Practice Problems

- Problem 1 - Sorting
- Problem 2 - Heaps
- Problem 3 - Tree traversals
- Problem 4 - Binary Trees
- Problem 5 - Binary Search Trees
- Problem 6 - Iterators

Problem 1 - Sorting

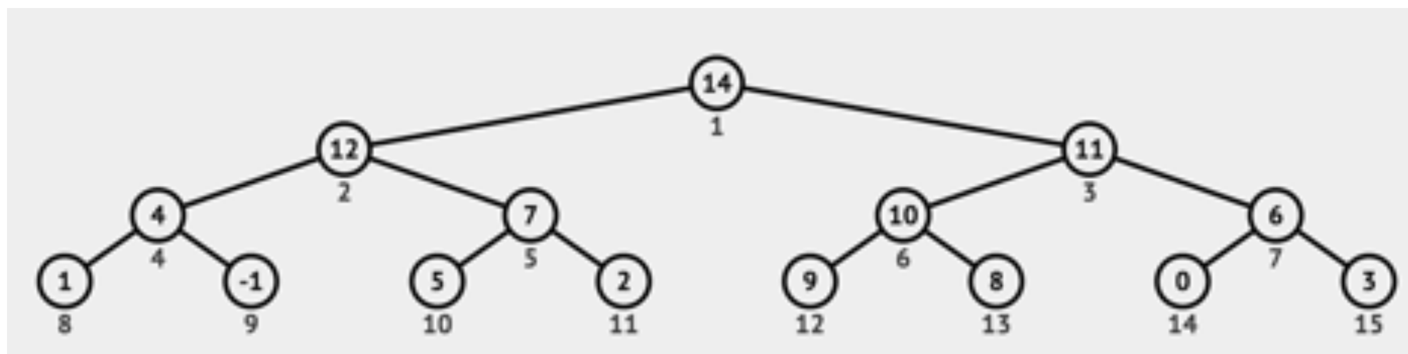
- In the next slide, you can find a table whose first row (last column 0) contains an array of 18 unsorted numbers between 1 and 50. The last row (last column 6) contains the numbers in sorted order. The other rows show the array in some intermediate state during one of these five sorting algorithms:
  - 1-Selection sort
  - 2-Insertion sort
  - 3-Mergesort
  - 4-Quicksort (no initial shuffling, one partition only) The question assumes the *first* element is the pivot!
  - 5-Heapsort In-place version of heapsort: 0…i will be a heap, i+1…n will be correct (largest values)
- Match each algorithm with the right row by writing its number (1-5) in the last column.

Problem 1 - Sorting

12	11	35	46	20	43	42	47	44	32	16	10	40	18	41	21	28	15	**0**
11	12	20	35	42	43	46	47	44	32	16	10	40	18	41	21	28	15	
10	11	12	46	20	43	42	47	44	32	16	35	40	18	41	21	28	15	
10	11	12	15	16	43	42	47	44	32	20	35	40	18	41	21	28	46	
43	32	42	28	20	40	41	21	15	11	16	10	35	18	12	44	46	47	
11	12	20	35	46	43	42	47	44	32	16	10	40	18	41	21	28	15	
10	11	12	15	16	18	20	21	28	32	35	40	41	42	43	44	46	47	**6**

Problem 2 - Heaps

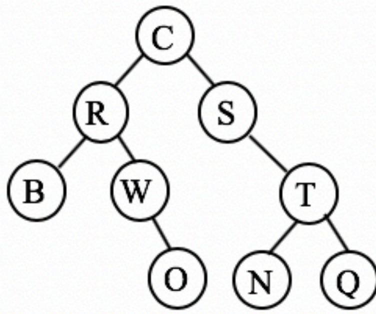
- Consider the following max-heap:

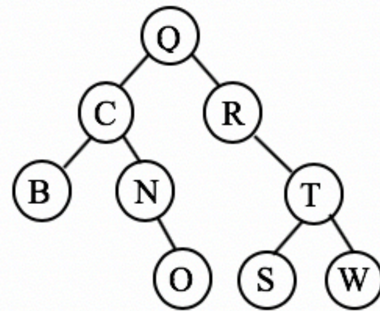


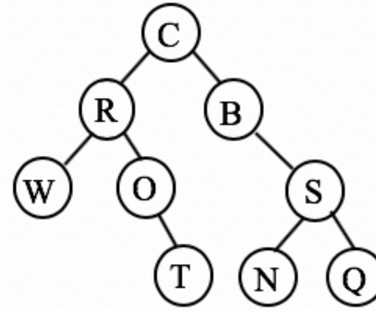
- Draw the heap after you insert key 13.
- Suppose you delete the maximum key from the original heap. Draw the heap after you delete 14.

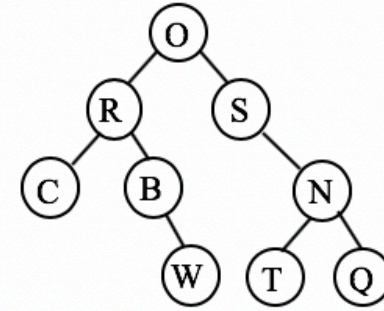
Problem 3 - Tree Traversals

- Circle the correct binary tree(s) that would produce both of the following traversals:
  - Pre-order: C R B W O S T N Q
  - In-order: B R W O C S N T Q









Problem 4 - Binary Trees

- You are extending the functionality of the `BinaryTree` class that represents binary trees with the goal of counting the number of leaves. Remember that `BinaryTree` has a pointer to a `root` Node and the inner class Node has two pointers, `left` and `left` to the root nodes that correspond to its left and right subtrees.
- You are given the following public method:

```
public int sumLeafTree()
  return sumLeafTree(root);
}
```

- Please fill in the body of the following recursive method

```
private int sumLeafTree(Node x){…}
```

Problem 5 - Binary Search Trees

- You are extending the functionality of the `BST` class that represents binary search trees with the goal of counting the number of nodes whose keys fall within a given `[low, high]` range. That is you want to count how many nodes have keys that are equal or larger than `low` and equal or smaller than `high`. Remember that `BST` has a pointer to a `root Node` and the inner class `Node` has two pointers, `left` and `left` to the root nodes that correspond to its left and right subtrees and a `Comparable Key key` (please ignore the value).
- You are given the following public method:

```
public int countRange(Key low, Key high)
  return countRange(root, Key low, Key high);
}
```

- Please fill in the body of the following recursive method

```
private int countRange(Node x, Key low, Key high){…}
```

Problem 6 - Iterators

- A programmer discovers that they frequently need only the odd numbers in an arraylist of integers. As a result, they decided to write a class `OddIterator` that implements the `Iterator` interface. Please help them implement the constructor and the `hasNext()` and `next()` methods so that they can retrieve the odd values, one at a time. For example, if the arraylist contains the elements [7, 4, 1, 3, 0], the iterator should return the values 7, 1, and 3. You are given the following public class:

```
public class OddIterator implements Iterator<Integer> {

// The array whose odd values are to be enumerated
private ArrayList<Integer> myArrayList;

//any other instance variables you might need

//An iterator over the odd values of myArrayList
public OddIterator(ArrayList<Integer> myArrayList){…}


public boolean hasNext(){…}


public Integer next(){…}
}
```

Answers

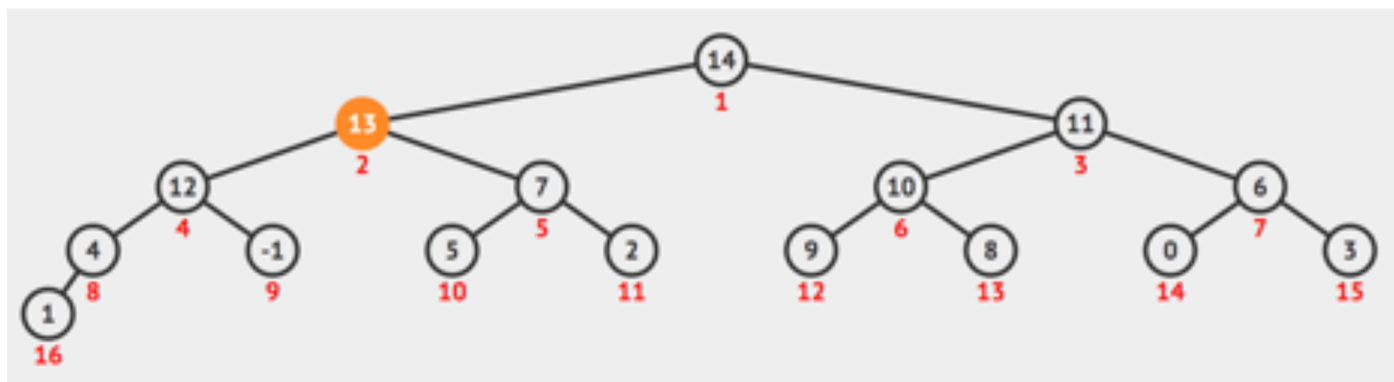
- Solution to Problem 1 - Sorting
- Solution to Problem 2 - Heaps
- Solution to Problem 3 - Tree traversals
- Solution to Problem 4 - Binary Trees
- Solution to Problem 5 - Binary Search Trees
- Solution to Problem 6 - Iterators

Solution to Problem 1 - Sorting

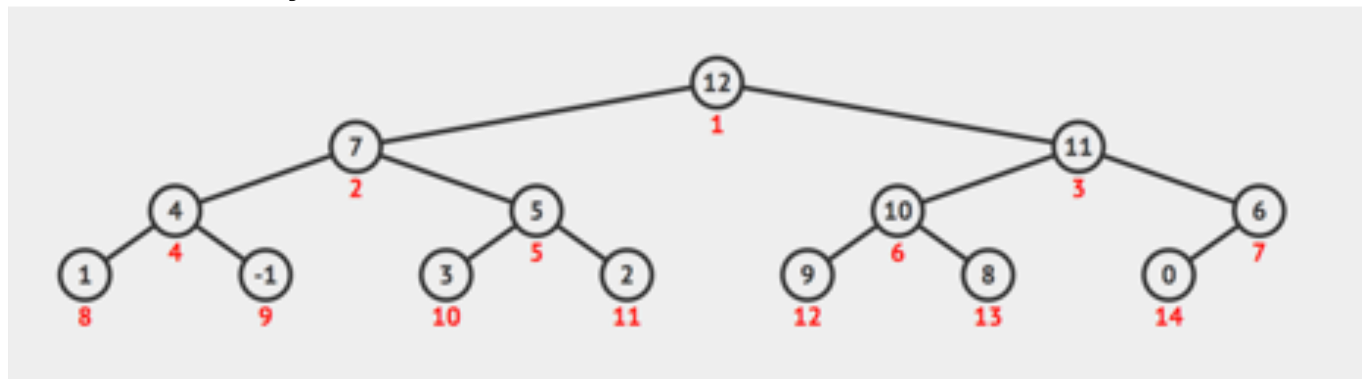
12	11	35	46	20	43	42	47	44	32	16	10	40	18	41	21	28	15	**0**
11	12	20	35	42	43	46	47	44	32	16	10	40	18	41	21	28	15	**2**
10	11	12	46	20	43	42	47	44	32	16	35	40	18	41	21	28	15	**4**
10	11	12	15	16	43	42	47	44	32	20	35	40	18	41	21	28	46	**1**
43	32	42	28	20	40	41	21	15	11	16	10	35	18	12	44	46	47	**5**
11	12	20	35	46	43	42	47	44	32	16	10	40	18	41	21	28	15	**3**
10	11	12	15	16	18	20	21	28	32	35	40	41	42	43	44	46	47	**6**

Solution to Problem 2 - Heaps

- Insert key 13:

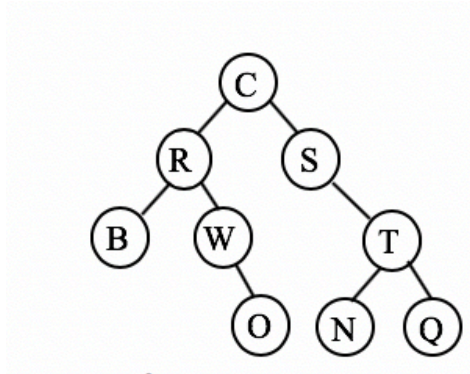


- Delete max-key (14):



Solution to Problem 3 - Tree traversals

- Pre-order: C R B W O S T N Q
- In-order: B R W O C S N T Q



Solution to Problem 4 - Binary Trees

```
private int sumLeafTree(Node x){
   if (x == null){
      return 0;
   }
   else if (x.left == null && x.right == null){
        return 1;
   }
   else{
      return sumLeafTree(x.left) + sumLeafTree(x.right);
   }
}
```

Solution to Problem 5 - Binary Search Trees

```
private int countRange(Node x, Key low, Key high){
    if (x == null){
        return 0;
    }
    if (x.key.compareTo(low)>=0 && x.key.compareTo(high)<=0){
        return 1 + countRange(x.left, low, high) + countRange(x.right, low, high);
    }
    else if (x.key.compareTo(low)<0){
        return countRange(x.right, low, high);
    }
    else{
        return countRange(x.left, low, high);
    }
}
```

```java
// SOLUTION TO 6 THAT DOESN'T REQUIRE hasNext BE CALLED BEFORE
// next()

public class OddIterator implements Iterator<Integer> {
    private ArrayList<Integer> list;
    private Integer nextVal = null;
    private int counter = 0;

    public OddIterator(ArrayList<Integer> list) {
        this.list = list;
        nextVal = loadNext();
    }

    // Search for the next odd value
    private Integer loadNext() {
        nextVal = null;

        while( nextVal == null && counter < list.size() ) {
            if( list.get(counter) %2 == 1 ) {
                nextVal = list.get(counter);
            }

            counter++;
        }

        return nextVal;
    }

    public boolean hasNext() {
        return nextVal != null;
    }

    public Integer next() {
        Integer answer = nextVal;
        nextVal = loadNext();
        return answer;
    }
}
```

Solution to Problem 6 - Iterators

```
public class OddIterator implements Iterator<Integer> {

    private ArrayList<Integer> myArrayList;
    int counter;

    public OddIterator(ArrayList<Integer> myArrayList){
        this.myArrayList = myArrayList;
        counter = 0;
    }

    //runs in O(n) time
    public boolean hasNext(){
        for (int i=counter; i<myArrayList.size(); i++){
            if(myArrayList.get(i)%2 == 1){
                counter = i;
                return true;
            }
        }
        return false;
    }


    //runs in O(1) time
    public Integer next(){
        return myArrayList.get(counter++);
    }
}
```

GOOD LUCK! YOU CAN DO THIS!