CSE 326: Data Structures
Lecture #4
Heaps more Priority Qs

Bart Niswonger

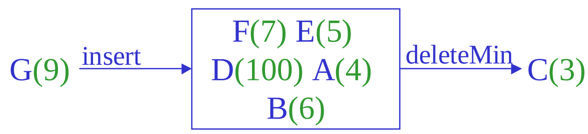
Summer Quarter 2001

Today's Outline

- Return quizzes
- Things Bart Didn't Finish on Friday (insert & d-Heaps)
- Leftist Heaps
- Skew Heaps
- Comparing Heaps

Priority Queue ADT

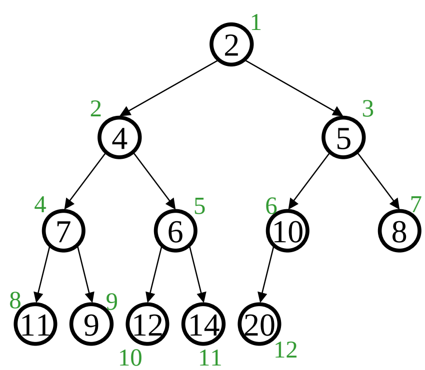
- Priority Queue operations
  - create
  - destroy
  - insert
  - deleteMin
  - is_empty



- Priority Queue property: for two elements in the queue, *x* and *y*, if *x* has a lower priority value than *y*, *x* will be deleted before *y*

Nifty Storage Trick

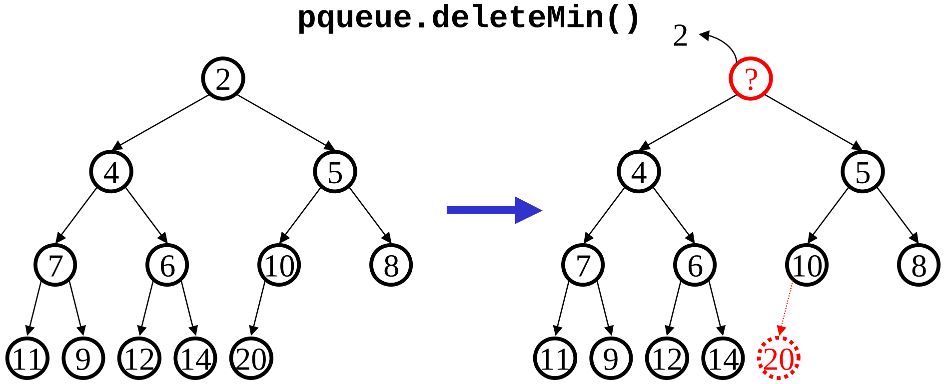
- Calculations:
  - child:
  - parent:
  - root:
  - next free:



0	1	2	3	4	5	6	7	8	9	10	11	12	
12	2	4	5	7	6	10	8	11	9	12	14	20	

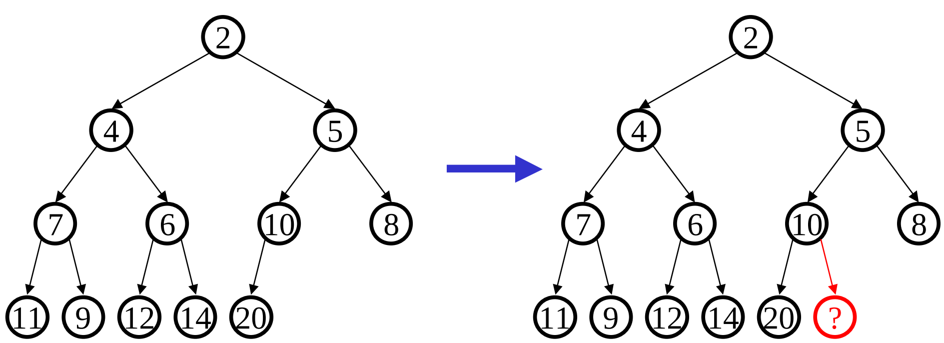
DeleteMin

`pqueue.deleteMin()`

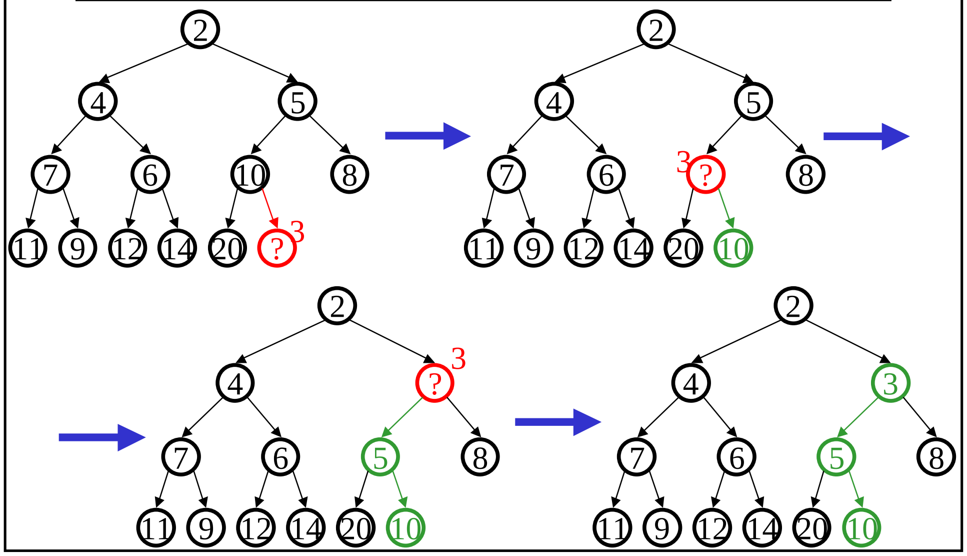


Insert

`pqueue.insert(3)`



Percolate Up



Insert Code

```
void insert(Object o) {
  assert(!isFull());
  size++;
  newPos =
    percolateUp(size,o);
  Heap[newPos] = o;
}
```

```
int percolateUp(int hole,
                Object val) {
  while (hole > 1 &&
         val < Heap[hole/2])
    Heap[hole] = Heap[hole/2];
    hole /= 2;
  }
  return hole;
}
```

runtime:

Other Priority Queue Operations

- decreaseKey
  - given the position of an object in the queue, reduce its priority value
- increaseKey
  - given the position of an an object in the queue, increase its priority value
- remove
  - given the position of an object in the queue, remove it
- buildHeap
  - given a set of items, build a heap

DecreaseKey, IncreaseKey, and Remove

```
void decreaseKey(int obj) {
  assert(size >= obj);
  temp = Heap[obj];
  newPos = percolateUp(obj, temp);
  Heap[newPos] = temp;
}

void increaseKey(int obj) {
  assert(size >= obj);
  temp = Heap[obj];
  newPos = percolateDown(obj, temp);
  Heap[newPos] = temp;
}
```

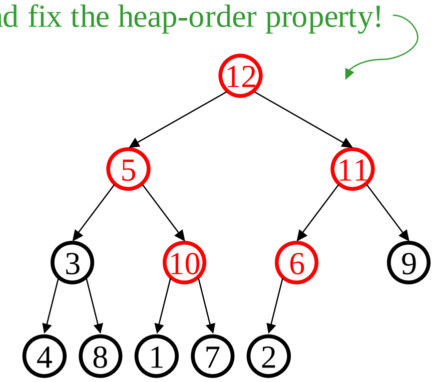
```
void remove(int obj) {
  assert(size >= obj);
  percolateUp(obj,
         NEG_INF_VAL);
  deleteMin();
}
```

BuildHeap

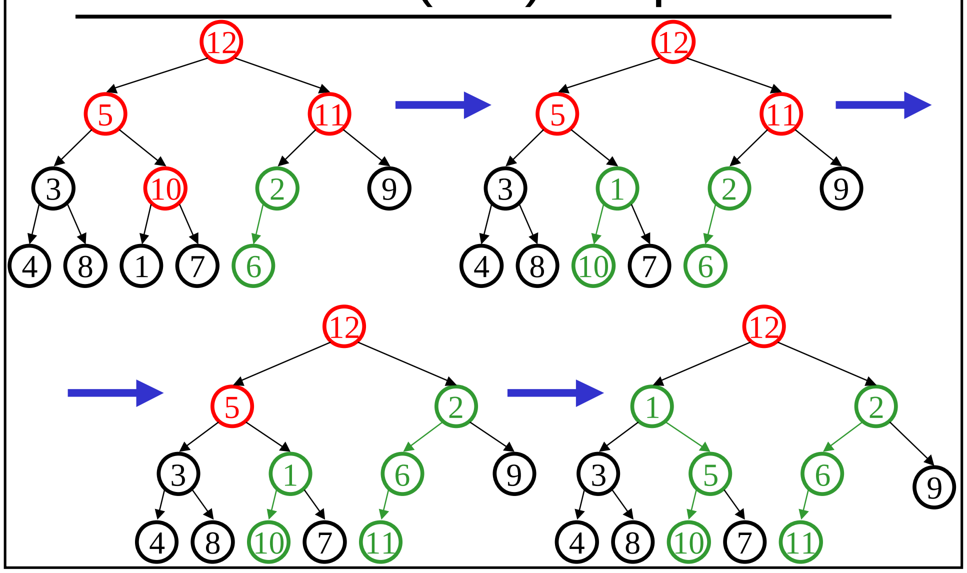
Floyd's Method. Thank you, Floyd.

12	5	11	3	10	6	9	4	8	1	7	2

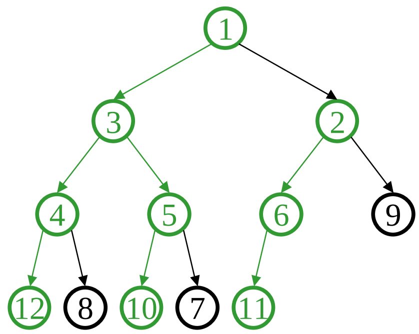
pretend it's a heap and fix the heap-order property!



Build(this)Heap



Finally…



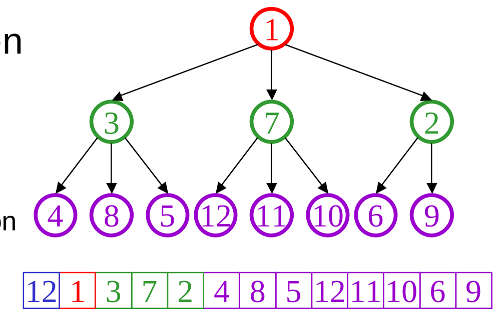
runtime:

Thinking about Heaps

- Observations
  - finding a child/parent index is a multiply/divide by two
  - operations jump widely through the heap
  - each operation looks at only two new nodes
  - inserts are at least as common as deleteMins
- Realities
  - division and multiplication by powers of two are **fast**
  - looking at one new piece of data sucks in a cache line
  - with **huge** data sets, disk accesses dominate

Solution: d-Heaps

- Each node has *d* children
- Still representable by array
- Good choices for *d*:
  - optimize performance based on # of inserts/removes
  - choose a power of two for efficiency
  - fit one set of children in a cache line
  - fit one set of children on a memory page/disk block



One More Operation

- Merge two heaps. Ideas?

Merge

Given two heaps, merge them into one heap

– first attempt: insert each element of the smaller heap into the larger.

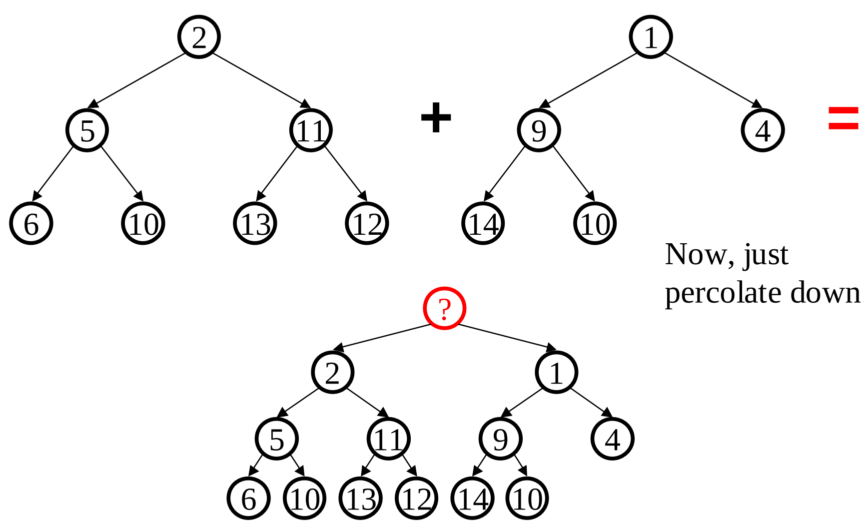
runtime:

– second attempt: concatenate heaps' arrays and run buildHeap.

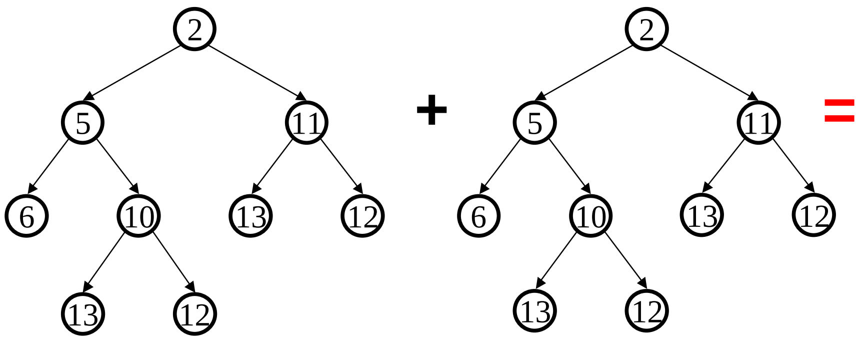
runtime:

How about O(log n) time?

Idea: Hang a New Tree



Idea: Hang a New Tree



Problem?

Leftist Heaps

- Idea:

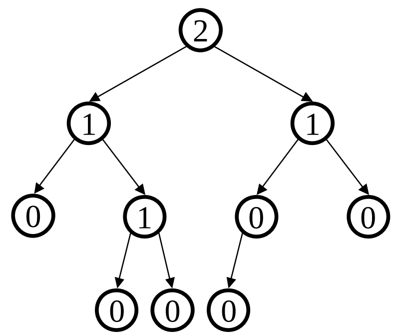
  make it so that all the work you have to do in maintaining a heap is in one small part

- Leftist heap:
  - almost all nodes are on the left
  - all the merging work is on the right

Random Definition: Null Path Length

the *null path length (npl)* of a node is the number of nodes between it and a null in the tree

- npl(null) = -1
- npl(leaf) = 0
- npl(single-child node) = 0



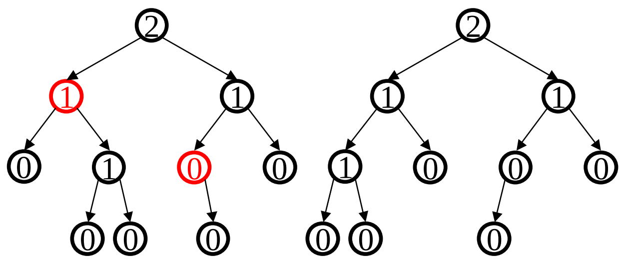
another way of looking at it: npl is the height of complete subtree rooted at this node

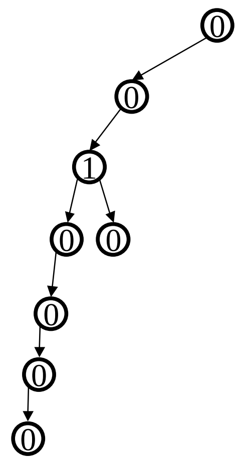
Leftist Heap Properties

- Heap-order property
  - parent's priority value is ≤ to childrens' priority values
  - result: minimum element is at the root
- Leftist property
  - null path length of left subtree is ≥ npl of right subtree
  - result: tree is at least as "heavy" on the left as the right

Are leftist trees complete? Balanced?

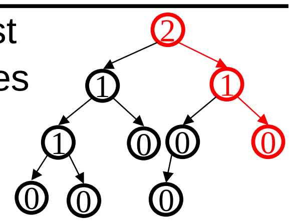
Leftist tree examples





every subtree of a leftist tree is leftist, comrade!

Right Path in a Leftist Tree is Short



- If the right path has length at least **r**, the tree has at least $2^r - 1$ nodes
- Proof by induction

  Basis: $r = 1$. Tree has at least one node: $2^1 - 1 = 1$

  Inductive step: assume true for $r' < r$. The right subtree has a right path of at least r - 1 nodes, so it has at least $2^{r-1} - 1$ nodes. The left subtree must also have a right path of at least $r - 1$ (otherwise, there is a null path of $r - 3$, less than the right subtree). Again, the left has $2^{r-1} - 1$ nodes. All told then, there are at least:

  $$2^{r-1} - 1 + 2^{r-1} - 1 + 1 = 2^r - 1$$

- So, a leftist tree with at least **n** nodes has a right path of at most **log n** nodes

Whew!

To Do

- Unix development Tutorial
  - Tuesday – 10:50am – Sieg 322
- Finish Project I for Wednesday
- Read chapters 1 & 2

Coming Up

- Theory!
- Proof by Induction
- Asymptotic Analysis
- Quiz #2 (Thursday)