Queue

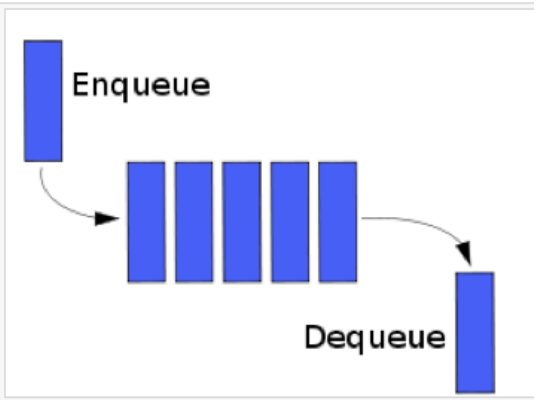
Stack

Queue (data structure)

From Wikipedia, the free encyclopedia

This article is about queuing structures in computing. For queues in general, see queue.

A **queue** (pronounced /kjuː/) is a particular kind of collection in which the entities in the collection are kept in order and the principal (or only) operations on the collection are the addition of entities to the rear terminal position and removal of entities from the front terminal position. This makes the queue a First-In-First-Out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that whenever an element is added, all elements that were added before have to be removed before the new element can



Representation of a FIFO Queue

Queue

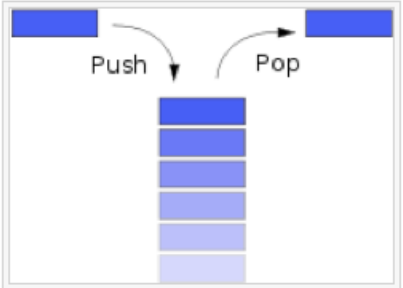
Stack

Stack (data structure)

From Wikipedia, the free encyclopedia

"Pushdown" redirects here. For the strength training exercise, see pushdown (exercise).

In computer science, a **stack** is a last in, first out (LIFO) abstract data type and data structure. A stack can have any abstract data type as an element, but is characterized by only two fundamental operations: *push* and *pop*. The push operation adds to the top of the list, hiding any items already on the stack, or initializing the stack if it is empty. The pop operation removes an item from the top of the list, and returns this value to the caller. A pop either reveals previously concealed items, or results in an empty list.



Simple representation of a stack

A stack is a *restricted data structure*, because only a small number of operations are performed on it. The nature of the pop and push operations also means that stack elements have a natural order. Elements are removed from the stack in the reverse order to the order of their addition: therefore, the lower elements are typically those that have been in the list the longest.

Queue

Stack

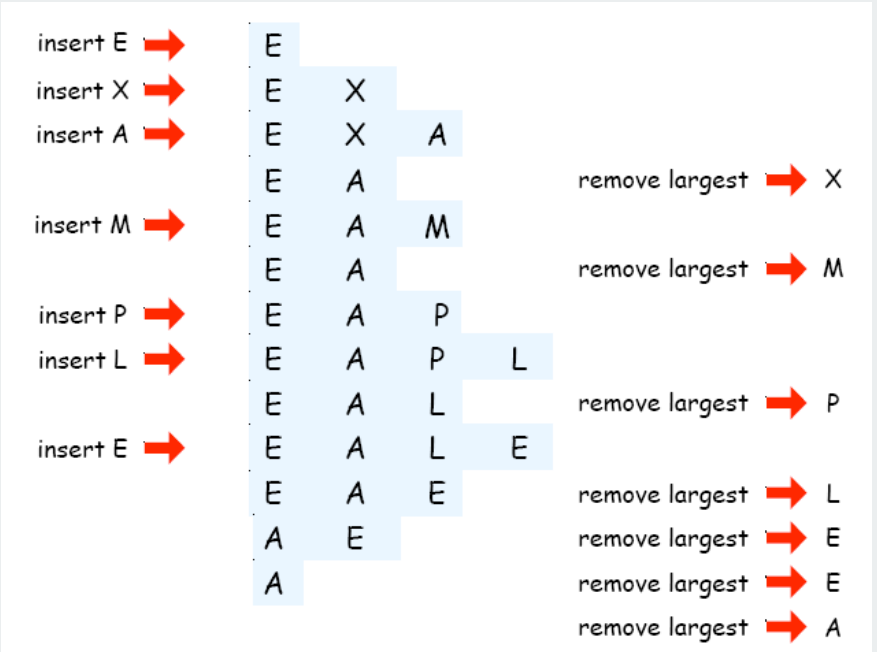
Priority Queues

Data. Items that can be compared.

Basic operations.

- Insert.
- Remove largest. defining ops

- Copy.
- Create.
- Destroy. generic ops
- Test if empty.

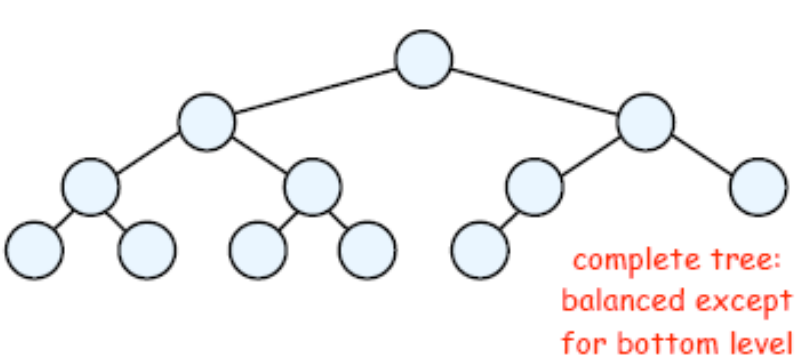


Binary Heap

Heap: Array representation of a heap-ordered complete binary tree.

Binary tree.

- Empty or
- Node with links to left and right trees.



Binary Heap

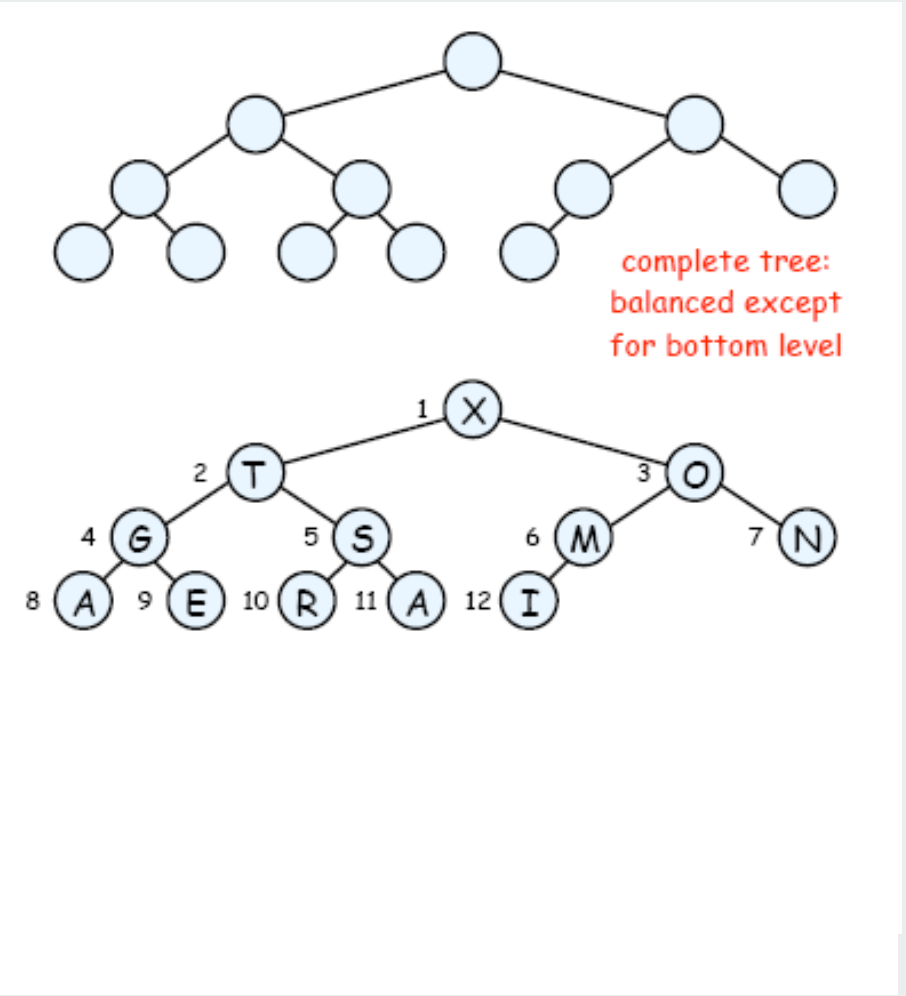
Heap: Array representation of a heap-ordered complete binary tree.

Binary tree.

- Empty or
- Node with links to left and right trees.

Heap-ordered binary tree.

- Keys in nodes.
- No smaller than children's keys.



Binary Heap

Heap: Array representation of a heap-ordered complete binary tree.

Binary tree.

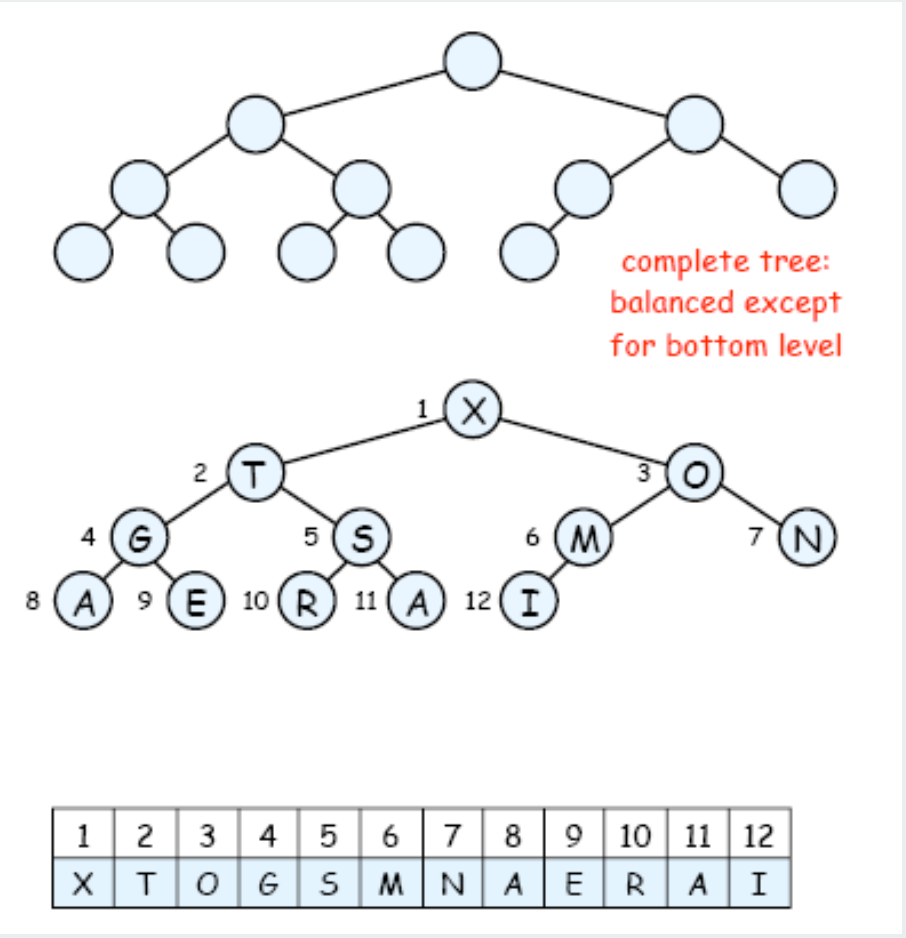
- Empty or
- Node with links to left and right trees.

Heap-ordered binary tree.

- Keys in nodes.
- No smaller than children's keys.

Array representation.

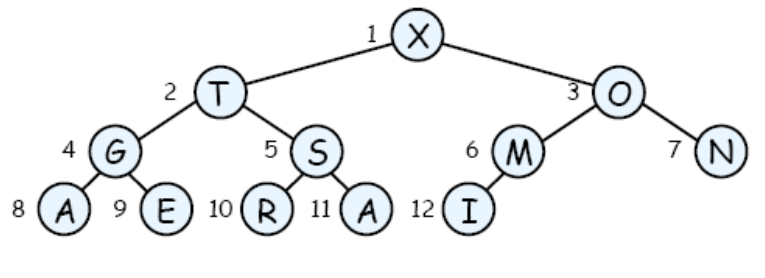
- Take nodes in level order.
- No explicit links needed since tree is complete.



1	2	3	4	5	6	7	8	9	10	11	12
X	T	O	G	S	M	N	A	E	R	A	I

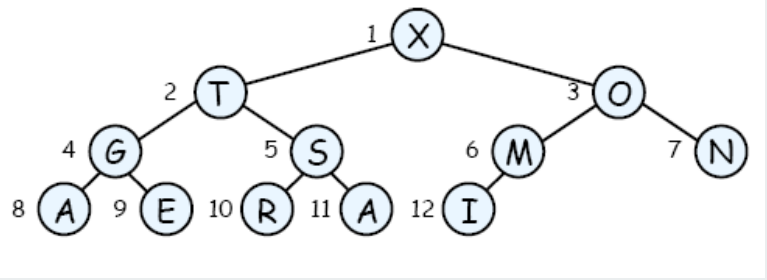
Binary Heap Properties

Property A. Largest key is at root.



Binary Heap Properties

Property A. Largest key is at root.



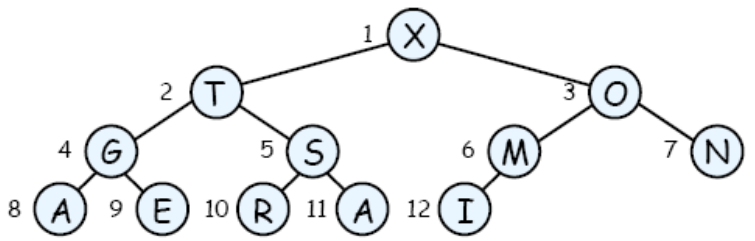
Property B. Can use array indices to move through tree.

- Note: indices start at 1.
- Parent of node at k is at k/2.
- Children of node at k are at 2k and 2k+1.

1	2	3	4	5	6	7	8	9	10	11	12
X	T	O	G	S	M	N	A	E	R	A	I

Binary Heap Properties

Property A. Largest key is at root.



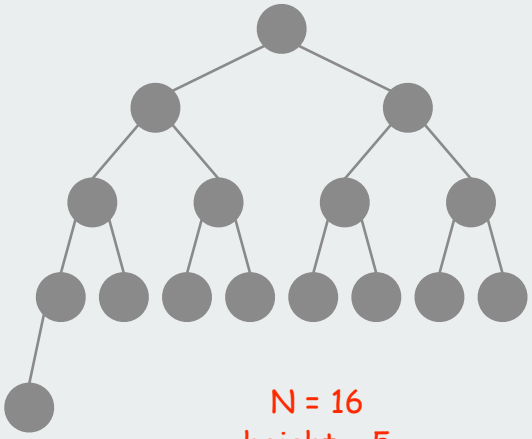
Property B. Can use array indices to move through tree.

- Note: indices start at 1.
- Parent of node at k is at k/2.
- Children of node at k are at 2k and 2k+1.

1	2	3	4	5	6	7	8	9	10	11	12
X	T	O	G	S	M	N	A	E	R	A	I

Property C. Height of N node heap is 1 + ⌊lg N⌋.

height increases only when
N is a power of 2

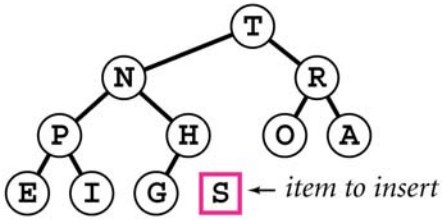


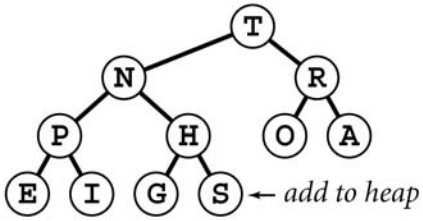
N = 16
height = 5

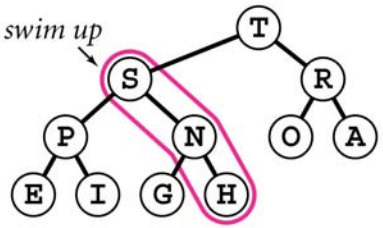
Insert

Insert. Add node at end, then promote.

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







Demotion In a Heap

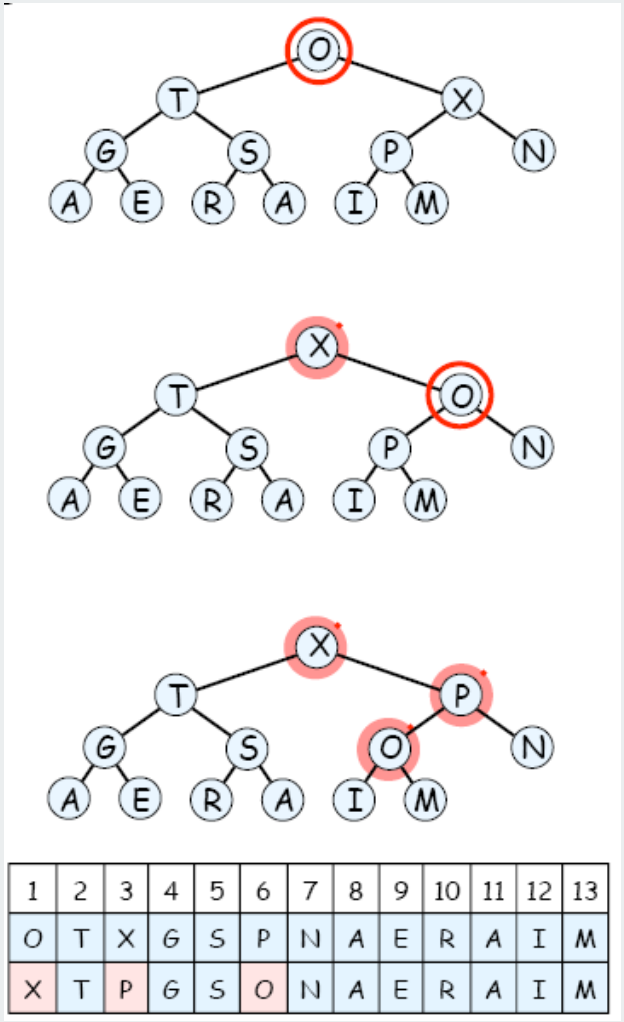
Scenario. Exactly one node has a smaller key than does a child.

To eliminate the violation:

- Exchange with 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



1	2	3	4	5	6	7	8	9	10	11	12	13
O	T	X	G	S	P	N	A	E	R	A	I	M
X	T	P	G	S	O	N	A	E	R	A	I	M

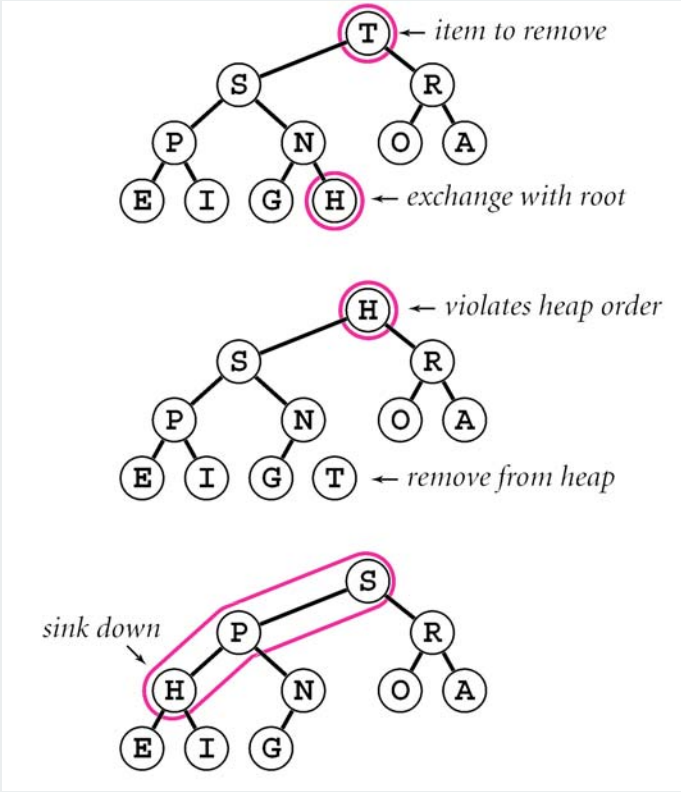
Power struggle: better subordinate promoted.

Remove the Maximum

Remove max. Exchange root with node at end, then demote.

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

prevent loitering



Priority Queues Implementation Cost Summary

Operation	Insert	Remove Max	Find Max
ordered array	N	1	1
ordered list	N	1	1
unordered array	1	N	N
unordered list	1	N	N
binary heap	lg N	lg N	1

worst-case asymptotic costs for PQ with N items

Hopeless challenge. Make all ops O(1).

Why hopeless?