Trees II

BSTs, Heaps, and PQs

Announcements

- DNA – Due October 23
- APT Set 5 – Due October 28

Today

- Trees
  - The importance of balanced trees
  - Traversals
  - Heaps
    - Priority Queues

BST

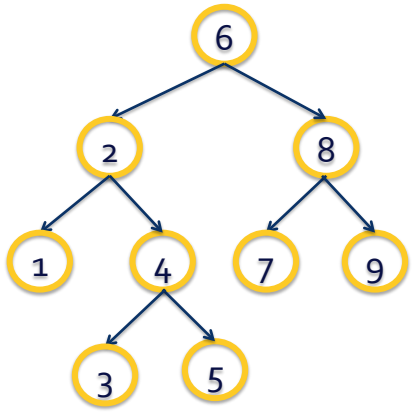
The following nodes are added to a binary search tree (BST) in order. Draw the resulting BST.

6,8,2,4,1,7,5,3,9

```
public void add(int newValue){
        if(root == null)
                root = new TreeNode(newValue);
        else
                add(newValue, root);
}
public void add(int newValue, TreeNode current) {
        if (newValue < current.myValue) {
                if (current.myLeft == null)
                        current.myLeft = new TreeNode(newValue);
                else
                        add(newValue, current.myLeft);
        } else
                if (current.myRight == null)
                        current.myRight = new TreeNode(newValue);
                else
                        add(newValue, current.myRight);
        }
}
```

BST

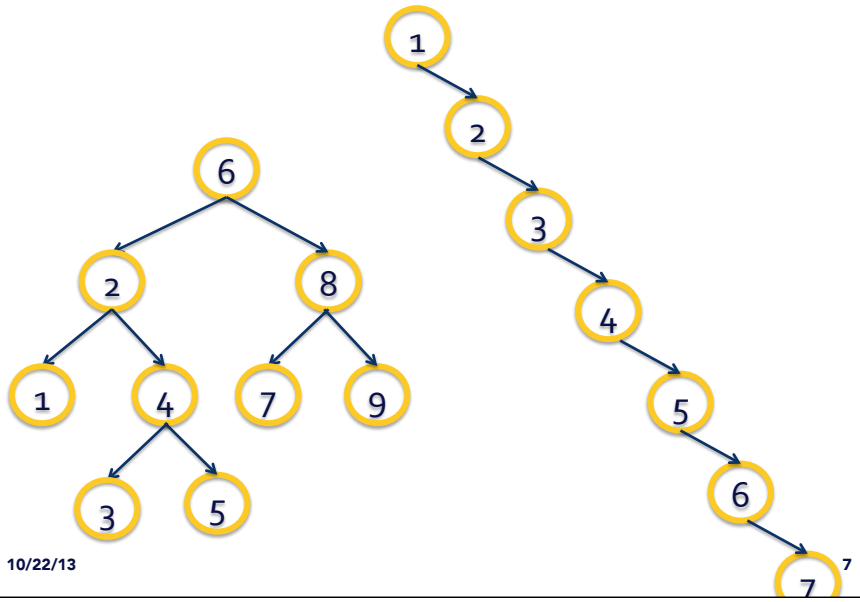
- My answer
  - 6,8,2,4,1,7,5,3,9



BST

- The following nodes are added to a binary search tree (BST) in order. Draw the resulting BST.
  - 1,2,3,4,5,6,7,8,9

BST

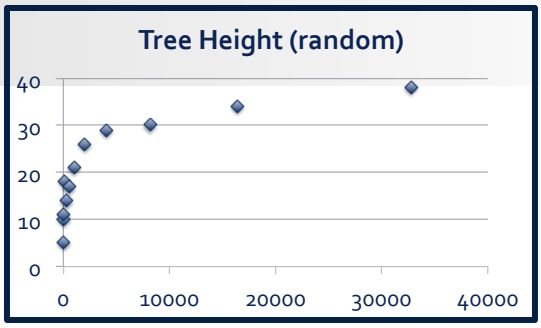


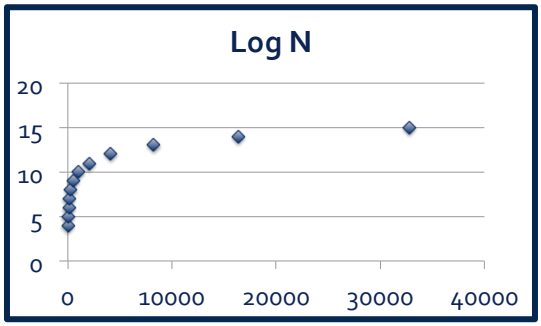
Code

Open TreeNodeExample.java
Change main to:

```
1       public static void main(String[] args) {
2           for(int j = 4; j < 15; j++){
3               TreeNodeExample tree = new TreeNodeExample();
4               double start = System.currentTimeMillis();
5               int nodes = (int)Math.pow(2,j);
6               for(int i = 0; i < nodes; i++)
7                   tree.add(i);
8
9               double end = System.currentTimeMillis();
10              double time = (end-start)/1000.0;
11              System.out.printf("Time: %f Height: %d Nodes: %d
    (2^%d)\n", time, tree.computeHeight(), nodes, j);
12          }
13  }
```

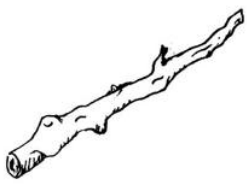
BST





BST

- Order matters!
  - random input ➔ height O(logN)
  - ordered input ➔ height O(N)

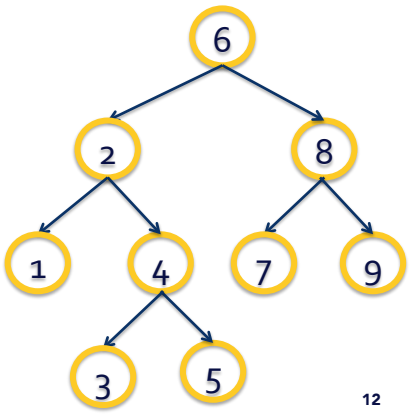


Today

- Trees
  - The importance of balanced trees
  - Traversals
  - Heaps
    - Priority Queues

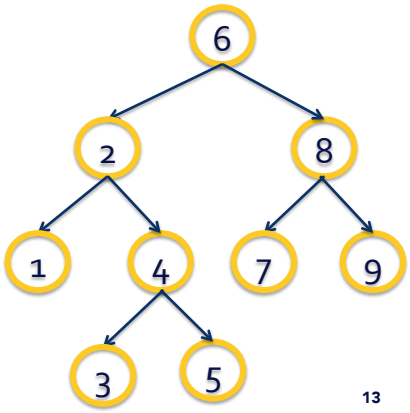
Tree traversals

- Given a BST, how would you print the nodes In Order?



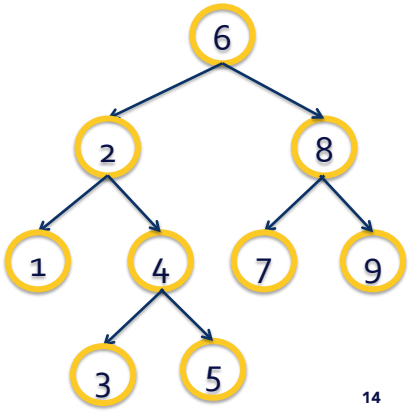
Tree traversals

- Given a BST, how would you print the nodes In Order?
  - go left
  - current
  - go right



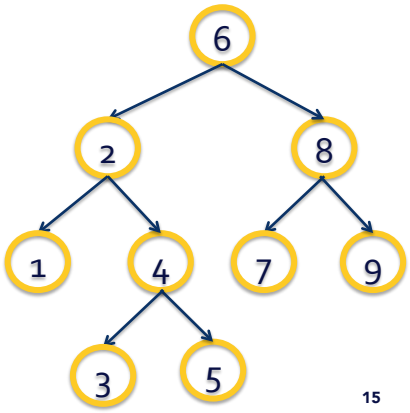
Tree traversals

- Given a BST, how would you print the nodes in Pre Order?
  - go left
  - current
  - go right



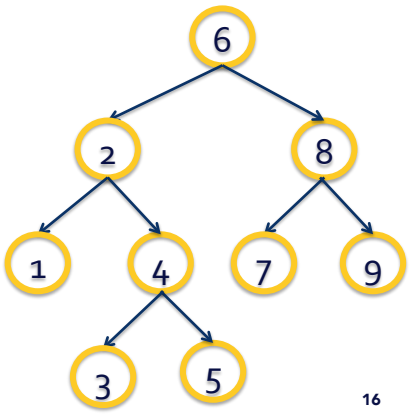
Tree traversals

- Given a BST, how would you print the nodes in Pre Order?
  - current
  - go left
  - go right
- Duplicate a tree



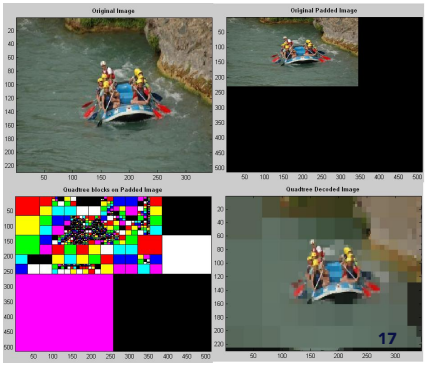
Tree traversals

- Given a BST, how would you print the nodes in Post Order?
  - go left
  - go right
  - current
- Delete a tree



Trees

- Applications
  - Computer graphics
  - Database
  - File storage on your computer
  - Internet protocols



Today

- Trees
  - The importance of balanced trees
  - Traversals
  - Heaps
    - Priority Queues

Queues and Stacks

```
1  public static void main(String[] args) {
2          Queue aQueue = new LinkedList();
3       Stack aStack = new Stack();
4       String[] wordsToAdd = {"compsci", "201", "is", "great"};
5
6       for(String s: wordsToAdd){
7           aQueue.add(s); //enqueue
8           aStack.push(s);
9       }
10                                              compsci 201 is great
11      while(!aQueue.isEmpty())
12          System.out.print(aQueue.remove() + " "); //dequeue
13
14      System.out.println();
15
16      while(!aStack.isEmpty())                great is 201 compsci
17          System.out.print(aStack.pop() + " ");
18  } compsci 201 is great
```


Priority Queue

- Airport queue
  - First class?

Priority Queue

```
1   public static void main(String[] args) {
2
3           PriorityQueue<String> aQueue = new
    PriorityQueue<String>();

4          String[] wordsToAdd = {"compsci", "201", "is",
    "great"};
5
6          for(String s: wordsToAdd){
7              aQueue.add(s);
8              aStack.push(s);
9          }
10
11         while(!aQueue.isEmpty())
12             System.out.print(aQueue.remove() + " ");
13
14     }
```

1. compsci 201 is great
2. great is compsci 201
3. is great compsci 201
4. 201 compsci great is

Priority Queue

- What is the output?

```
PriorityQueue<Integer> ex = new PriorityQueue<Integer>();
ex.add(2);
ex.add(13);
ex.add(9);
ex.add(75);
ex.add(4);
while(!ex.isEmpty()) {
    System.out.println(ex.remove());
}
```

- Add in any order
- Remove smallest first

Heaps

- Common implementation of priority queues
- A tree-like structure
- Almost completely filled
  - All nodes filled expect last level
- Max-Heap - Descendants have values <= to parent
- Min-Heap - Descendants have values >= to parent

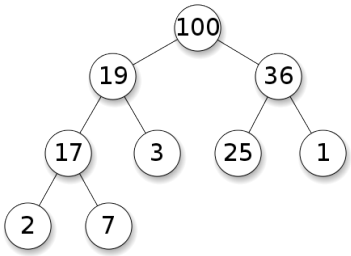
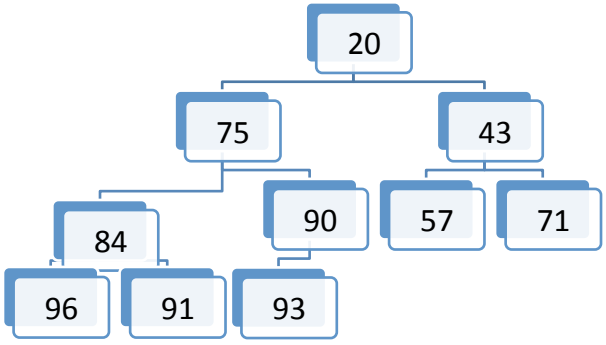


image form wikipedia

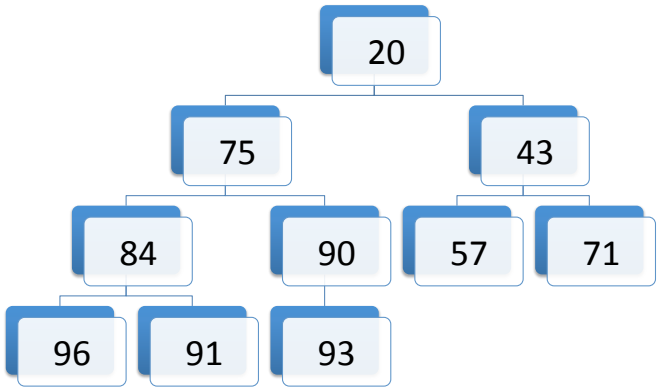
Heaps

- Why is a heap implemented with a priority queue?
  - Where is the min value?



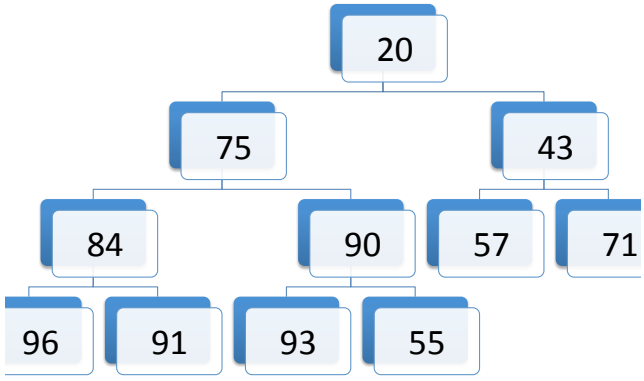
Heaps

- Add 55 to heap
  - add node to first open slot



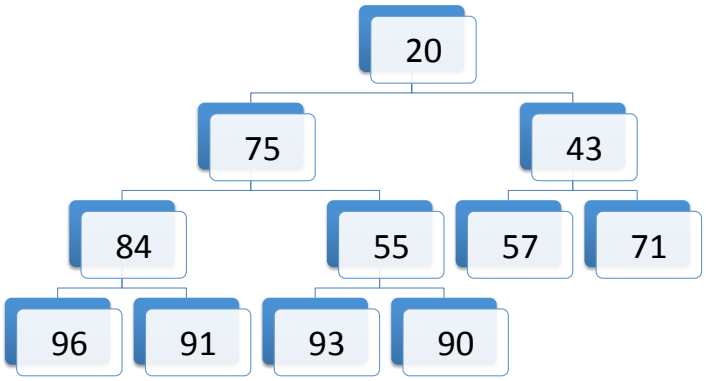
Heaps

- Add 55 to heap
  - If parent is larger, swap



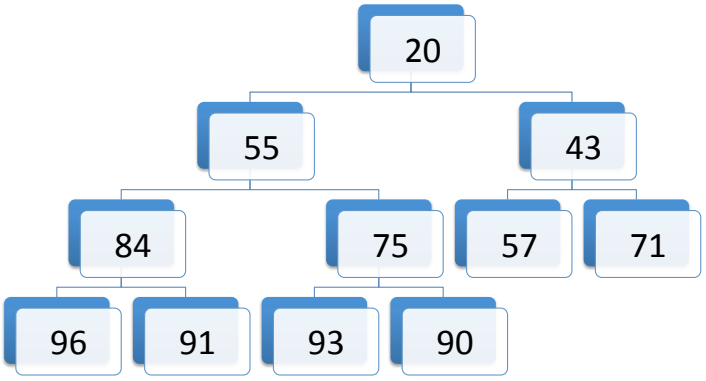
Heaps

- Add 55 to heap
  - If parent is larger, swap

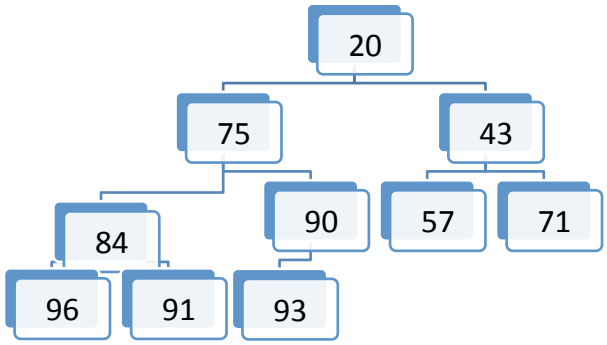


Heaps

- Add 55 to heap
  - If parent is larger, swap



Heaps as Arrays



	20	75	43	84	90	57	71	96	91	93

Layer 1 Layer 2 Layer 3 Layer 4

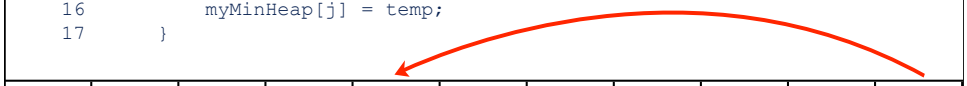
Heaps as Arrays

```
1       public void add(double d){
2           mySize++;
3           myMinHeap[mySize] = d;
4
5           int index = mySize;
6           int parentIndex = index/2;
7           while((myMinHeap[parentIndex] > myMinHeap[index]) &
    parentIndex != 0){
8               swap(index, parentIndex);
9               index = parentIndex;
10              parentIndex = index/2;
11          }
12      }

13      private void swap(int i, int j){
14          double temp = myMinHeap[i];
15          myMinHeap[i] = myMinHeap[j];
16          myMinHeap[j] = temp;
17      }
```

20	75	43	84	90	57	71	96	91	93	55
1	2	3	4	5	6	7	8	9	10	11

Heaps as Arrays

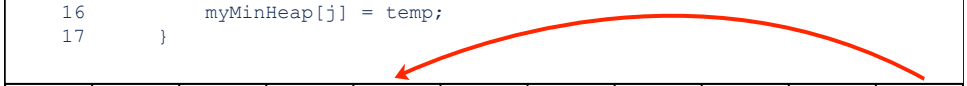
```
1       public void add(double d){
2           mySize++;
3           myMinHeap[mySize] = d;
4
5           int index = mySize;
6           int parentIndex = index/2;
7           while((myMinHeap[parentIndex] > myMinHeap[index]) &
    parentIndex != 0){
8               swap(index, parentIndex);
9               index = parentIndex;
10              parentIndex = index/2;
11          }
12      }

13      private void swap(int i, int j){
14          double temp = myMinHeap[i];
15          myMinHeap[i] = myMinHeap[j];
16          myMinHeap[j] = temp;
17      }
```



20	75	43	84	90	57	71	96	91	93	55
1	2	3	4	5	6	7	8	9	10	11

Heaps as Arrays

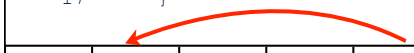
```
1       public void add(double d){
2           mySize++;
3           myMinHeap[mySize] = d;
4
5           int index = mySize;
6           int parentIndex = index/2;
7           while((myMinHeap[parentIndex] > myMinHeap[index]) &
    parentIndex != 0){
8               swap(index, parentIndex);
9               index = parentIndex;
10              parentIndex = index/2;
11          }
12      }

13      private void swap(int i, int j){
14          double temp = myMinHeap[i];
15          myMinHeap[i] = myMinHeap[j];
16          myMinHeap[j] = temp;
17      }
```



20	75	43	84	55	57	71	96	91	93	90
1	2	3	4	5	6	7	8	9	10	11

Heaps as Arrays

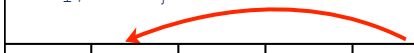
```
1       public void add(double d){
2           mySize++;
3           myMinHeap[mySize] = d;
4
5           int index = mySize;
6           int parentIndex = index/2;
7           while((myMinHeap[parentIndex] > myMinHeap[index]) &
    parentIndex != 0){
8               swap(index, parentIndex);
9               index = parentIndex;
10              parentIndex = index/2;
11          }
12      }

13      private void swap(int i, int j){
14          double temp = myMinHeap[i];
15          myMinHeap[i] = myMinHeap[j];
16          myMinHeap[j] = temp;
17      }
```



20	75	43	84	55	57	71	96	91	93	90
1	2	3	4	5	6	7	8	9	10	11

Heaps as Arrays

```
1       public void add(double d){
2           mySize++;
3           myMinHeap[mySize] = d;
4
5           int index = mySize;
6           int parentIndex = index/2;
7           while((myMinHeap[parentIndex] > myMinHeap[index]) &
    parentIndex != 0){
8               swap(index, parentIndex);
9               index = parentIndex;
10              parentIndex = index/2;
11          }
12      }

13      private void swap(int i, int j){
14          double temp = myMinHeap[i];
15          myMinHeap[i] = myMinHeap[j];
16          myMinHeap[j] = temp;
17      }
```



20	55	43	84	75	57	71	96	91	93	90
1	2	3	4	5	6	7	8	9	10	11

Remove

- Remove the root
- Move last value into root
- If a child is smaller than root
- promote the smallest child
  - What would the array look like if I called remove()?

20	55	43	84	75	57	71	96	91	93	90
1	2	3	4	5	6	7	8	9	10	11

Practice

- Draw the array for the following heap
  - Add the value 27

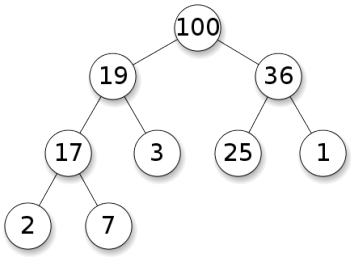
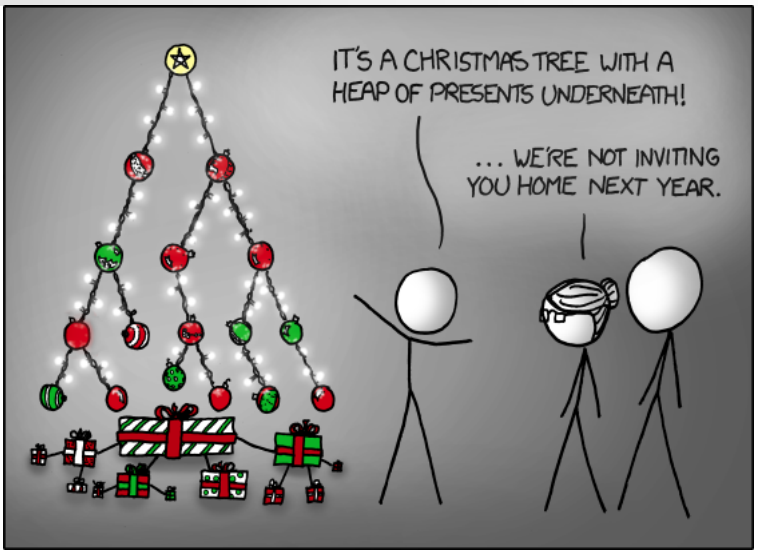


image form wikipedia

Today


IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.