**CMPS 101**
**Final Review**
**Solutions to Selected Problems**

1. Let $G = (V, E)$ be a graph with *n* vertices, *m* edges, and *k* connected components.
   a. Show that if *G* is connected and acyclic, then $m = n - 1$. Use induction on either *m* or *n*.
   b. Show that if *G* is acyclic, then $m = n - k$. Use part (a).
   c. Show that if *G* is connected, then $m \geq n - 1$. Use induction on *m*.
   d. Show that in any case $m \geq n - k$. Use part (c).

   **Solution:**
   See the proofs of Lemmas 1-4 in the Graph Theory handout.

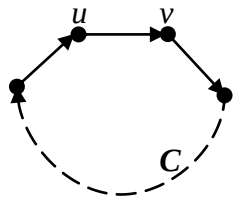
2. Let *G* be a digraph. Determine whether, at any point during a Depth First Search of *G*, there can exist an edge of the following kind.
   a. A tree edge that joins a white vertex to a gray vertex.
   b. A back edge that joins a black vertex to a white vertex.
   c. A forward edge that joins a gray vertex to a black vertex.
   d. A cross edge that joins a black vertex to a gray vertex.
   e. A tree edge that joins a gray vertex to a gray vertex.
   f. A forward edge that joins a black vertex to a black vertex.
   g. A cross edge that joins a white vertex to a black vertex.
   h. A back edge that joins a gray vertex to a white vertex.

   **Solution:**
   See the solution to problem 4 in Homework Assignment 5.

3. a. State the parenthesis theorem. — See page 606 (all pages are 3rd edition)
   b. State the white path theorem. — See page 608
   c. State the max-Heap property. — See page 152
   d. State the min-Heap property. — See page 153
   e. State the Binary Search Tree properties. — See page 287
   f. State the Red Black Tree properties. — See page 308

4. Let *G* be a directed graph. Prove that if *G* contains a directed cycle, then *G* contains a back edge. (Hint: use the white path theorem.)

   **Proof:**
   Suppose *G* contains a directed cycle, call it *C*. Let *v* be the first vertex on *C* to be discovered by DFS, and let *u* be the vertex on *C* which precedes *v*.




   Since no vertex on *C* is discovered before *v*, at the time of discovery of *v* the vertices of *C* form a path from *v* to *u* consisting of white vertices. By the white-path theorem, *u* becomes a descendent of *v* in some DFS tree. Therefore (*u*, *v*) is a back edge. ///

5. Let $T$ be a binary tree. Let $n(T)$ denote the number of nodes in $T$, and $h(T)$ denote the height of $T$. Show that $h(T) \geq \lfloor \lg(n(T)) \rfloor$.

**Solution:**
See the solution to problem 4 on Homework Assignment 6.

6. Re-write the algorithms Heapify, and HeapIncreaseKey from the point of view of a min-Heap, rather than a max-Heap. (In particular, HeapIncreaseKey should be renamed HeapDecreaseKey.)

**Solution:**
The min-Heap property says: $A[\text{parent}(i)] \leq A[i]$ for all $i$ in the range $2 \leq i \leq \text{HeapSize}[A]$. We re-write Heapify to maintain this property:

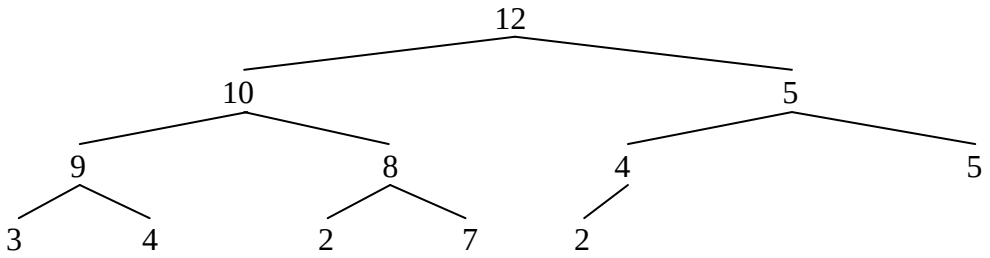
Heapify($A$, $i$) pre: The left and right subtrees of node $i$ are valid min-Heaps.
1. $l = \text{left}[i]$
2. $r = \text{right}[i]$
3. if $l \leq \text{HeapSize}[A]$ and $A[l] < A[i]$
4. smallest $= l$
5. else
6. smallest $= i$
7. if $r \leq \text{HeapSize}[A]$ and $A[r] < A[\text{smallest}]$
8. smallest $= r$
9. if smallest $\neq i$
10. swap $A[i] \leftrightarrow A[\text{smallest}]$
11. Heapify($A$, smallest)

HeapIncreaseKey becomes, upon renaming:

HeapDecreaseKey($A$, $i$, $k$) pre: $1 \leq i \leq \text{HeapSize}[A]$
1. if $k < A[i]$
2. $A[i] = k$
3. while $i \geq 2$ and $A[\text{parent}(i)] > A[i]$
4. swap $A[i] \leftrightarrow A[\text{parent}(i)]$
5. $i = \text{parent}(i)$

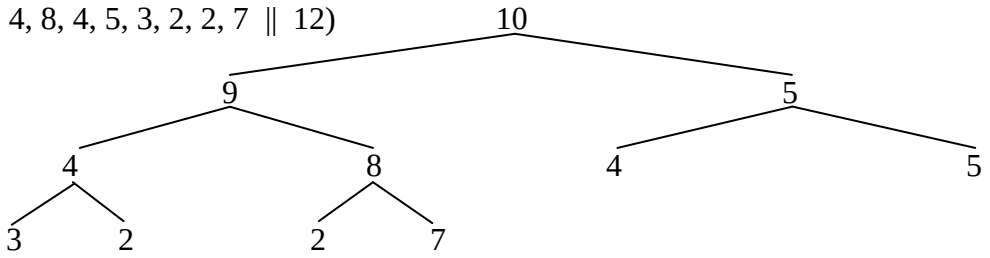
7. Trace HeapSort on the following arrays:
   a. (9, 3, 5, 4, 8, 2, 5, 10, 12, 2, 7, 4)
   b. (5, 3, 7, 1, 10, 12, 19, 24, 5, 7, 2, 6)
   c. (9, 8, 7, 6, 5, 4, 3, 2, 1)

**Solution to part (a):** After performing BuildHeap on the array (9, 3, 5, 4, 8, 2, 5, 10, 12, 2, 7, 4) we get the array (12, 10, 5, 9, 8, 4, 5, 3, 4, 2, 7, 2). Drawn as an ACBT this looks like:

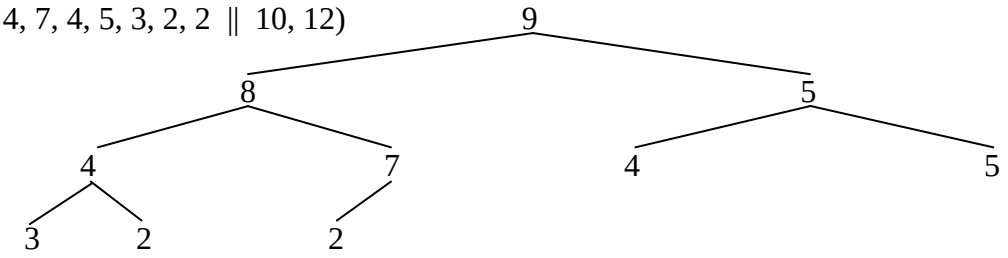



We repeatedly swap the root with the rightmost leaf, decrement HeapSize, and call Heapify() on the root. We show the array and the corresponding tree after each iteration of this process, placing a double vertical bar || within the array at the end of the heap.

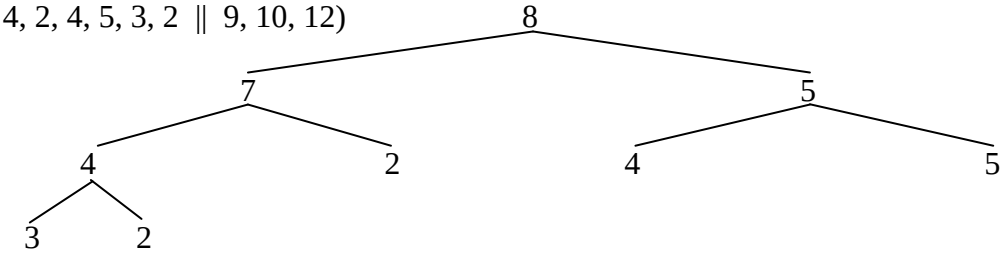
(10, 9, 5, 4, 8, 4, 5, 3, 2, 2, 7 || 12)



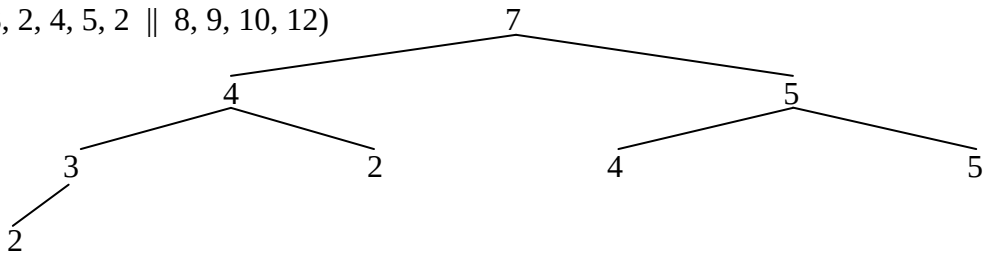

(9, 8, 5, 4, 7, 4, 5, 3, 2, 2 || 10, 12)



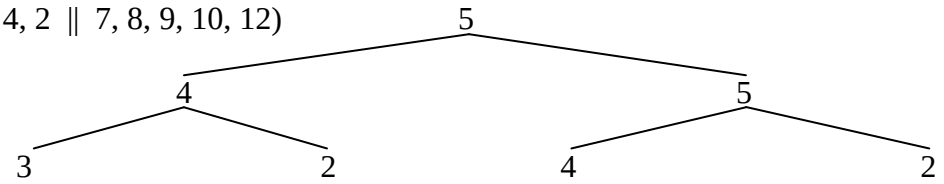

(8, 7, 5, 4, 2, 4, 5, 3, 2 || 9, 10, 12)



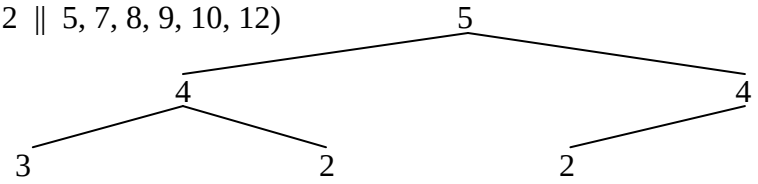
(7, 4, 5, 3, 2, 4, 5, 2 || 8, 9, 10, 12)



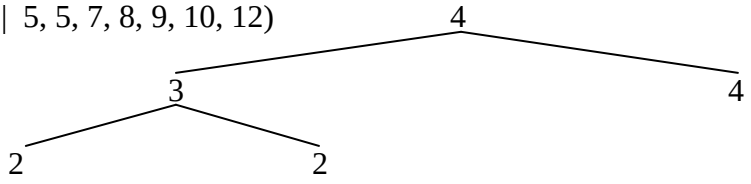

(5, 4, 5, 3, 2, 4, 2 || 7, 8, 9, 10, 12)



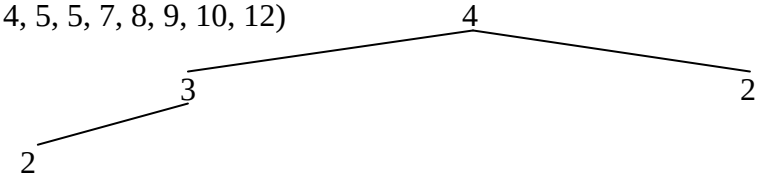

(5, 4, 4, 3, 2, 2 || 5, 7, 8, 9, 10, 12)



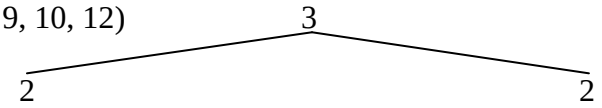

(4, 3, 4, 2, 2 || 5, 5, 7, 8, 9, 10, 12)



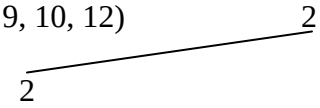

(4, 3, 2, 2 || 4, 5, 5, 7, 8, 9, 10, 12)




(3, 2, 2 || 4, 4, 5, 5, 7, 8, 9, 10, 12)




(2, 2 || 3, 4, 4, 5, 5, 7, 8, 9, 10, 12)




(2 || 2, 3, 4, 4, 5, 5, 7, 8, 9, 10, 12) 2

The algorithm stops at this point. / / /

8. Let $G$ be a directed graph, and let $s, x \in V(G)$. Suppose that after Initialize($G$, $s$) is executed, some sequence of calls to Relax( , ) results in $d[x]$ becoming finite. Show that $G$ contains an $s$-$x$ path of weight $d[x]$. (Use strong induction on the number of calls to Relax( , ).)

**Proof:**
If $n = 0$ the only $d$-value that is finite is that of the source vertex $s$. Indeed, there is an $s$-$s$ path in $G$ of weight $d[s] = 0$, namely the trivial path. The base case is therefore satisfied.

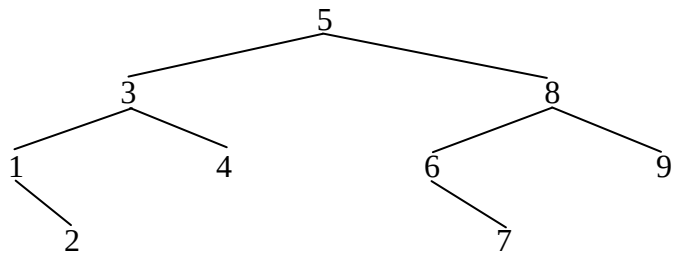
Let $n > 0$, and assume for any vertex $y$, that if $d[y]$ becomes finite during a sequence of fewer than $n$ relaxations, then $G$ contains an $s$-$y$ path of weight $d[y]$. Now suppose $d[x]$ becomes finite during a sequence of $n$ calls to Relax( , ). Then an edge having $x$ as terminus must have been relaxed in this sequence. (Recall relaxation of an edge only changes the terminal vertex.) Let $y$ be the origin of that edge. On the call to $\text{Relax}(y, x)$, $d[x]$ was set to the value $d[y] + w(y, x)$. Since we suppose that this number is finite, $d[y]$ must have been finite before this relaxation step was executed. Thus $d[y]$ became finite during a sequence of fewer than $n$ relaxations, and by the induction hypothesis, $G$ contains an $s$-$y$ path of weight $d[y]$. That path, followed by the single edge $(y, x)$, constitutes an $s$-$x$ path in $G$ of weight $d[y] + w(y, x) = d[x]$. The result follows for any relaxation sequence by induction. ///

9. Let $G$ be a directed graph, $s, x \in V(G)$, and suppose Initialize($G$, $s$) is executed. Show that the inequality $\delta(s, x) \le d[x]$ is maintained over *any* sequence of calls to Relax( , ). (Use the result of problem 8.)

**Proof:**
Assume, to get a contradiction, that some sequence of calls to Relax( , ) results in $d[x] < \delta(s, x)$ being true. Then necessarily $d[x]$ is finite. By the result of problem 8, $G$ contains an $s$-$x$ path of weight $d[x]$. But this contradicts the very definition of $\delta(s, x)$ as the weight of a <u>minimum-weight</u> $s$-$x$ path in $G$. This contradiction shows our assumption was false, and hence no such relaxation sequence can exist. The inequality $\delta(s, x) \le d[x]$ is therefore maintained over *any* sequence of calls to Relax( , ). ///

12. Draw the Binary Search Tree resulting from inserting the keys: 5 8 3 4 6 1 9 2 7 (in that order) into an initially empty tree. Write pseudo-code for the following recursive algorithms, and write their output when run on this tree.
    a. InOrderTreeWalk()
    b. PreOrderTreeWalk()
    c. PostOrderTreeWalk()

**Solution:**




(a) 1 2 3 4 5 6 7 8 9
(b) 5 3 1 2 4 8 6 7 9
(c) 2 1 4 3 7 6 9 8 5

13. Assign colors to the nodes in the above BST in such a way that it becomes a valid RBT. Note there is more than one way to do this. Find all such color assignments.
**Solution:**
There are two distinct ways to assign colors so as to satisfy the RBT properties: (1) Assign 2 and 7 to be colored Red, and all other nodes Black; and (2) assign 2, 3, 7, and 8 to be Red, all other nodes Black.

14. Let *x* be a node in a Red-Black Tree, and let $N(x)$ denote the number of internal nodes in the subtree rooted at *x*. Show that $N(x) \geq 2^{\text{bh}(x)} - 1$. (Hint: use strong induction on $\text{height}(x)$.)

**Proof:**
If $\text{height}(x) = 0$, then *x* is a leaf, so also $\text{bh}(x) = 0$ and $2^{\text{bh}(x)} - 1 = 1 - 1 = 0$. In this case the subtree rooted at *x* has no internal nodes, whence $N(x) = 0$. Thus the inequality reduces to $0 \geq 0$, which is true, and the base case is satisfied.

Let $\text{height}(x) > 0$, and assume for any node *y* with $\text{height}(y) < \text{height}(x)$ that $N(y) \geq 2^{\text{bh}(y)} - 1$. Since $\text{height}(x) > 0$, *x* is an internal node and necessarily has two children (one or both of which may be nil.) If $\text{left}[x]$ is Red, then $\text{bh}(\text{left}[x]) = \text{bh}(x)$, while if $\text{left}[x]$ is Black, then $\text{bh}(\text{left}[x]) = \text{bh}(x) - 1$. In any case $\text{bh}(\text{left}[x]) \geq \text{bh}(x) - 1$. Likewise $\text{bh}(\text{right}[x]) \geq \text{bh}(x) - 1$. Now since $\text{height}(\text{left}[x]) < \text{height}(x)$ and $\text{height}(\text{right}[x]) < \text{height}(x)$, the induction hypothesis and the preceding inequalities give us both

$$N(\text{left}[x]) \geq 2^{\text{bh}(\text{left}[x])} - 1 \geq 2^{\text{bh}(x)-1} - 1$$

and

$$N(\text{right}[x]) \geq 2^{\text{bh}(\text{right}[x])} - 1 \geq 2^{\text{bh}(x)-1} - 1$$

The set of internal nodes belonging to the subtree rooted at *x* consists of those internal nodes in *x*'s left subtree, those in *x*'s right subtree, together with *x* itself. Therefore

$$N(x) = N(\text{left}[x]) + N(\text{right}[x]) + 1 \geq \left(2^{\text{bh}(x)-1} - 1\right) + \left(2^{\text{bh}(x)-1} - 1\right) + 1 = 2 \cdot 2^{\text{bh}(x)-1} - 1 = 2^{\text{bh}(x)} - 1$$

The result follows for all nodes at any height by induction. ///

15. Let *T* be a Red-Black Tree having *n* internal nodes, and height *h*. Show that $h \leq 2\lg(n+1)$. (Hint: use the result of problem 14 and RBT property (4).)

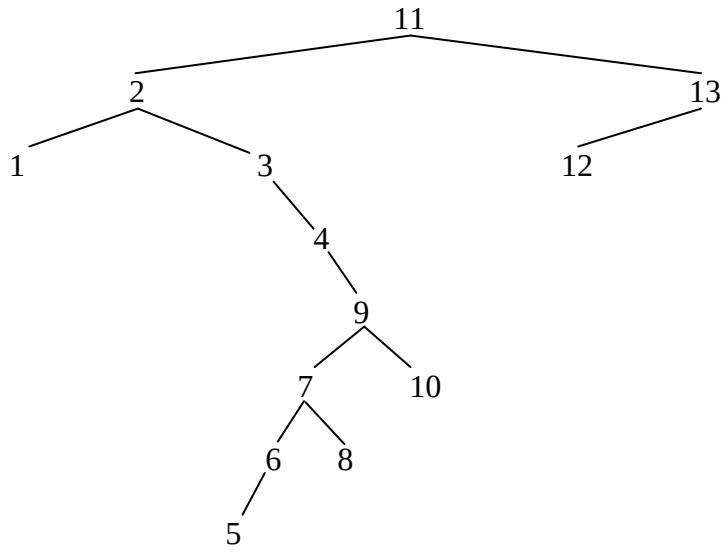
**Proof:**
Recall the RBT property (4) says that every Red node has two black children. Consequently no path from $\text{root}[T]$ to a descendant leaf can have two consecutive Red nodes, and therefore at least half of the nodes on such a path are Black. It follows that $\text{bh}(\text{root}[T]) \geq h/2$. By the result of problem 14

$$n = N(\text{root}[T]) \geq 2^{\text{bh}(\text{root}[T])} - 1 \geq 2^{h/2} - 1$$
$$\therefore \quad 2^{h/2} \leq n + 1$$
$$\therefore \quad \frac{h}{2} \leq \lg(n+1)$$
$$\therefore \quad h \leq 2\lg(n+1)$$

as claimed. ///

16. Insert the following keys (in order) into an initially empty Binary Search Tree: 11, 2, 13, 1, 3, 12, 4, 9, 7, 10, 6, 8, 5. Draw the resulting Binary Search Tree. Prove that it is not possible to assign colors Red and Black to the nodes of this tree in such a way that the Red-Black tree properties are satisfied. (Hint: use contradiction and the result of problem 15.)

**Solution:**
After the insertions we get:




Assume, to get a contradiction, that it is possible to assign colors Red and Black to the above nodes in such a way that the Red-Black Tree properties are satisfied. After drawing in the required nil leaves, we would have a RBT with $n = 13$ internal nodes, and height $h = 8$. Observe $2\lg(13+1) = 3.807..$, so that $h > 2\lg(n+1)$, contradicting the result of problem 15. We conclude that no such assignment of colors can exist. ///

17. Insert the keys from problem 16 (in order) into an initially empty Red-Black Tree using the algorithms `RB-Insert()` and `RB-Insert-Fixup()` from lecture and the text. Draw all intermediate trees in this process.

See the animation at: http://www.ece.uc.edu/~franco/C321/html/RedBlack/