Question 1, Sorting questions:

Merge-sort and quick-sort are two advanced sorting algorithms which can obtain an average running time complexity of $O(n \log n)$. Usually, the actual sorting time depends on the way the elements arranged themselves (We assume the sorting is in an ascending order in this question). Counting the inversions in an array is a way to measure the order. An inversion is a pair of elements $(a_i, a_j)$ such that $i < j$ but $a_i > a_j$. For example, the array \{2, 4, 1, 3, 5\} has three inversions \( (2, 1), (4, 1), (4, 3) \). For a sorted array, the inversion number is 0.

(1-1)  (10 points) Implement a method with time complexity of $O(n^2)$ to count how many inversions there are for a given array. Print out the number of inversions for array \{47, 24, 83, 78, 36, 17, 96, 55\}.

(1-2)  (15 points) Through using the divide and conquer technique, you can improve the time complexity of counting inversions to $O(n \log n)$.

Implement a method to count inversions by modifying the code of merge-sort. Print out the number of inversions for array \{96, 83, 78, 55, 47, 36, 24, 17\}. (Hint: During the merging process of merge-sort, an inversion happens when each time you put the first element of the right array into the sorted array. This is because the first element in the left array has larger value than the first element in the right array.)

(1-3)  (10 points) In QuickSort, the element chosen as the pivot can be any element in an array. Instead of using the last element as the pivot in the sample code given in the lecture, pick the first element as the pivot in this question.

For array \{47, 24, 83, 78, 36, 17, 96, 55\}, write the numbers in the array after each partitioning step. Taking the first partitioning step for example, your answer should be 47 (as the pivot), \{36, 24, 17, 47, 78, 83, 96, 55\} (as the array after the partitioning step).

(1-4)  (15 points) In the real world, our unsorted data is not necessarily in a random order. The data could be almost sorted (i.e., its inversion number is small). In this situation, using neither the first element nor the last element as the pivot in QuickSort is a good choice. Because the two parts divided by the pivot element is quite uneven, this will make the QuickSort algorithm close to the worst case ($O(n^2)$).
in complexity). To avoid this situation, we can choose the pivot more wisely by picking a random element in the array.

Implement a quicksort method by using a random element in the array as a pivot. (Hint: you can use the Math.random() function to generate the index of a random element.)

**Question 2, Binary Tree questions:**

All the nodes of the Binary Tree for this question are defined with the following class:

```java
class TreeNode
{
    public int key;
    public Node left;
    public Node right;
}
```

You can refer to the sample code in lecture 19 for tree construction and traversal.

(2-1)  (10 points) What is the minimum and maximum number of nodes that a binary tree with height of \(h\) can contain? Write the number of nodes in terms of \(h\). The height of binary tree is the maximum number of nodes traversed from the root node to a leaf node. For example, a tree containing a root node only has a height of 1.

(2-2)  (15 points) Given a set of numbers \{16, 21, 4, 17, 5, 10, 5\} as keys, draw binary search trees with heights of 3 and 4. (Hint: it is possible that several binary search trees have a height of 4. Draw one of them.) Write the post-order traversal of the tree with a height of 3.

(2-3)  (10 points) Implement a method called `decreaseOddBy1`, which will go through each node of a given binary tree and decrease the key by 1 if the key of the node is an odd number.

(2-4)  (15 points) Implement a method `getChildrenNum` that returns the number of children for a given node.