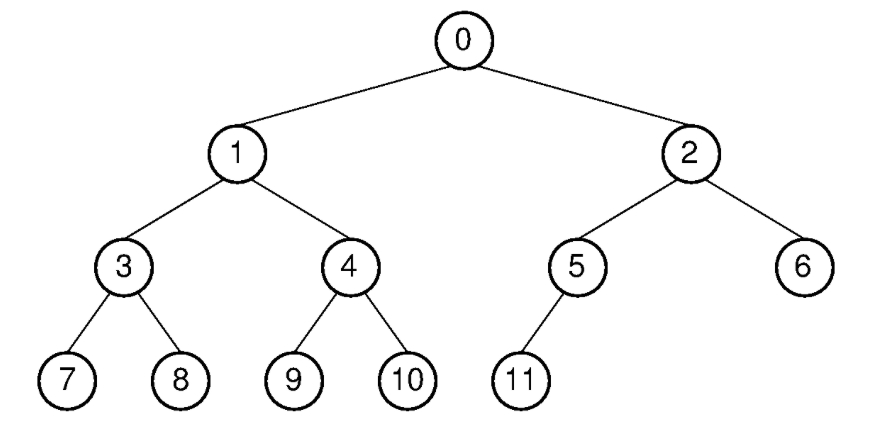
**HW5 CSCD320**

**Question 1 and 2 does NOT require programming, while question 3 need your programming.**

**To Turn in: please save your answers to the questions 1 and 2 in a pdf file. Then please bundle your MinHeap.java and Tester.java files for the question 3, along with the pdf file that contains the answers to question 1 and 2 into a zip file.** Turn in your single zip file on the **EWU Canvas** by going to CSCD320-01 course page on Canvas, then clicking Assignments🡪hw5->submit. Please name your zip file with your last name, followed by the first initial of your first name, followed by hw5. For example, if you are John Smith, name you file as smithjhw5.zip

**Question 1 (20%)**

**Give an array of integers {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}, which represents a complete binary tree in memory.**

****

As we learned in class, please explain how to construct a Max-heap with this input array by repeatedly calling siftdown procedure. Please include step by step explanations. After each step of swapping values, please **draw** the intermediate **graphical** tree diagram to guarantee partial credits. The examples in the lecture notes are quite illuminating regarding these intermediate status of the tree.

**Question 2 (10%)**

Please use **Mathematical Induction** to prove the following theorem about Full Binary Trees. In your proof, you need to provide three components -- a base case, induction hypothesis, and induction step, as we did in the classroom.

**Theorem to prove**: The number of leaves in a non-empty full binary tree is one more than the number of internal nodes.

**Question 3 (programming project) (70%)**

On basis of what we have learned about a MaxHeap, you are required to implement a **MinHeap java** class that meets the following requirements,

1. Data item in a node is smaller than or equal to its descendants. This is true for any node in the MinHeap.
2. You have to use a one dimensional array in the MinHeap implementation, as we did with the MaxHeap.
3. The MinHeap class will contain the following methods,

* void siftdown() //the method **equivalent** to the siftdown() in a MaxHeap. (15%)
* void buildheap(), // the method **equivalent** to the buildheap() in a MaxHeap. (5%)
* Comparable removemin(), //remove the root data of the MinHeap, equivalent to the removemax() in a MaxHeap (15%)
* int heapsize() //returns the actual number of elements in this MinHeap.
* public MinHeap(Comparable[] h, int num, int max) // constructor of the MinHeap
* boolean isLeaf(int pos) //as we did in the MaxHeap
* int leftchild(int pos) //as we did in the MaxHeap
* int rightchild(int pos)//as we did in the MaxHeap
* int parent(int pos) //as we did in the MaxHeap

Then, write a **Tester.java** file, and do the following operations,

1. Create an array A that contains a set of strings { “zoo”, “big”, “bike”, “good”, “apple”, “moon”, “mud”}.
2. Create a MinHeap object myheap = MinHeap(A, 7, 7);
3. myheap.buildheap(); //build a min-heap out of array A.
4. Then perform a heap sorting by following the logic idea (**pseudo code**) presented below: (20%)

* ArrayList sorted = new ArrayList();
* While(myheap.heapsize() > 0) {
* String cur = myheap.removemin(); // remove and return the next smallest object on each call.
* sorted.add(cur);
* }
* Print out all strings in the sorted ArrayList.

1. You expect to see on the standard output the sorted list of strings are {“apple”, “big”, “bike”, “good”, “moon”, “mud”, “zoo”}.
2. Please intuitively analyze the **time complexity** of the heap sorting algorithm presented in step 4 above. Please **print out** **on the standard output** your analysis of the time complexity, for example, using System.out.println(“………your analysis…..”);(5%)

**You can have your own design for any details that have not been explicitly stated in this document. You have the freedom to add more methods into the MinHeap.java file if needed.**