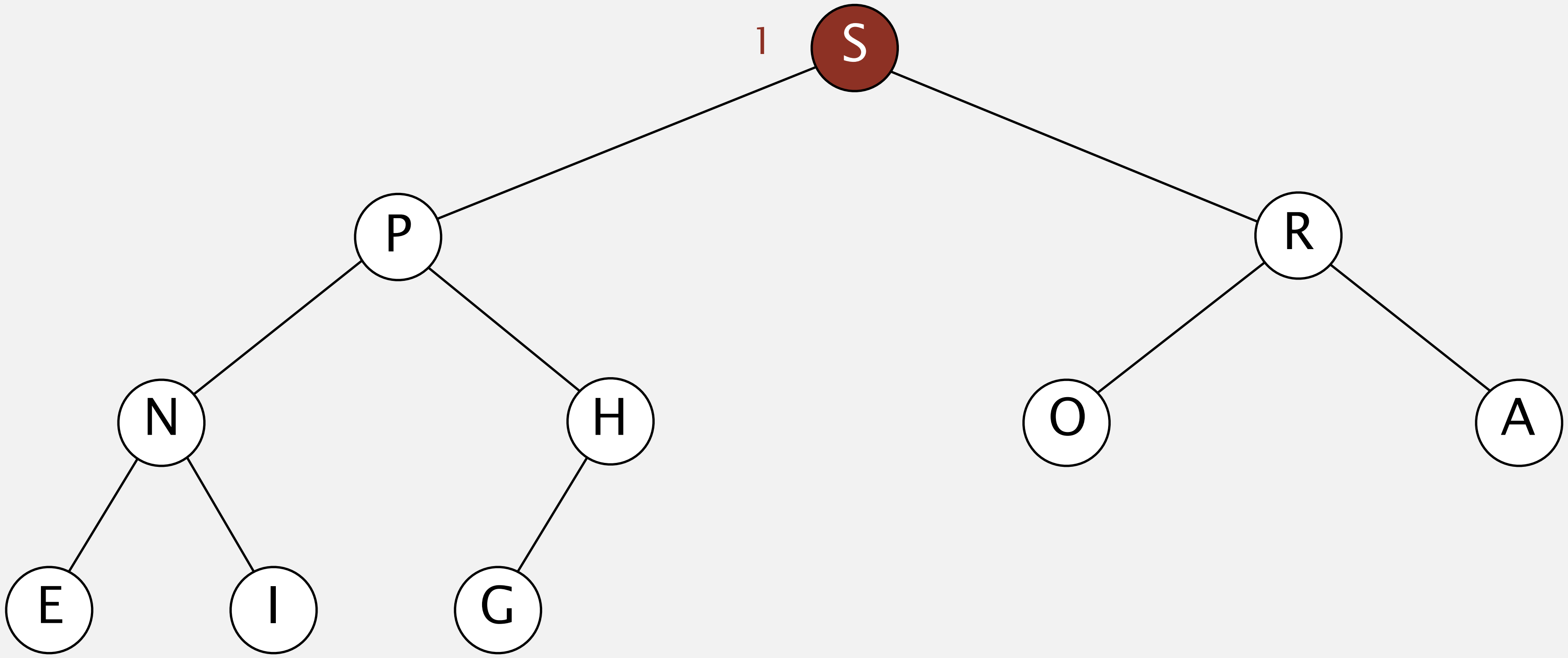


Binary heap demo

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

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

remove the maximum



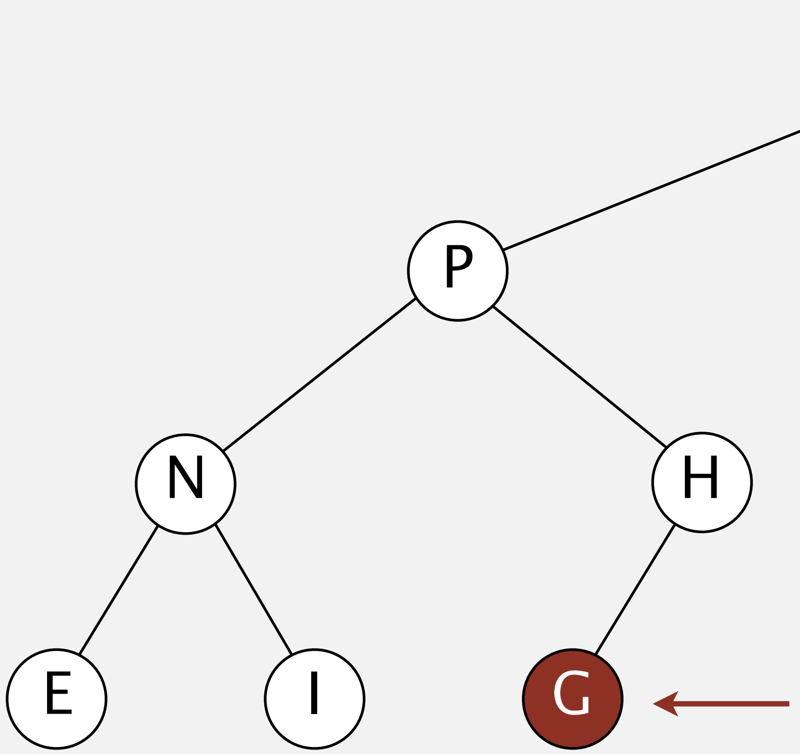
S	P	R	N	H	O	A	E	I	G	
1										

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.

remove the maximum



exchange with root

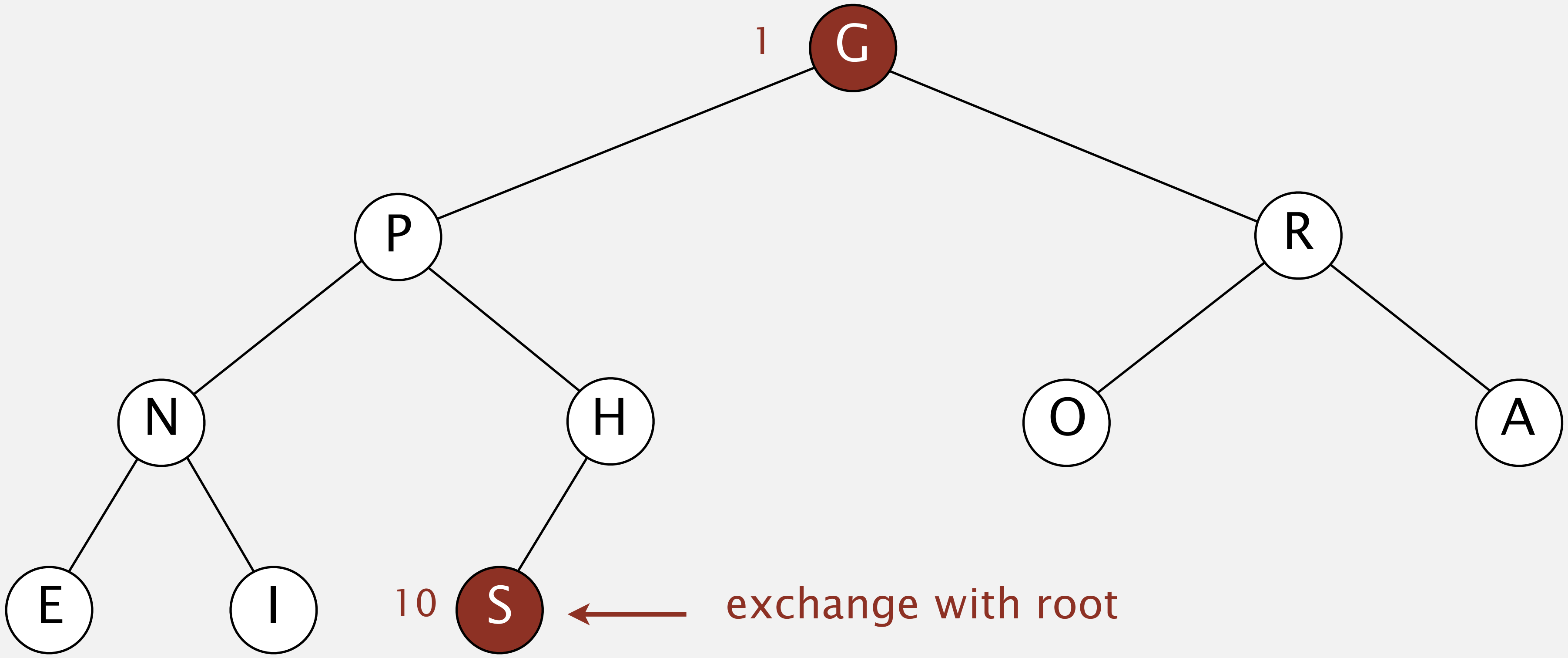
S	P	R	N	H	O	A	E	I	G	
1										

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.

remove the maximum



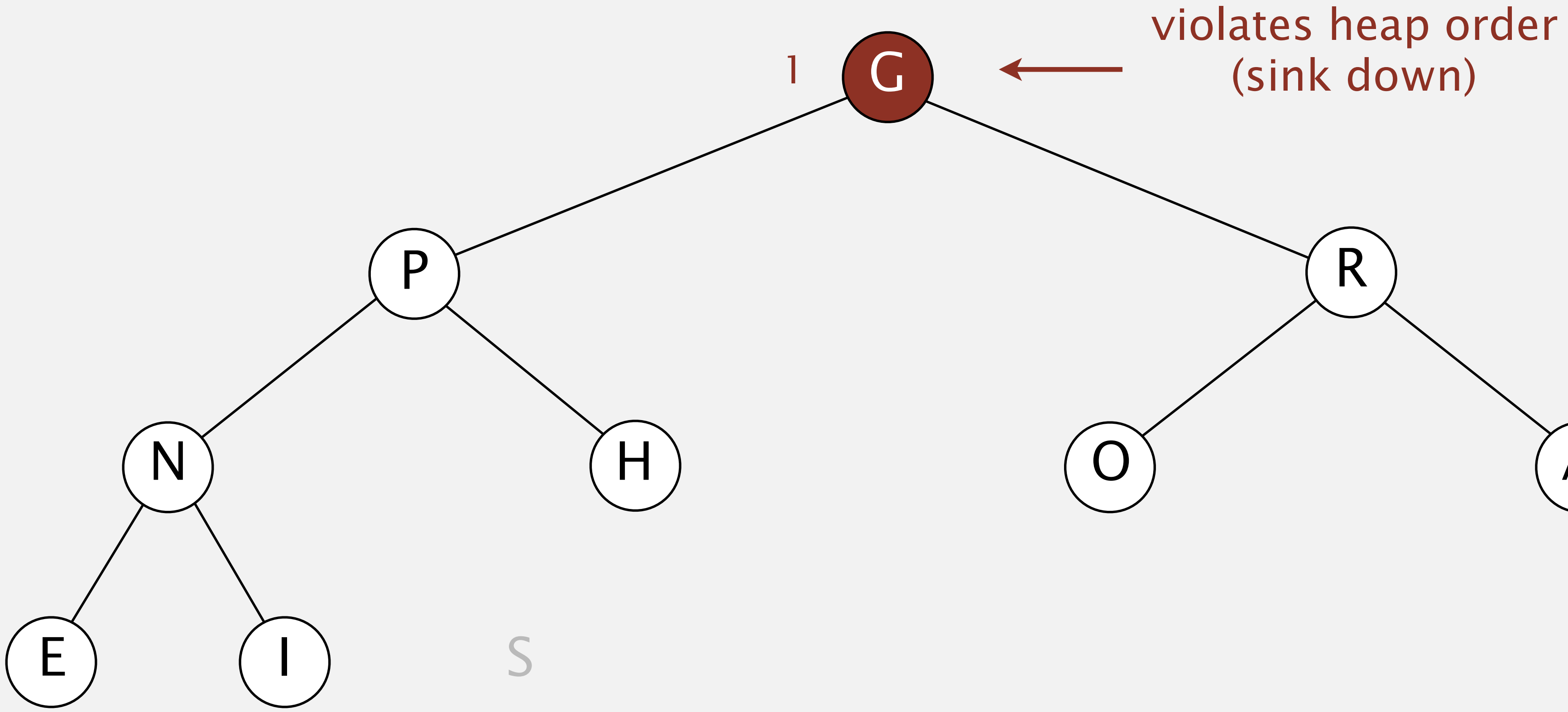
G	P	R	N	H	O	A	E	I	S	
1									10	

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.

remove the maximum



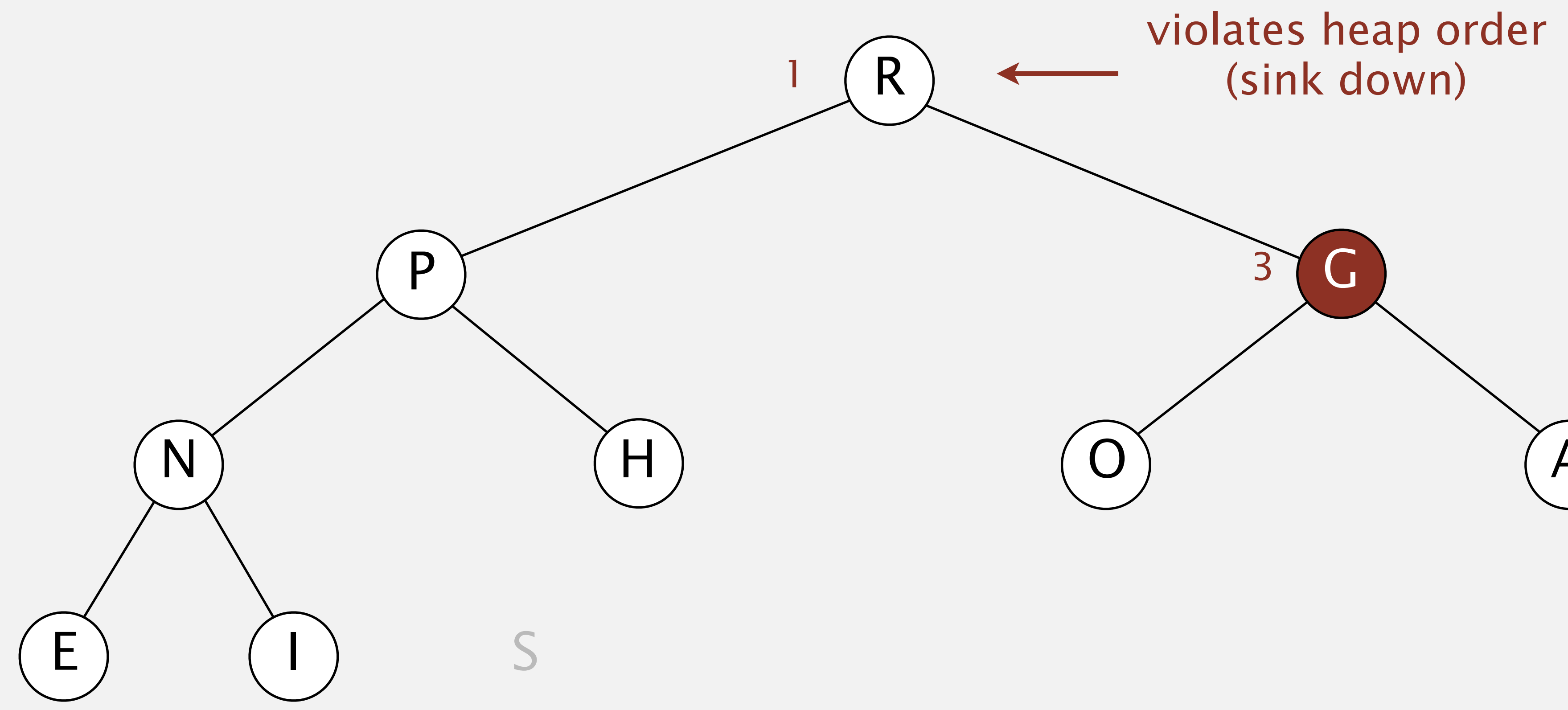
G	P	R	N	H	O	A	E	I	S	
1										

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.

remove the maximum



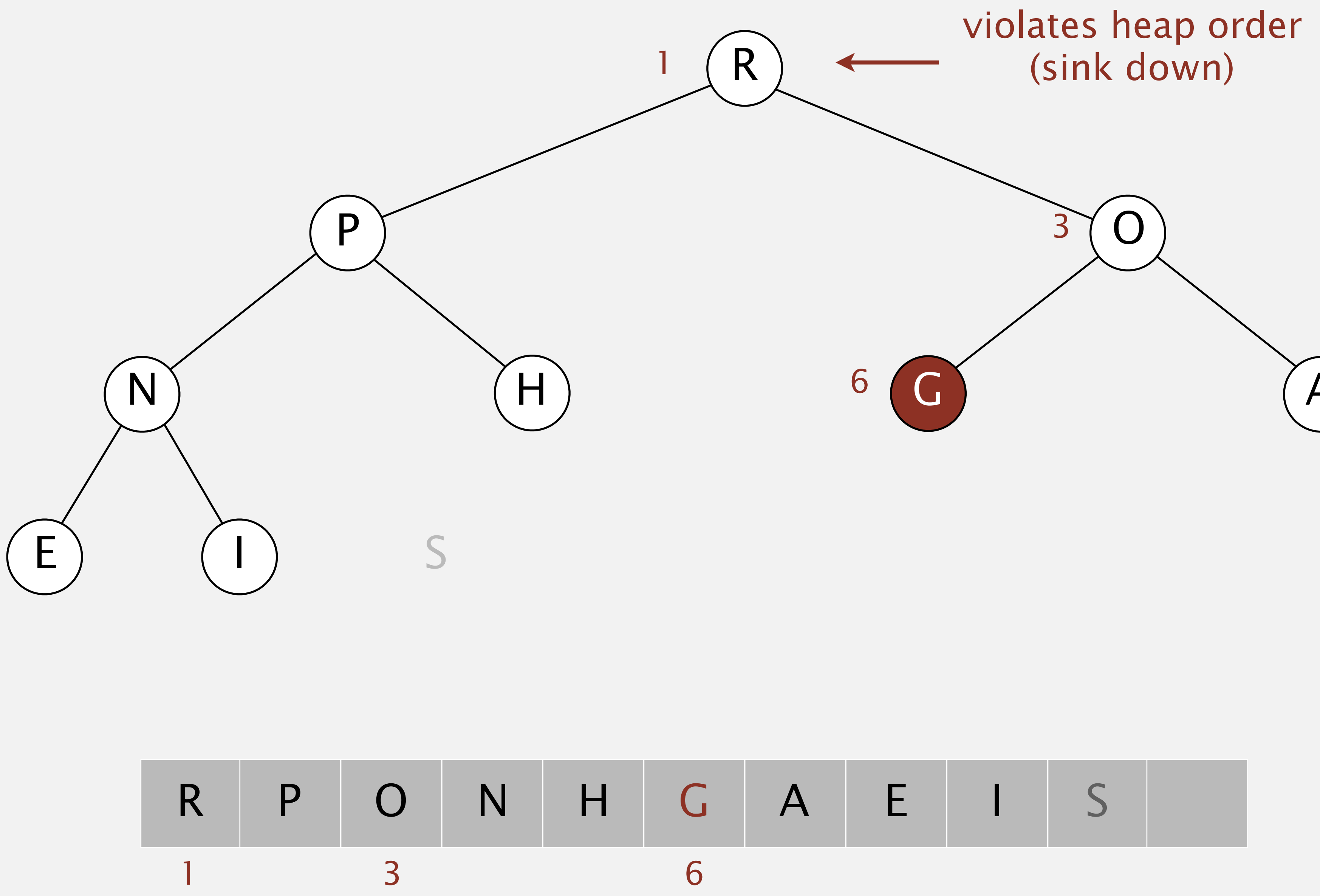
R	P	G	N	H	O	A	E	I	S	
1		3								

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.

remove the maximum

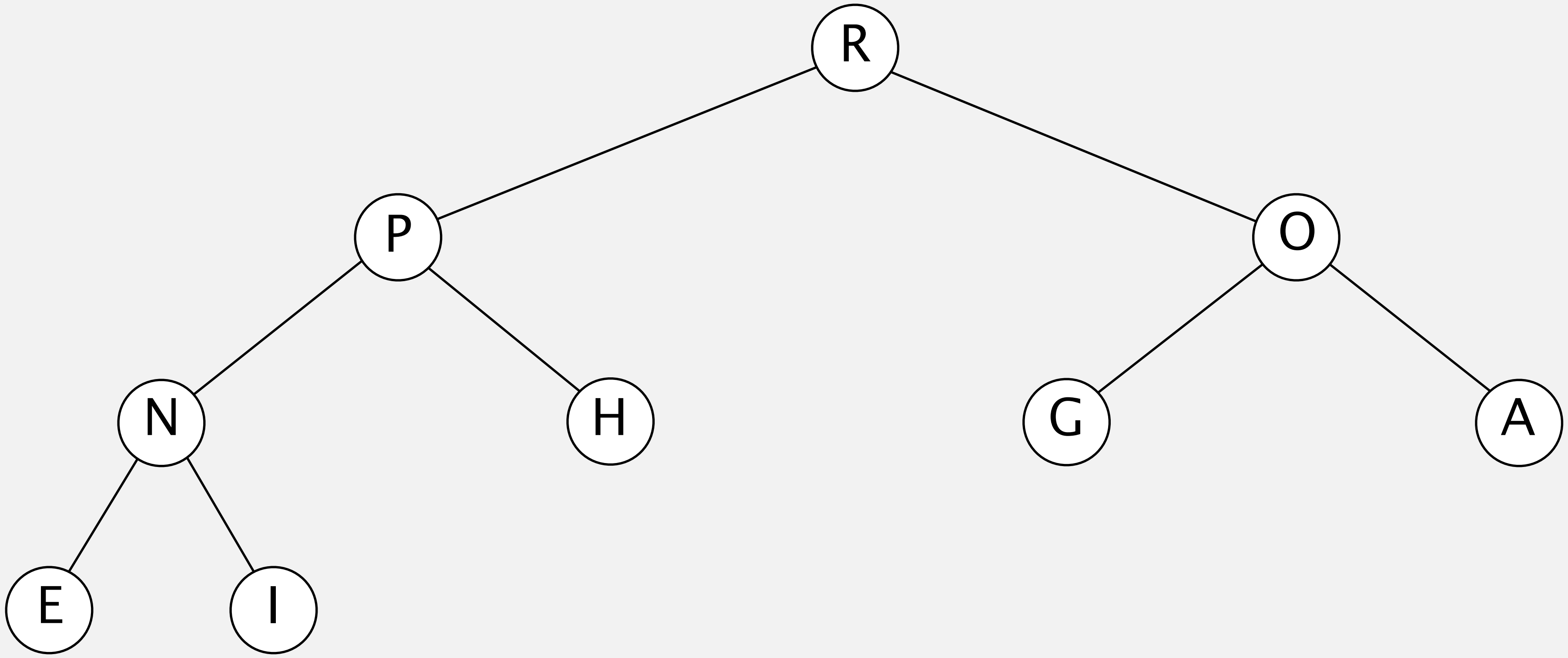


Binary heap demo

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

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

heap ordered



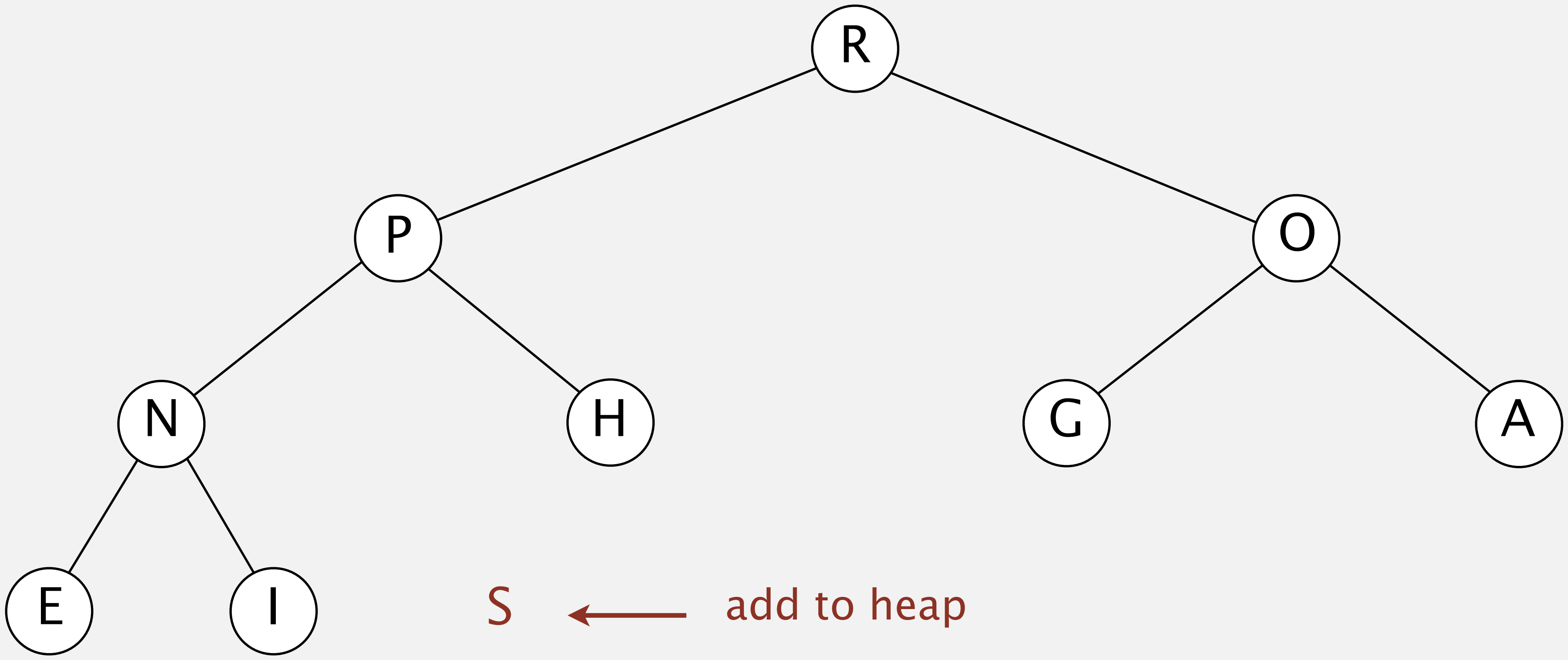
R	P	O	N	H	G	A	E	I		

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.

insert S



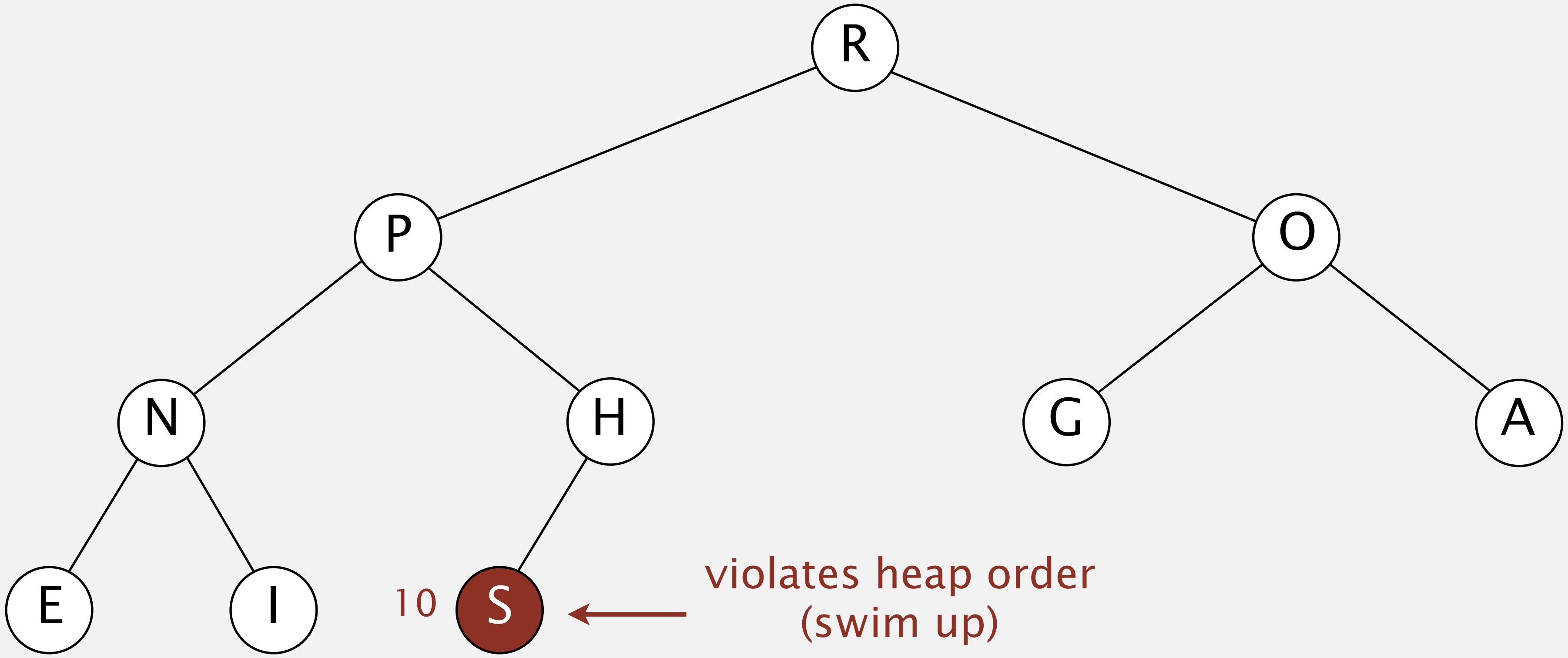
R	P	O	N	H	G	A	E	I	S	
									10	

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.

insert S



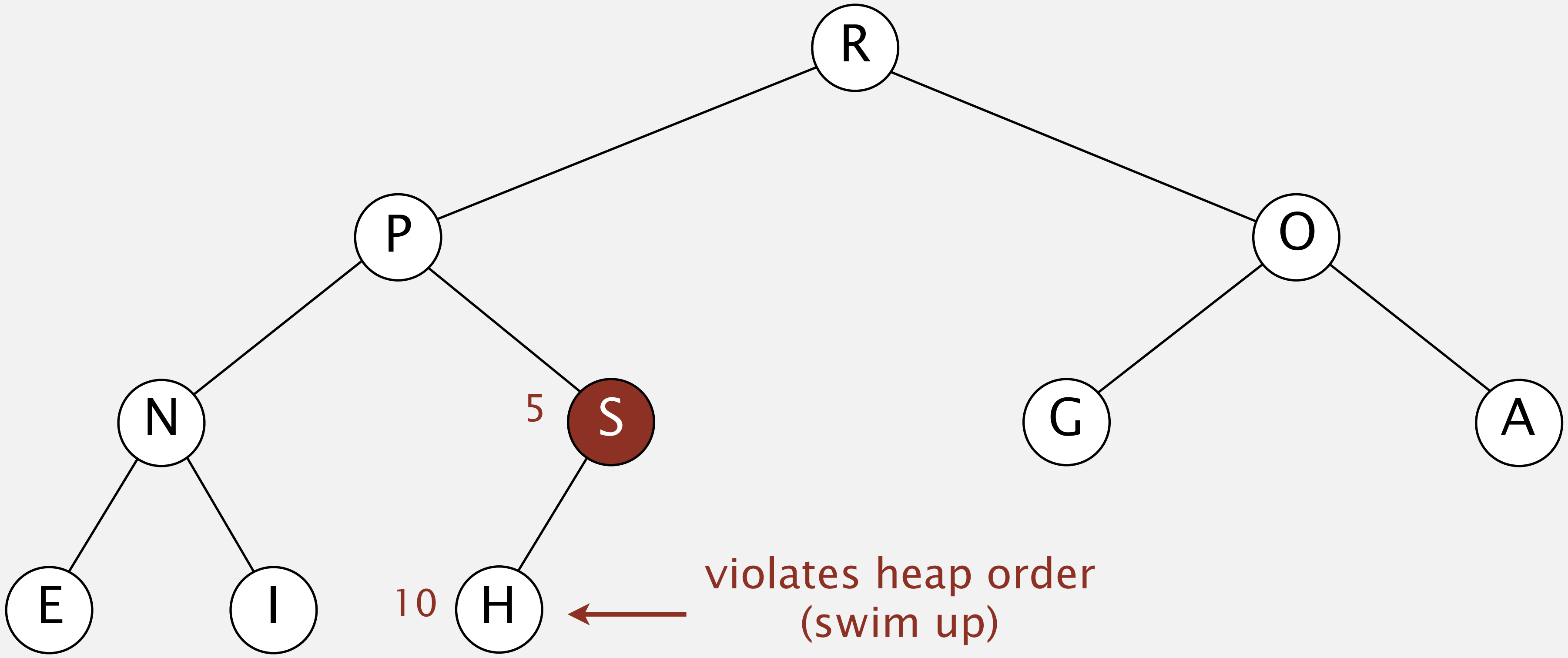
R	P	O	N	H	G	A	E	I	S	
									10	

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.

insert S



violates heap order
(swim up)

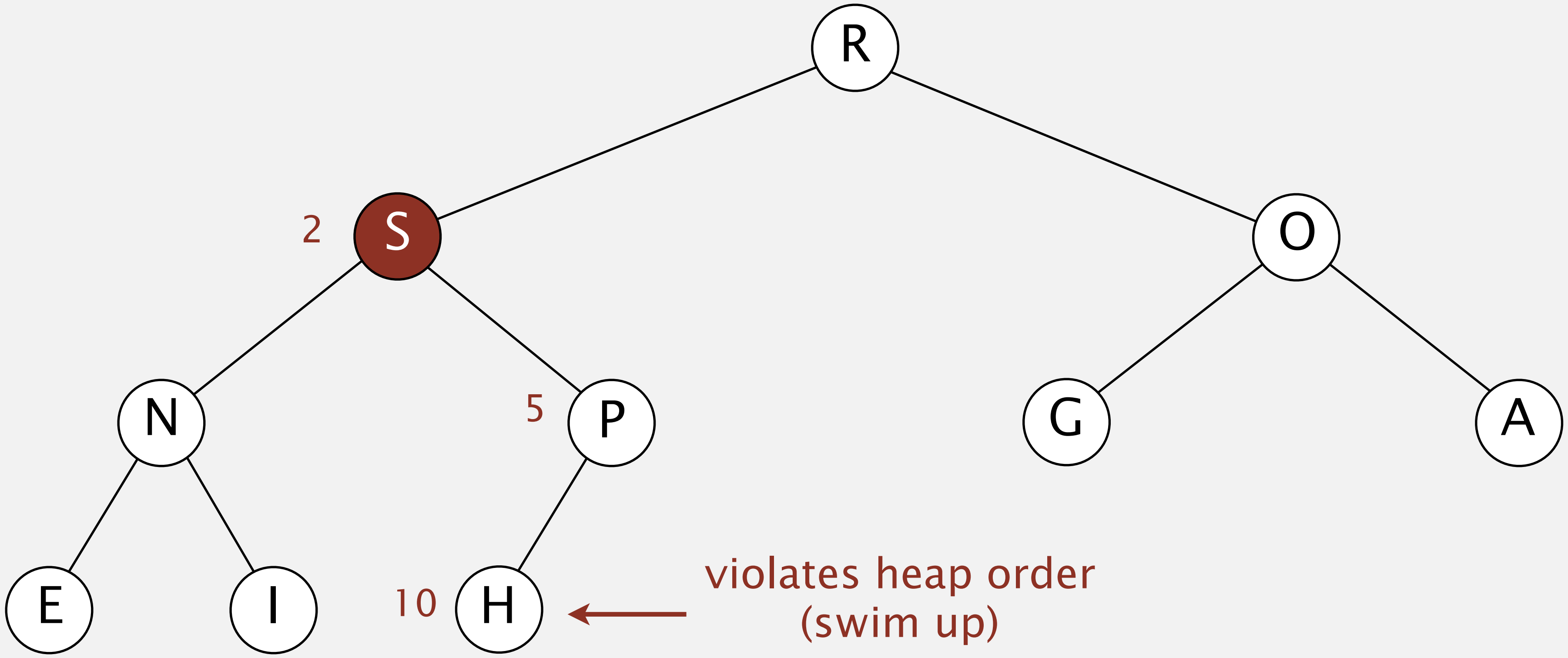
R	P	O	N	S	G	A	E	I	H	
				5					10	

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.

insert S



violates heap order
(swim up)

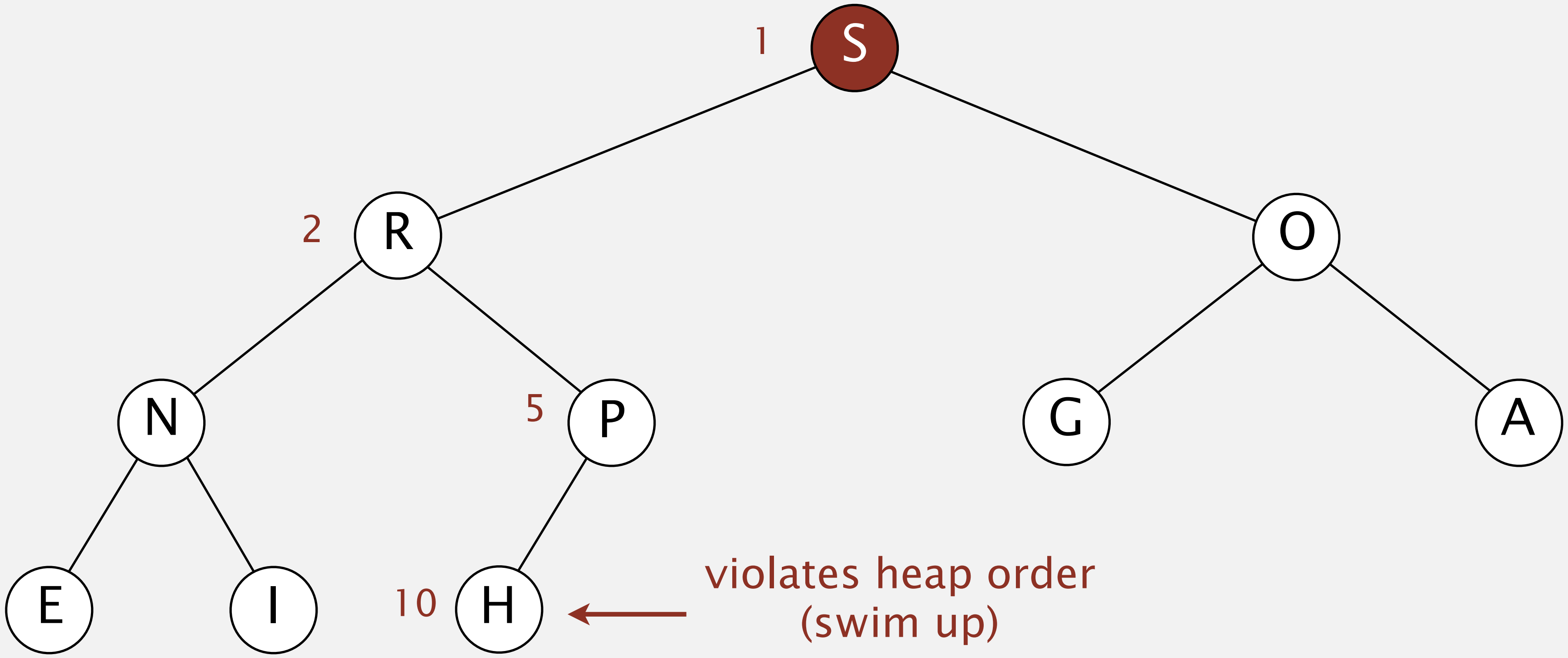
R	S	O	N	P	G	A	E	I	H	
	2			5					10	

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.

insert S



violates heap order (swim up)

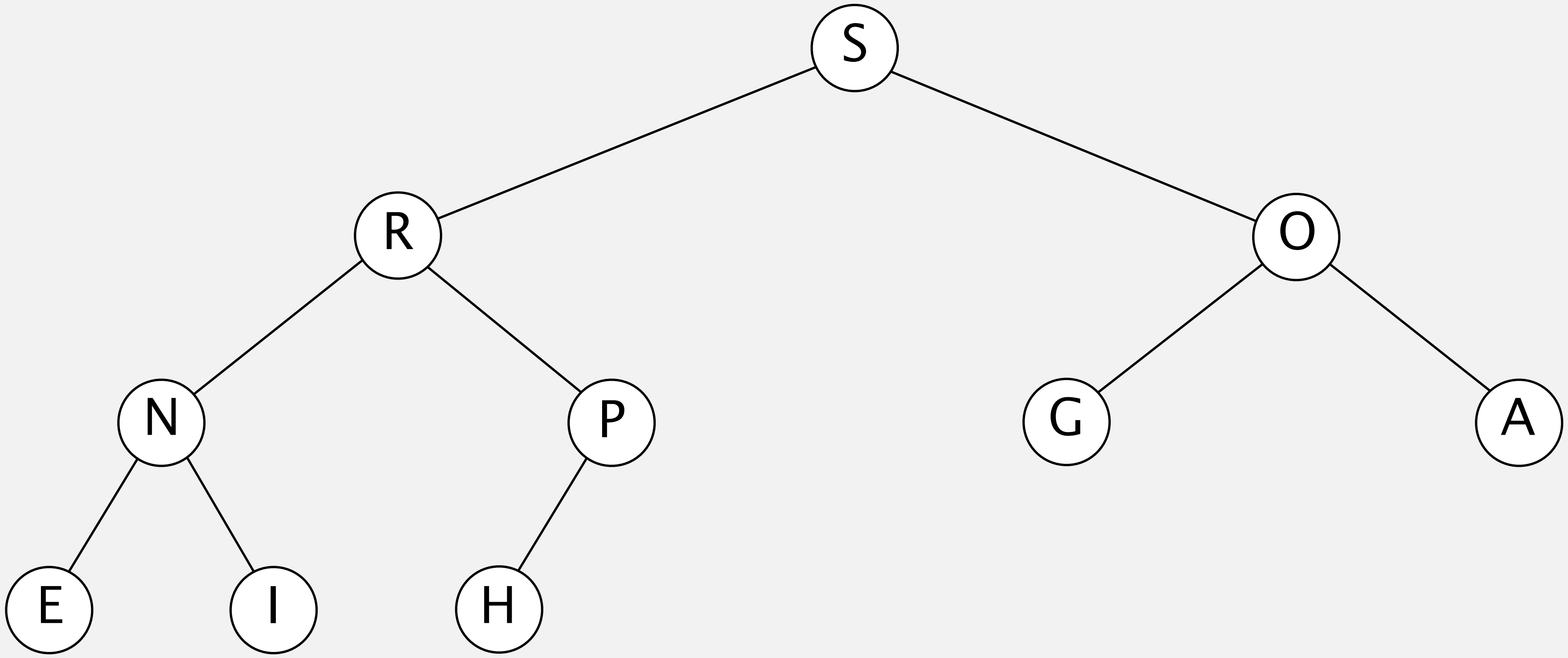
S	R	O	N	P	G	A	E	I	H	
1	2			5					10	

Binary heap demo

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

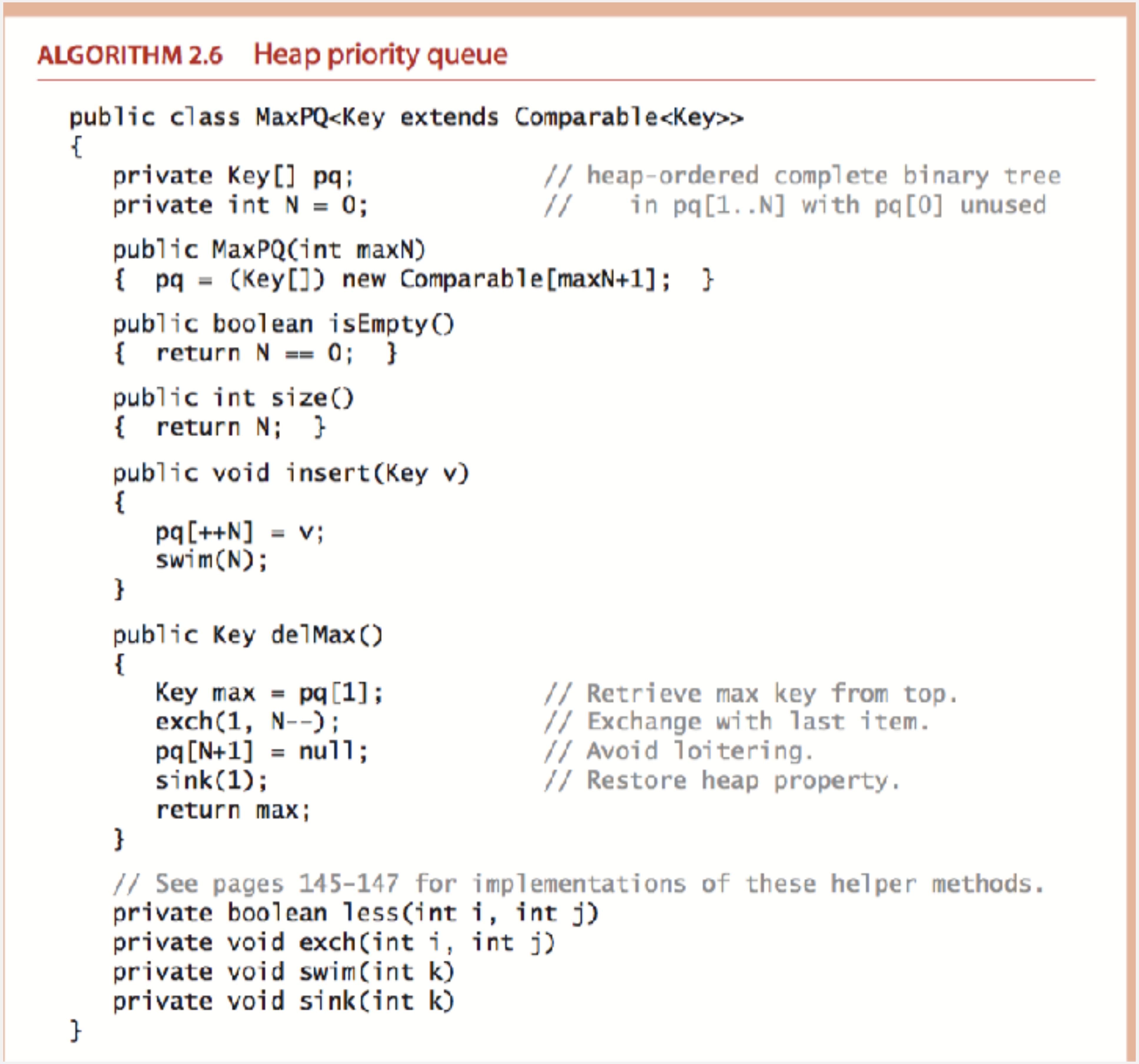
Remove the maximum. Exchange root with node at end, then sink it down.

heap ordered



S	R	O	N	P	G	A	E	I	H	

Binary heap: Java implementation



ALGORITHM 2.6 Heap priority queue

```
public class MaxPQ<Key extends Comparable<Key>>
{
   private Key[] pq;                  // heap-ordered complete binary tree
   private int N = 0;                 //    in pq[1..N] with pq[0] unused

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

   public boolean isEmpty()
   {  return N == 0;  }

   public int size()
   {  return N;  }

   public void insert(Key v)
   {
      pq[++N] = v;
      swim(N);
   }

   public Key delMax()
   {
      Key max = pq[1];                // Retrieve max key from top.
      exch(1, N--);                   // Exchange with last item.
      pq[N+1] = null;                 // Avoid loitering.
      sink(1);                        // Restore heap property.
      return max;
   }

   // See pages 145-147 for implementations of these helper methods.
   private boolean less(int i, int j)
   private void exch(int i, int j)
   private void swim(int k)
   private void sink(int k)
}
```

Priority queues implementation cost summary

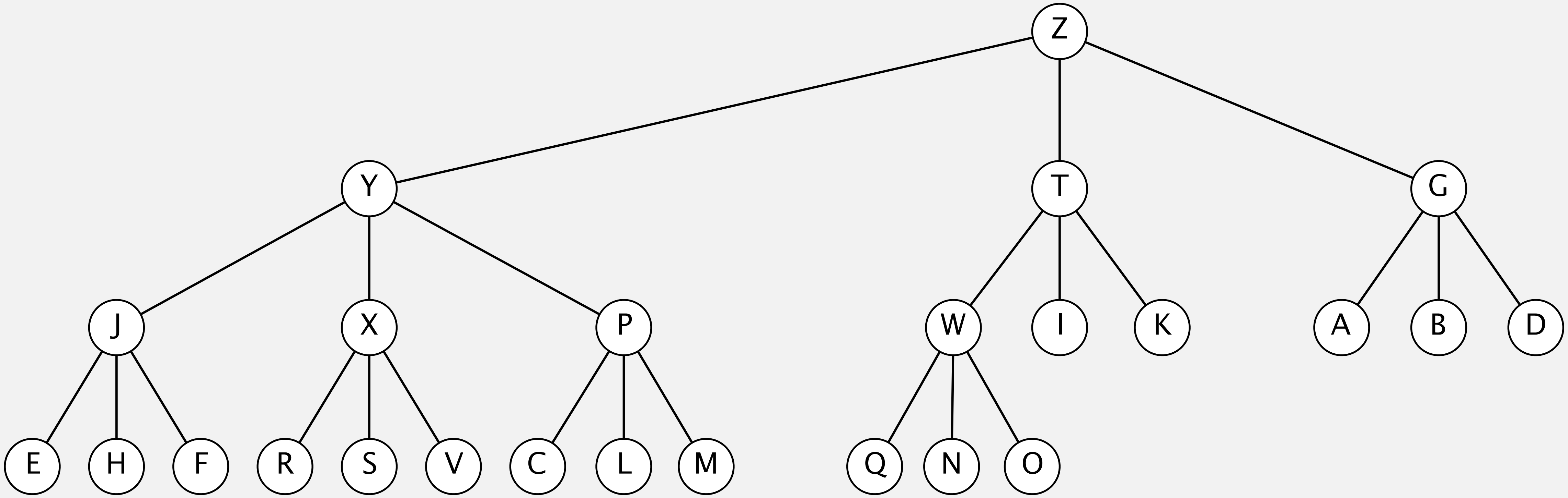
implementation	insert	del max	max
binary heap	$\log M$	$\log M$	1

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

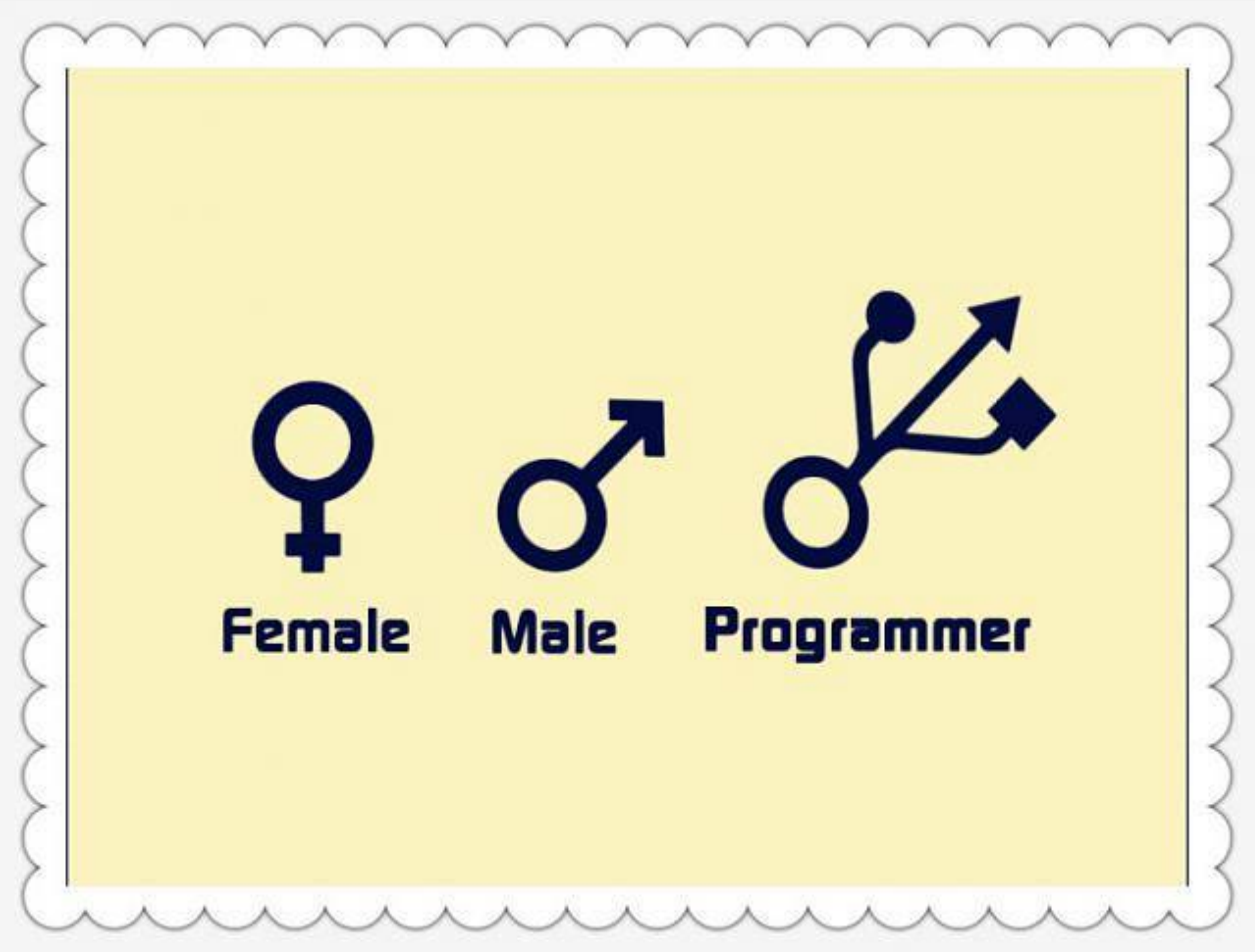
Binary heap: practical improvements

Multiway heaps.

- Complete d-way tree.
- Parent's key no smaller than its children's keys.
- Swim takes $\log_d N$ compares; sink takes $d \log_d N$ compares.
- Trade-off ?????????



3-way heap


Female
Male
Programmer

HEAPSORT

- *binary heaps*
- *in-place heapsort*

Sorting with a binary heap

Q. What is 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();
}
```

Q. What are its properties?

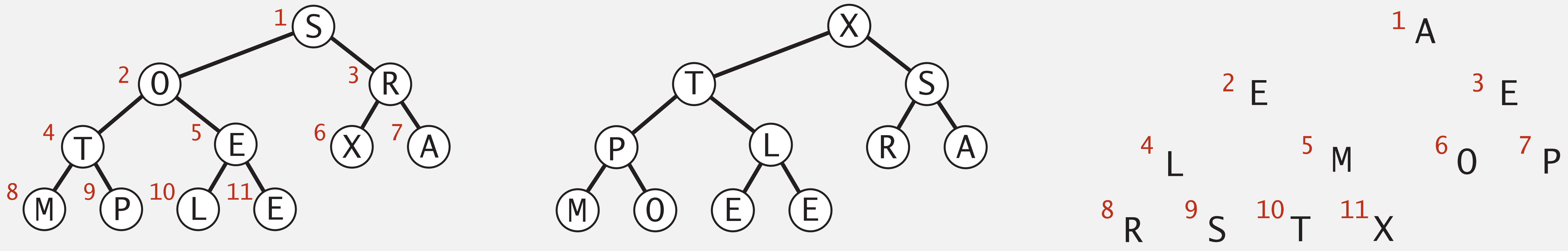
A. $N \log N$, extra array of length N, not stable.

Heapsort intuition. A heap is an array; do sort in place.

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: Java implementation

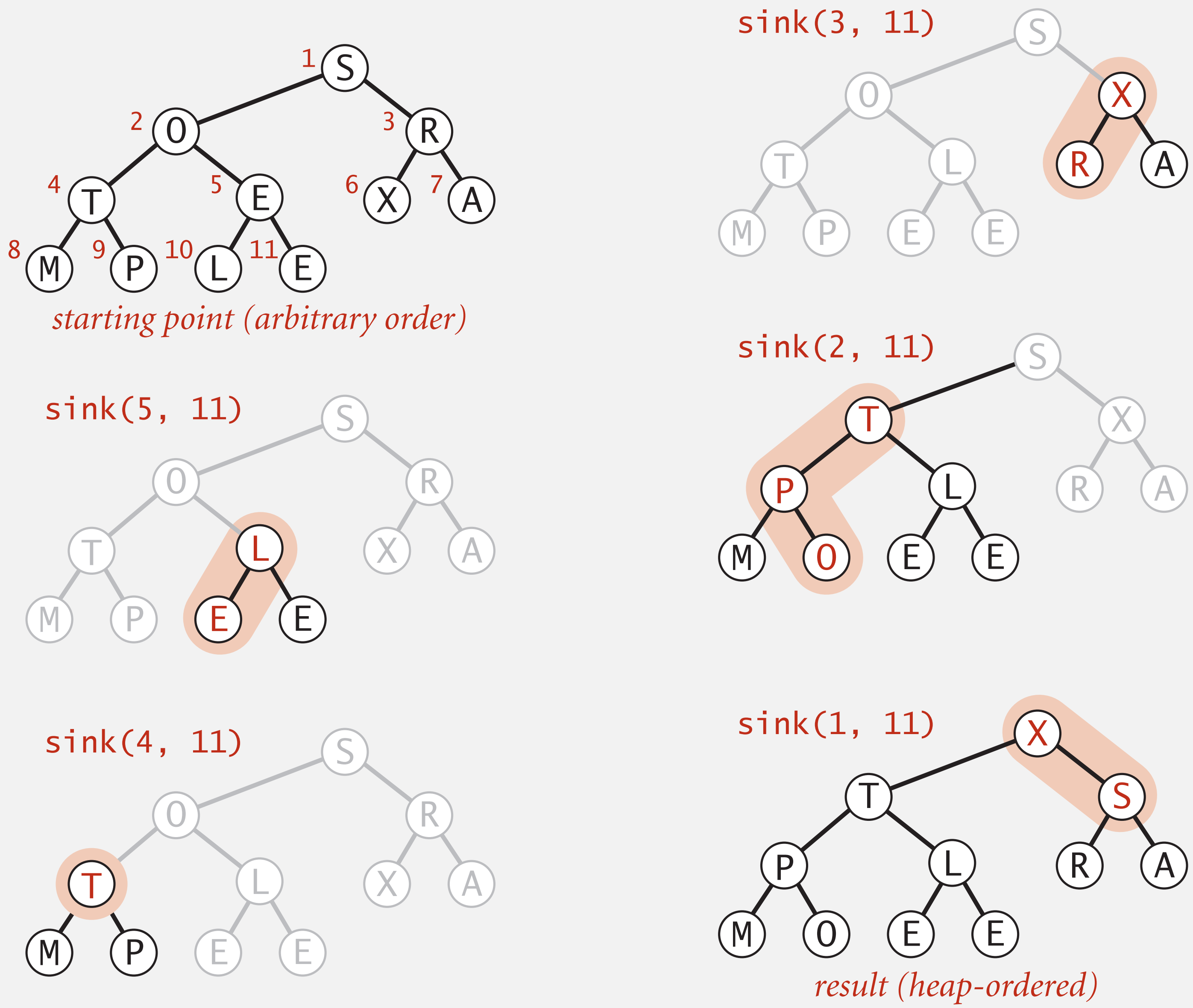
```
public class Heap
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int k = N/2; k >= 1; k--)
         sink(a, k, N);

      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 */  }

}
```

Heapsort: sink-based heap construction

First pass. Build heap using bottom-up method.

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

N/2: the number of second leafs

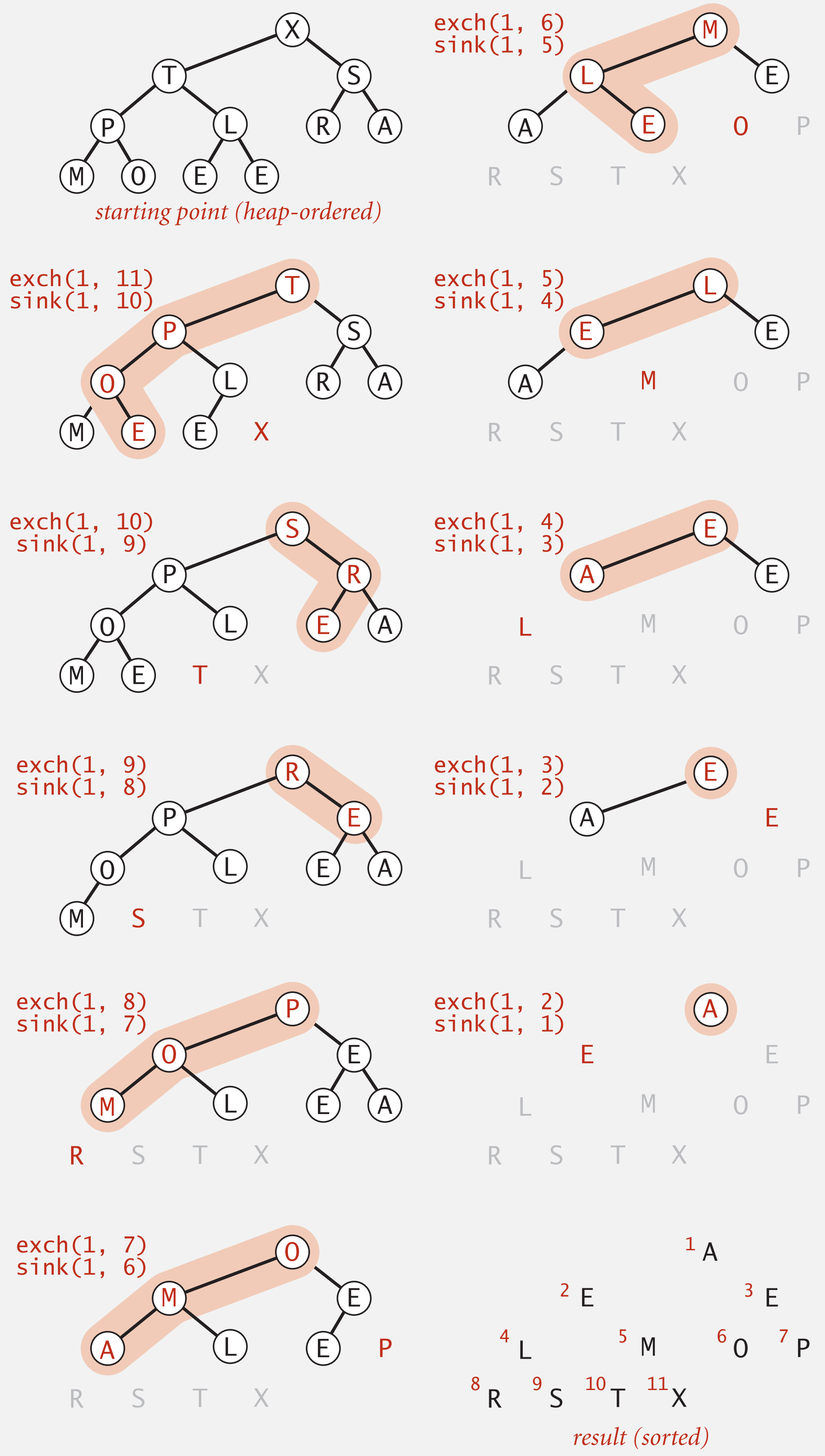


Heapsort: sortdown

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);
}
```



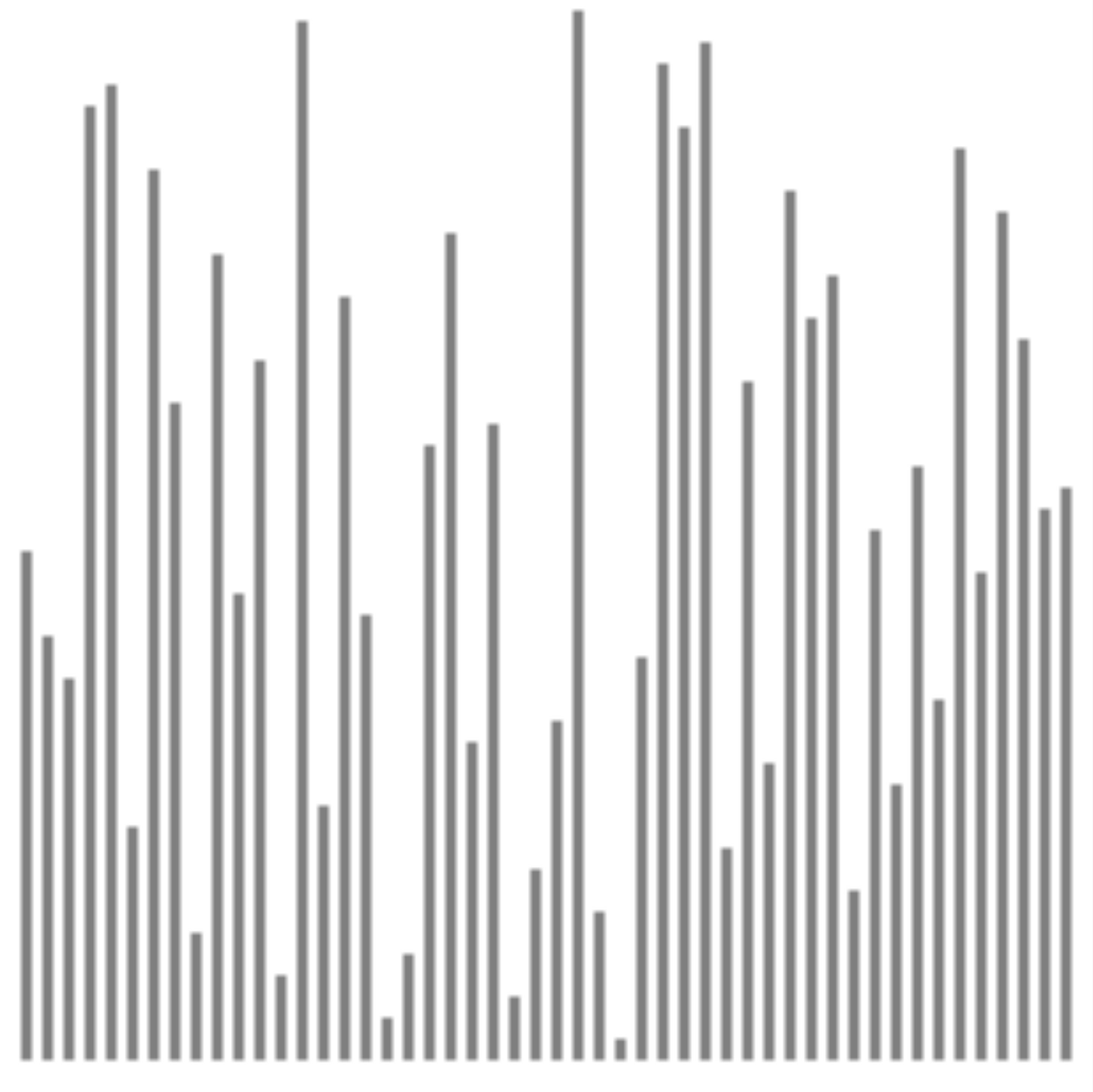
Heapsort: trace

		a[i]											
N	k	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 animation

50 random items



http://www.sorting-algorithms.com/heap-sort

▲ algorithm position

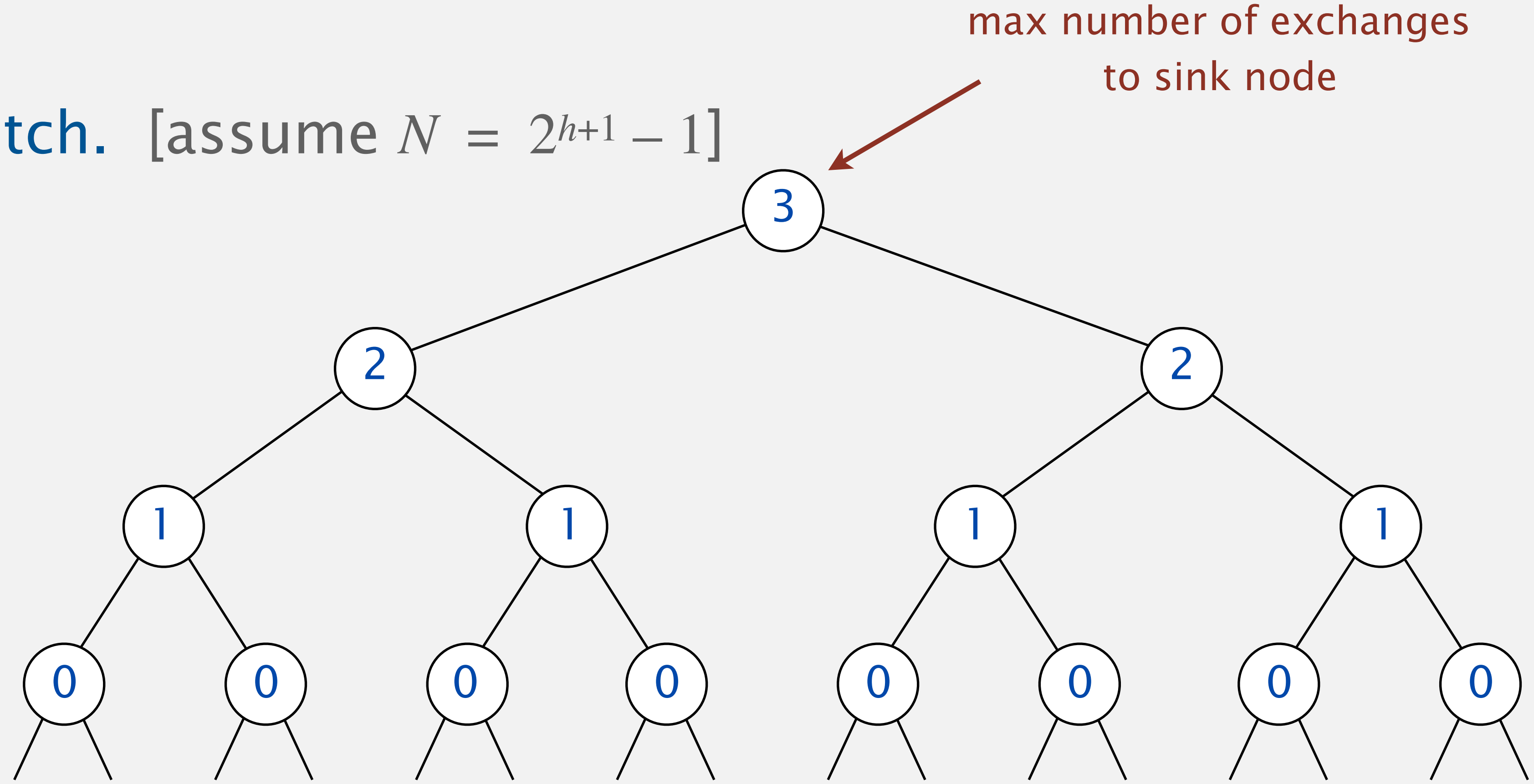
▬ in order

▭ not in order

Heapsort: mathematical analysis

Proposition. Sink-based Heap construction uses $\leq 2N$ compares and $\leq N$ exchanges.

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



binary heap of height h = 3

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

$$\sum_{i=0}^{h} i * 2^i = (h-1)2^{h+1} + 2$$

Heapsort: mathematical analysis

Proposition. Sink-based Heap construction uses $\le 2N$ compares and $\le N$ exchanges.

Proposition. Heapsort uses $\le 2N \lg N$ compares and exchanges.

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:

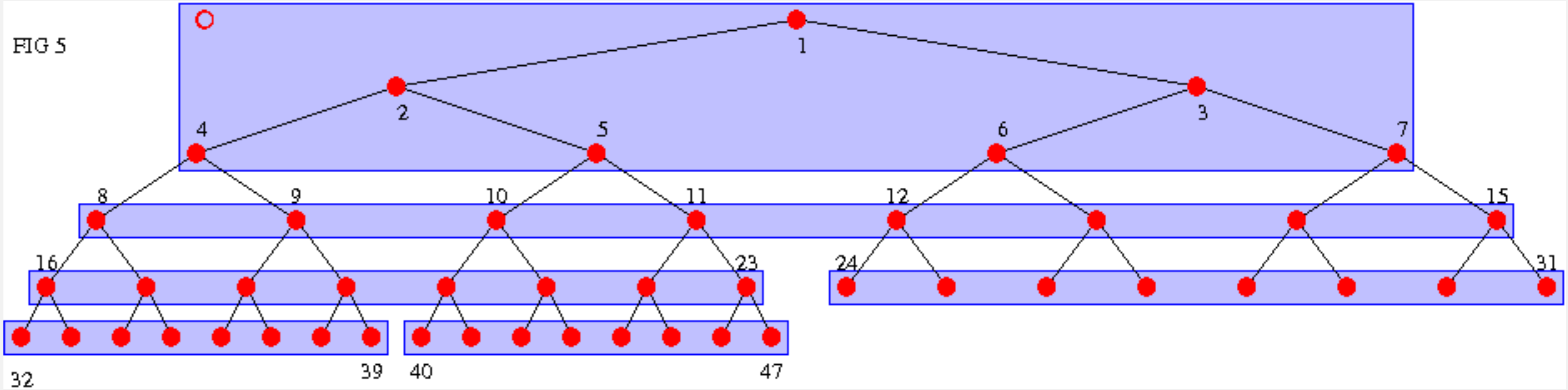
- Inner loop longer than quicksort's.
- Makes poor use of cache.
- Not stable.

↖ advanced tricks for improving

Binary heap problem

Caching. Binary heap is not cache friendly.

FIG 5



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
shell	✔		$N \log_3 N$	?	$c N^{3/2}$	tight code; subquadratic
merge		✔	$\frac{1}{2} N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee; stable
timsort		✔	N	$N \lg N$	$N \lg N$	improves mergesort when preexisting order
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	✔		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

Homework Assignment # 5: MinValueSearch Implementation

```
public abstract class MinValueSearch
{
   public abstract int Min(int[] array);
}
```

5	6	7	8	9	10	11	12	13	3	4

No delay is allowed. Submit your Java code

Deadline is 5/8 pm23:59