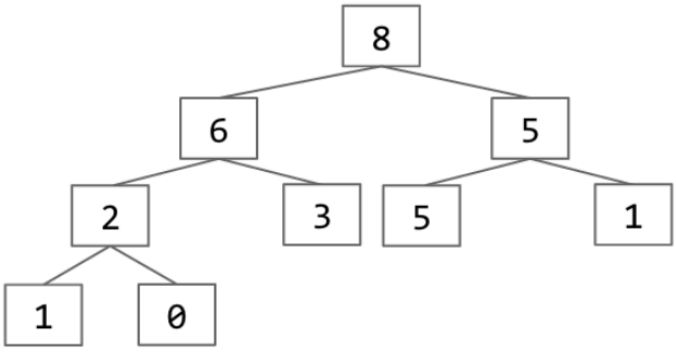
Spring 2016 MT2: Heaps and Balanced Trees

8. Balanced Trees (8 Points)

a) Suppose we have the max heap below, with array representation as shown. Show the heap after the maximum is deleted, using the procedure described in class. **Give your answer as an array.**

---	8	6	5	2	3	5	1	1	0			

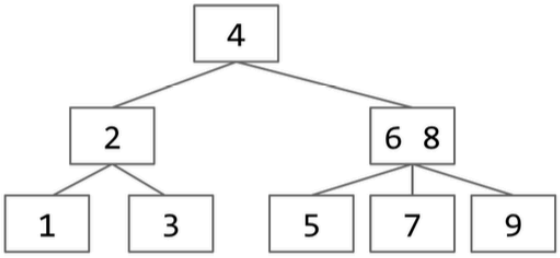


Your answer:

---												

b) Consider the 2-3 tree below. What order should we insert these numbers so that we get the tree shown? There may be multiple correct answers.

Answer:



c) Tumelo Barttrain suggests creating a special version of a heap where each item has 4 children to improve performance. Would such a heap have better, worse, or the same asymptotic performance in Θ notation as compared to a normal binary heap? Briefly explain your reasoning.

Spring 2016 Final: Heaps and Testing

9. Heaps and JUnit (10 points).

Write a JUnit test to check a method `public void minheapify(int[] arr) {...}` that is supposed to perform bottom-up min-heap heapification. This means ensuring that the array is actually a heap, and also that the array still has all the same items. Our tests will verify only correctness, not runtime. Assume there will be no duplicates (which may make it easier to test that the array still contains the same inputs after heapification).

- **Hint: We're not leaving index 0 blank, so the left child of k is 2k + 1, and the right child is 2k + 2.**
- Hint 2: Feel free to use assertEquals(x, y), assertTrue(b), assertArrayEquals(x, y), etc.
- Hint 3: If you don't remember the exact syntax but your meaning is clear, penalties will be minimal.

```
@Test
public static void testHeapify() {
      int[] arr = generateRandomIntArray();
      int[] original = new int[arr.length]; // copy of array before being
heapified
      System.arraycopy(arr, 0, original, 0, arr.length);
      minheapify(arr);
      // Check the integrity of the result using one or two calls to helper
methods
      testIsAHeap(arr);
      testHaveSameItems(original, arr);
}
private static void testIsAHeap(int[] arr) {




} //You may not need all lines for these methods. Must include at least one
assert!
private static void testHaveSameItems(int[] original, int[] arr) {



 }
```

Spring 2016 Final: Sorting and Trees

2. **Sorting (8 points).** a) **(6 pts)** Below, the leftmost column is an array of strings to be sorted. The column to the far right gives the strings in sorted order. Each of the remaining columns gives the contents of the array during some intermediate step of one of the algorithms listed below. Match each column with its corresponding algorithm. You will use each answer once. Write your answer in the blanks provided.

7777	1979	1979	7777	2001	1234	1234
2001	1234	2001	7777	8009	2001	1979
3015	1984	2015	4444	3015	3015	1981
2015	1981	2048	2015	2015	2015	1984
2048	2001	3015	7450	2016	2048	2001
8009	2015	4444	3015	2048	1981	2015
1979	2048	4500	2016	9150	1979	2016
7777	2016	7450	2001	1234	2016	2048
9150	3015	7777	1979	4444	1984	3015
4500	4500	7777	4500	7450	4500	4444
7450	4444	8009	2048	4500	7450	4500
4444	7777	9150	1981	7777	4444	7450
1234	7777	1234	1234	7777	7777	7777
1984	7450	1984	1984	1979	9150	7777
2016	8009	2016	8009	1981	7777	8009
1981	9150	1981	9150	1984	8009	9150
----	----	----	----	----	----	----
_1_	_ __	_ __	_ __	_ __	_ __	_7__

1: Unsorted, 2: Insertion, 3: Quick, 4: Heap, 5: LSD, 6: MSD, 7: Sorted

Notes:

- Quicksort is non-random and uses leftmost item as a pivot. The pivoting strategy is the Hoare two-pivot strategy, discussed in class.
- Insertion, Heapsort, LSD Radix Sort, and MSD Radix Sort as described in class.

b) **(1 pt)** One way to sort N items is to insert them randomly into a left leaning red black tree, and then traverse the LLRB. Which traversal should we use in order to print out the keys in sorted order? Circle one:

Preorder **Inorder** **Postorder** **Reverse-Preorder** **Reverse-Postorder**

c) **(1 pt)** _____ What is the worst case runtime of the sort described in part b? **Give your answer in Big Theta notation in terms of N**. Put your answer in the given blank.

Spring 2015 Final: Trees, Sorts, Best and Worst

2. **Basic Operations Turbo Edition (3 Points).**

a. Draw a binary tree that has a pre-order and in-order traversal of A, B, C, D, E.

b. Draw a **directed** graph whose DFS pre-order traversal is A, B, C, D, E, and whose DFS post-order traversal is C, B, D, E, A. Assume that ties are broken alphabetically.

3. **Best and Worst (4.5 Points).** Give bests and worsts as requested below.

a. Give an input that results in worst case runtime for insertion sort. Give your answer as an array of 5 integers, written in the blanks below.

____ ____ ____ ____ ____

b. Given an initially empty BST, give an insertion order for A, B, C, D, E, F, G that minimizes height. Assume we are not performing any balancing operations.

____ ____ ____ ____ ____ ____ ____

c. Suppose we want to build an array based stack that supports push and pop. In class, we learned how appropriate resizing rules can empower data structures with excellent amortized runtime. Suppose our stack has the following two rules:

- Before each push, if the array **is full, double** the size before pushing.
- After each pop, if the array is less than **half full, halve** the size of the array.

What is the worst-case amortized runtime for a sequence of pushes and pops? Give your answer in $\Theta(\cdot)$ notation in terms of N, the maximum size of the array that is reached during the stack's lifetime.

Spring 2015 Final: Data Structures and Algorithms

4. Comparative Data Structures and Algorithms (7.5 points).

a. What is the size of the largest binary min heap that is also a valid BST? Draw an example assuming all keys are letters from the English alphabet. **Assume the heap has no duplicate elements.**

b. What is the size of the largest binary max heap that is also a valid BST? Draw an example assuming all keys are letters from the English alphabet. **Assume the heap has no duplicate elements.**

c. What is the size of the largest binary min heap in which the fifth largest element is a child of the root? You do not need to draw an example. **Assume the heap has no duplicate elements.**

d. Give an example of when you'd use Quicksort over Mergesort.

e. Give an example of when you'd use a Trie-based map instead of a HashMap.

f. Why did we bother introducing heaps for implementing priority queues instead of just using left-leaning red-black binary search trees?