# Heaps ‘o’ Fun

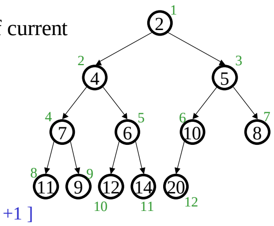
A watered down version of a Winter 2002 lecture by

Nick Deibel

## Nifty Storage Trick

- **Calculations:**

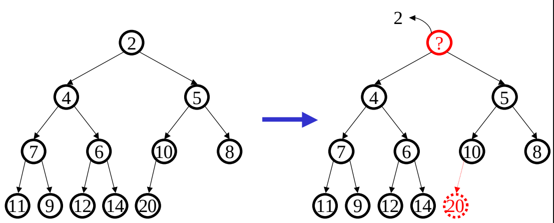
  For a node i in a heap of current capacity m
  - child: array[ 2i ] and array[ 2i+1 ]
  - parent: array[ ⌊i/2⌋ ]
  - root: array[0]
  - next free: array[ array[0] +1 ]



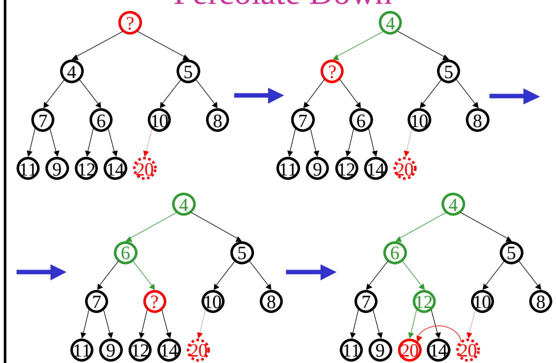
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12 | 2 | 4 | 5 | 7 | 6 | 10 | 8 | 11 | 9 | 12 | 14 | 20 | |

## DeleteMin

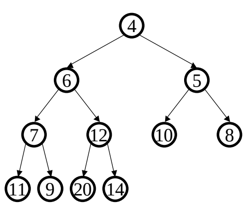
```
pqueue.deleteMin()
```



## Percolate Down



## Finally…



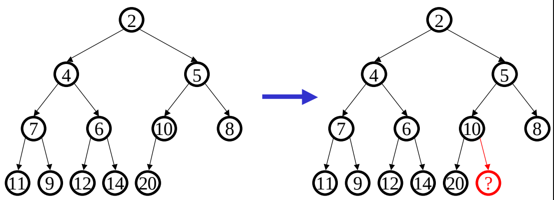
## DeleteMin Code

```
Object deleteMin() {
  assert(!isEmpty());
  returnVal = Heap[1];
  size--;
  newPos =
    percolateDown(1,
        Heap[size+1]);
  Heap[newPos] =
    Heap[size + 1];
  return returnVal;
}
```

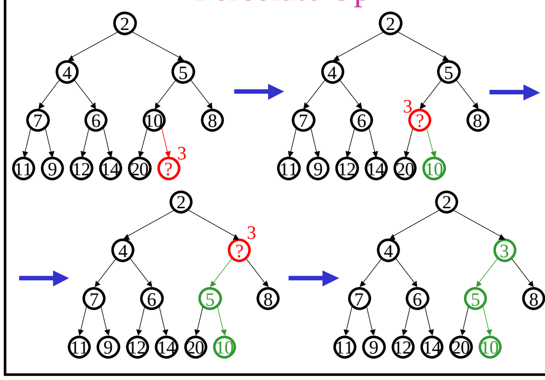
```
int percolateDown(int hole,
                  Object val) {
while (2*hole <= size) {
    left = 2*hole;
    right = left + 1;
    if (right <= size &&
        Heap[right] < Heap[left])
      target = right;
    else
      target = left;

    if (Heap[target] < val) {
      Heap[hole] = Heap[target];
      hole = target;
    }
    else
      break;
  }
  return hole;
}
```

## Insert

`pqueue.insert(3)`



## Percolate Up



## Insert Code

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

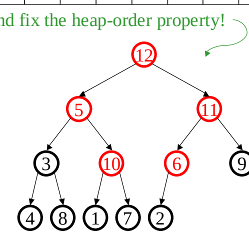
```
int percolateUp(int hole,
                Object val) {
  while (hole > 1 &&
         val < Heap[hole/2])
    Heap[hole] = Heap[hole/2];
    hole /= 2;
  }
  return hole;
}
```

## BuildHeap

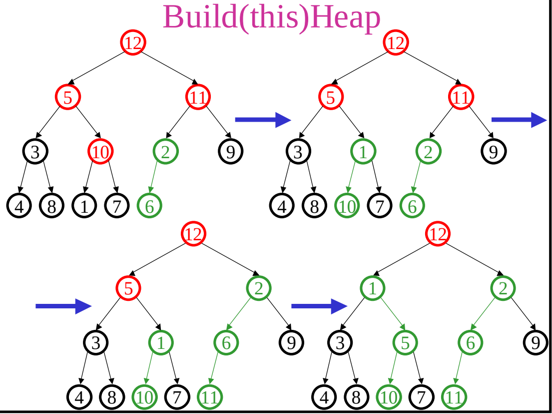
Floyd's Method. Thank you, Floyd.

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

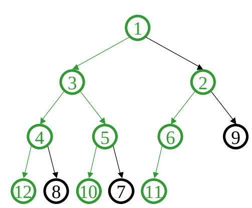
pretend it's a heap and fix the heap-order property!



## Build(this)Heap



## Finally…



## Complexity of Build Heap

- Note: size of a perfect binary tree doubles (+1) with each additional layer
- At most n/4 percolate down 1 level
  at most n/8 percolate down 2 levels
  at most n/16 percolate down 3 levels…

$$1\cdot\frac{n}{4}+2\cdot\frac{n}{8}+3\cdot\frac{n}{16}+\cdots=\sum_{i=1}^{\log n} i\cdot\frac{n}{2^{i+1}}$$

$$=\frac{n}{2}\sum_{i=1}^{\log n}\frac{i}{2^i}\le\frac{n}{2}(2)=n = O(n)$$

## Proof of Summation

$$S=\sum_{i=1}^{x}\frac{i}{2^i}=\frac{1}{2}+\frac{2}{4}+\frac{3}{8}+\cdots+\frac{x-1}{2^{x-1}}+\frac{x}{2^x}$$

$$2S=1+\frac{2}{2}+\frac{3}{4}+\cdots+\frac{x}{2^{x-1}}$$

$$S=2S-S=1+\left(\frac{2}{2}-\frac{1}{2}\right)+\left(\frac{3}{4}-\frac{2}{4}\right)+\cdots+\left(-\frac{x}{2^x}\right)$$

$$S\le 1+\sum_{i=1}^{x-1}\frac{1}{2^i}\le 1+1=2$$

## Heap Sort

- Input: unordered array A[1..N]
  1. Build a max heap (largest element is A[1])
  2. For i = 1 to N-1:
     A[N-i+1] = Delete_Max()

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 7 | 50 | 22 | 15 | 4 | 40 | 20 | 10 | 35 | 25 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 50 | 40 | 20 | 25 | 35 | 15 | 10 | 22 | 4 | 7 |
| 40 | 35 | 20 | 25 | 7 | 15 | 10 | 22 | 4 | 50 |
| 35 | 25 | 20 | 22 | 7 | 15 | 10 | 4 | 40 | 50 |

## Properties of Heap Sort

- Worst case time complexity O(n log n)
  - Build_heap O(n)
  - n Delete_Max's for O(n log n)
- In-place sort – only constant storage beyond the array is needed