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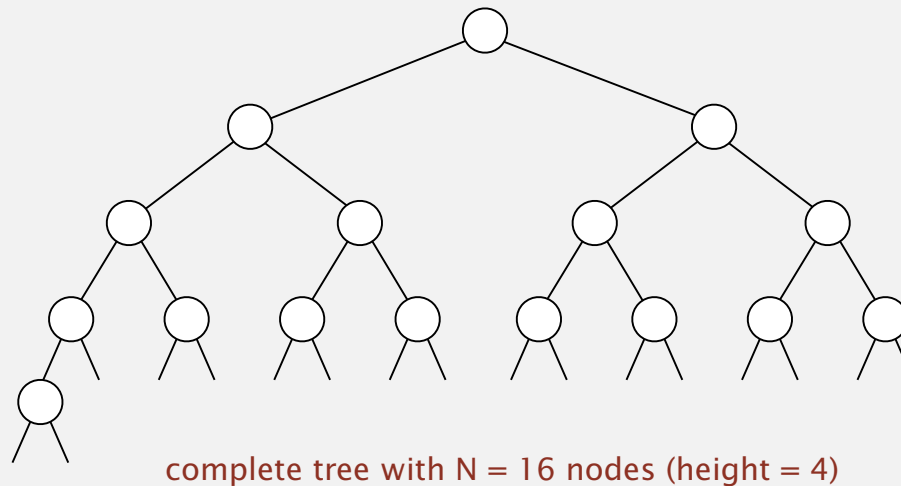
Binary tree. Empty or node with links to left and right binary trees.

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 only increases when N is a power of 2.

A complete binary tree in nature

Hyphaene Compressa - Doum Palm

© Shlomit Pinter
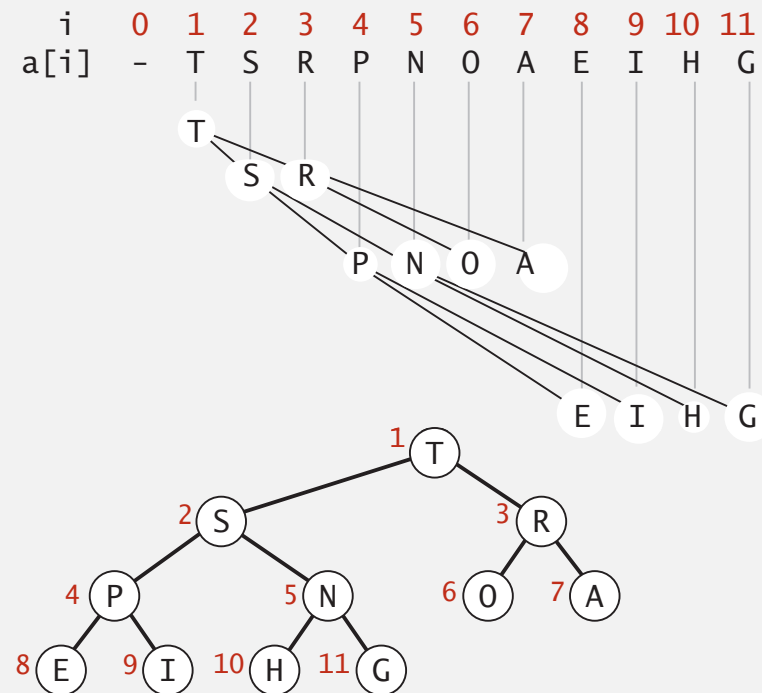
Binary heap representations

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.
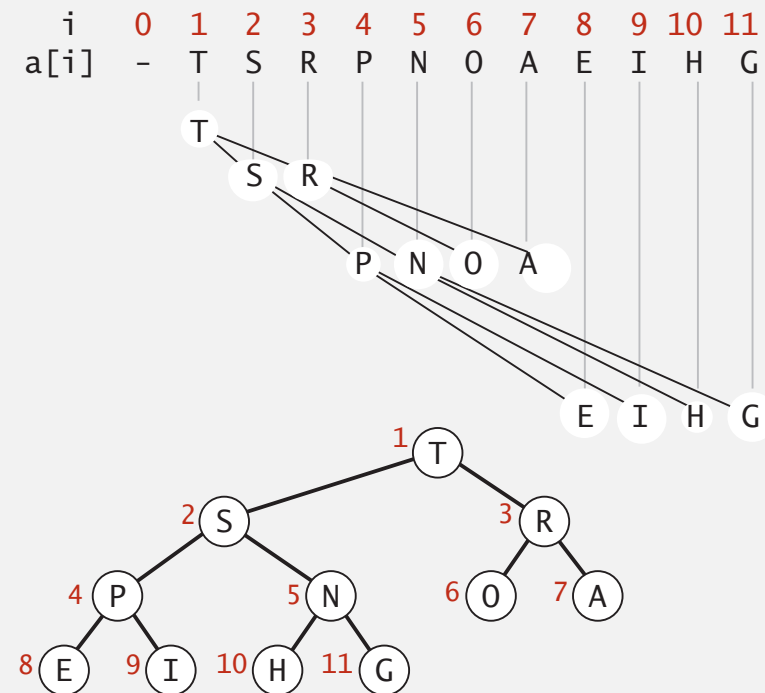- Take nodes in level order.
- No explicit links needed!

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

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`.

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

T
S R
P N O A
E I H G

1 T
2 S   3 R
4 P   5 N   6 O   7 A
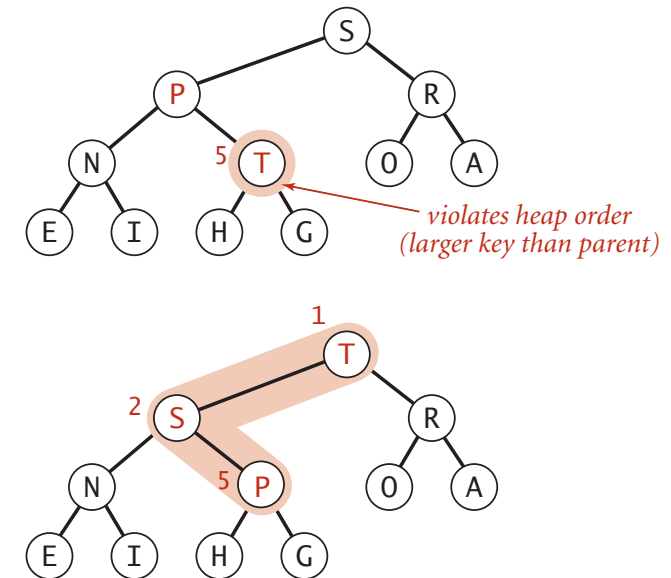8 E 9 I 10 H 11 G

Heap representations

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;
    }
}
```
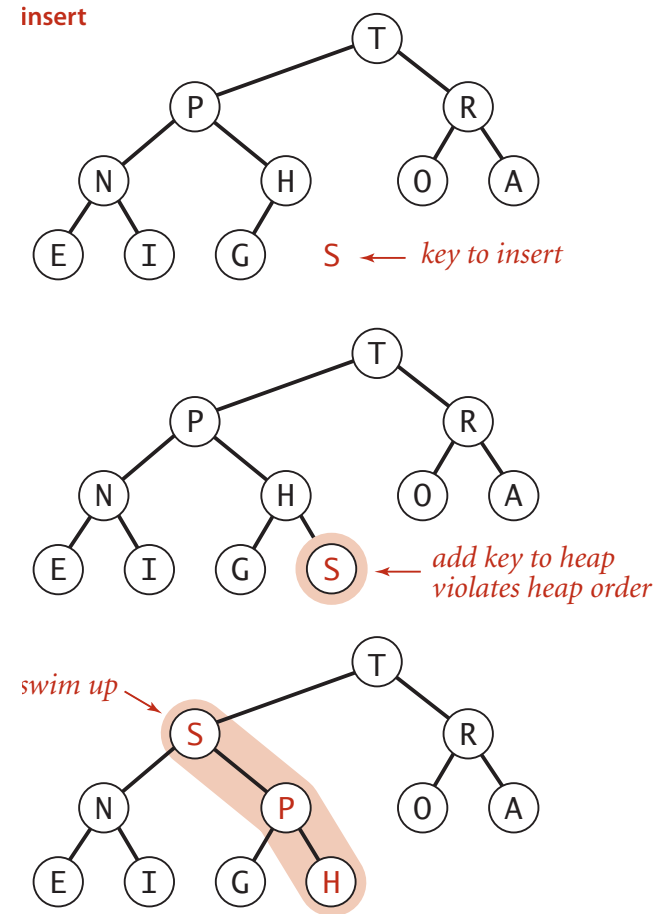
parent of node at k is at k/2

violates heap order
(larger key than parent)

Peter principle. Node promoted to level of incompetence.

Insertion in a heap

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

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

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

Demotion in a heap

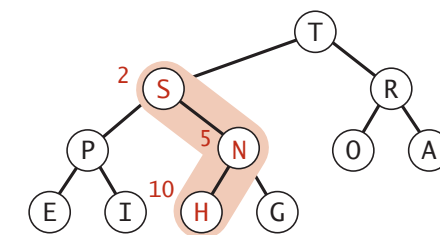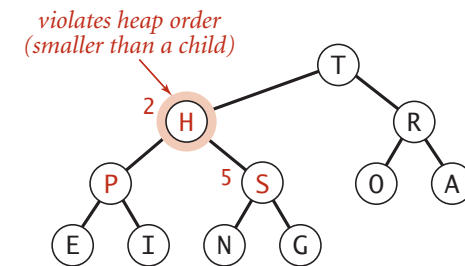**Scenario.** Parent's key becomes smaller than one (or both) of its children's.

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

*violates heap order
(smaller than a child)*

Top-down reheapify (sink)

Power struggle. Better subordinate promoted.

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.

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

prevent loitering