Name: Key

Email address: ____________________

CSE 373 Spring 2009: Midterm #2

(closed book, closed notes, NO calculators allowed)

Instructions: Read the directions for each question carefully before answering. We may give partial credit based on the work you **write down**, so if time permits, show your work! Use only the data structures and algorithms we have discussed in class or that were mentioned in the book so far.

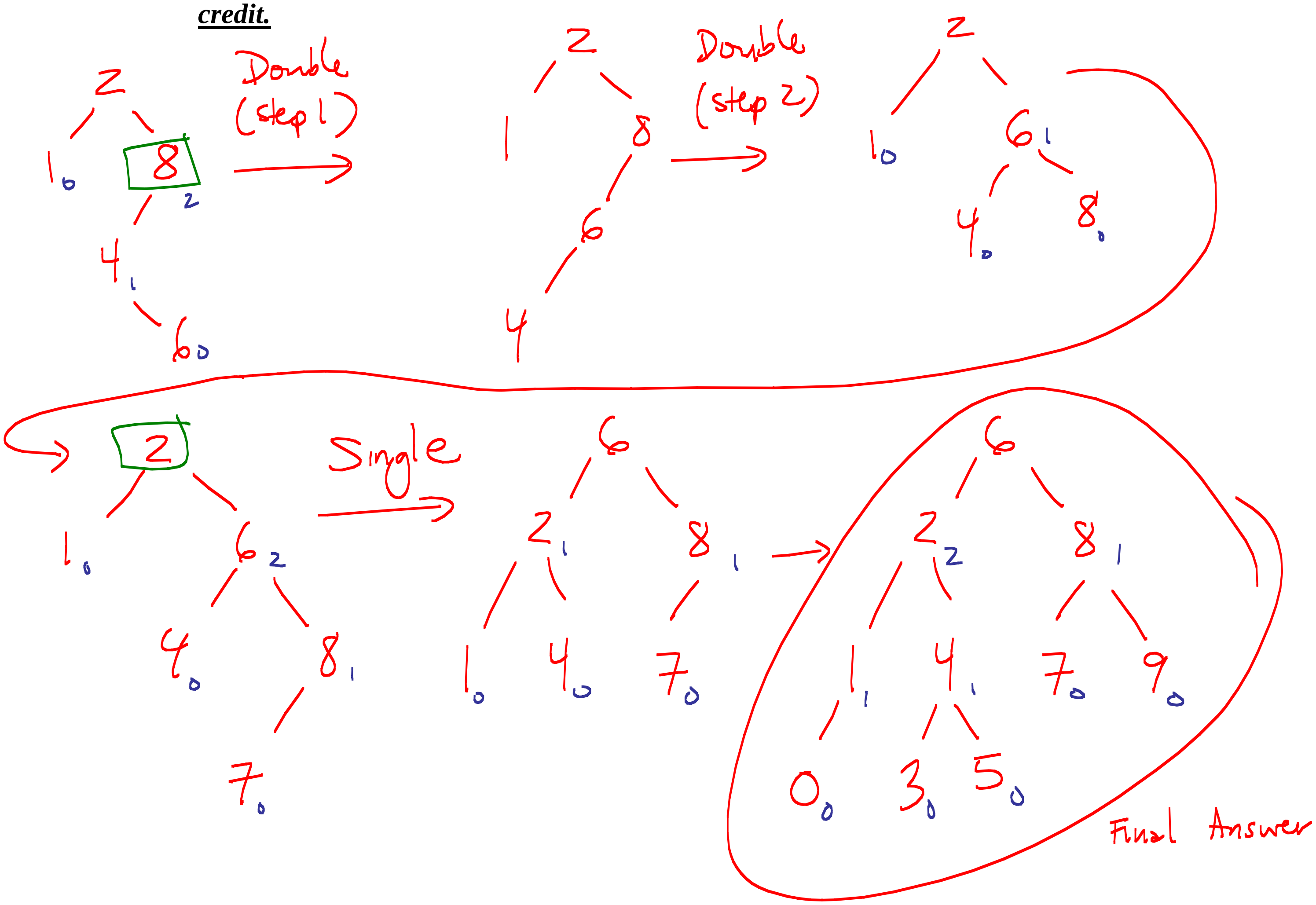
Note: For questions where you are drawing pictures, please circle your final answer for any credit.

Good Luck!

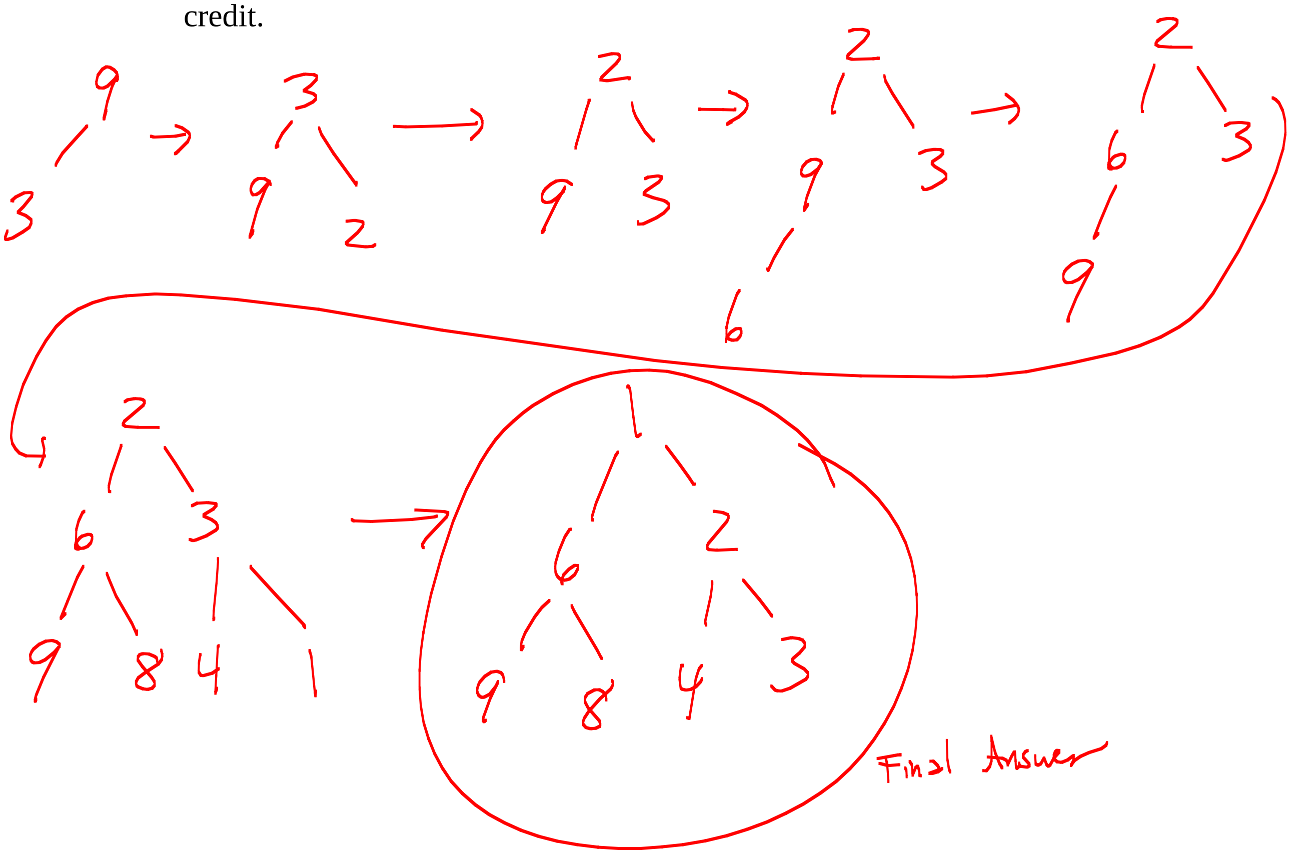
Total: 63 points. Time: 50 minutes.

Question	Max Points	Score
1	6	
2	6	
3	12	
4	7	
5	11	
6	6	
7	15	
Total	63	

1) [6 pts total] **AVL Trees** Draw the AVL tree that results from inserting the keys: 2, 8, 1, 4, 6, 7, 9, 3, 5, 0 in that order into an initially empty AVL tree. You are only required to show the final tree, although drawing intermediate trees may result in partial credit. If you draw intermediate trees, ***please circle your final tree for ANY credit.***



2) [6 pts total] **Binary Min Heaps** Draw the binary min heap that results from inserting 9, 3, 2, 6, 8, 4, 1 in that order into an initially empty binary min heap. You are only required to show the final tree, although drawing intermediate trees may result in partial credit. If you draw intermediate trees, please **circle your final result** for any credit.

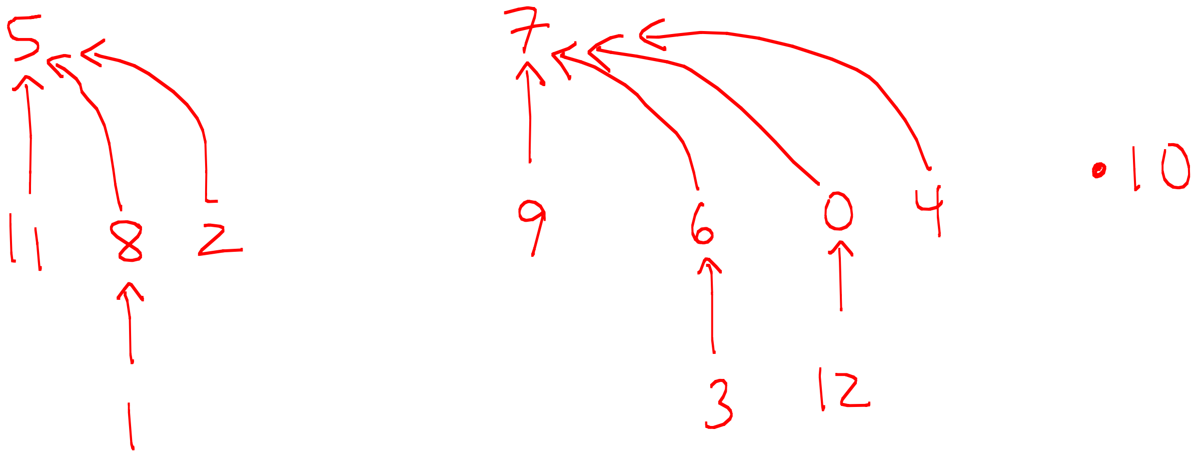


3) [14 pts total] **Disjoint Sets:** Consider the set of initially unrelated elements:
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12.

a) [6 pts] Draw the final forest of up-trees that result from the following sequence of operations using on union-by-size. Break ties in size by keeping the root of the set the first argument belongs to as the new root. Use the hw4Union given below that is able to take any element as a parameter, not necessarily the name of a set. It finds the set each element belongs to and then unions those two sets. Be sure to include all elements of the set in your final answer.

```
hw4Union(int x, int y) {
  u = Find(x); // Find here is done without path compression.
  v = Find(y);
  Union(u,v); // Use Union-by-size
}
```

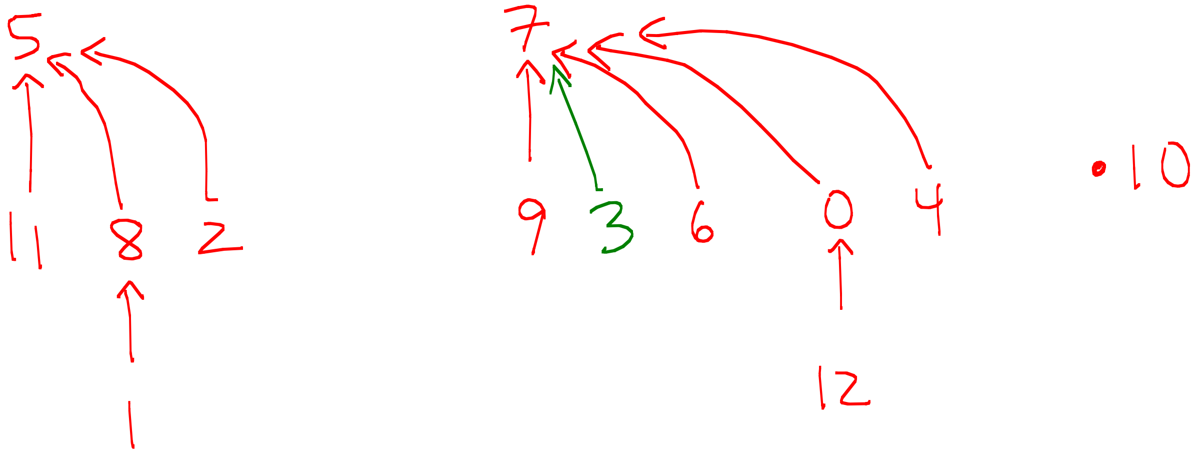
hw4Union(5,11), hw4Union(7,9), hw4Union(6,3), hw4Union(9,3), hw4Union(0, 12), hw4Union (8,1), hw4Union(12,6), hw4Union(5,1), hw4Union(11, 2), hw4Union(4, 7).



b) [2 pts] Draw the array representation of your answer above. ***Use the value: -size to represent a root.***

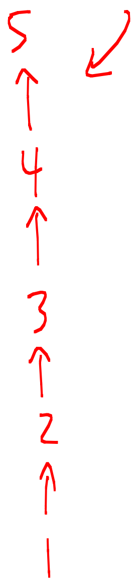
0	1	2	3	4	5	6	7	8	9	10	11	12
7	8	5	6	7	-5	7	-7	5	7	-1	5	0

c) [2 pts] Draw the new forest of up-trees that results from doing a Find(3) where here Find is done with path compression on your forest of up-trees from a).



d) [2 points] In terms of big-O, what is the worst case running time for a **single find operation** with path compression. Assume that union is *always* done by having the second parameter point to the first parameter (assuming you are always passed roots as parameters). In other words, union by size is NOT used. N = total # of elements in all sets.

ex.



O(N) A simple union alg like the one described here might create a set like this: Union(2,1), Union(3,2), Union(4,3), Union(5,4), Worst case, for find(1), requires traversal of all elements to reach the root. Path compression only adds O(height) work & does not provide a guaranteed bound for an individual operation.

in this case = N

4) [7 pts total] **Miscellaneous**:

a) [2 pts] Inserting a value into a D heap containing N elements takes worst case time (give the most exact answer you can, but no explanation necessary):

$O(\log_D N)$

b) [2 pts] Deletemin on a D-heap containing N elements takes worst case time (give the most exact answer you can, but no explanation necessary):

$O(D \log_D N)$

c) [3pts] As defined in lecture, a dynamic equivalence relation is one where the following three properties apply (circle the three best answers):

(1) Equivocal

[✓ (2) Transitive]

(3) Reciprocal

[✓ (4) Reflexive]

(5) Elastic

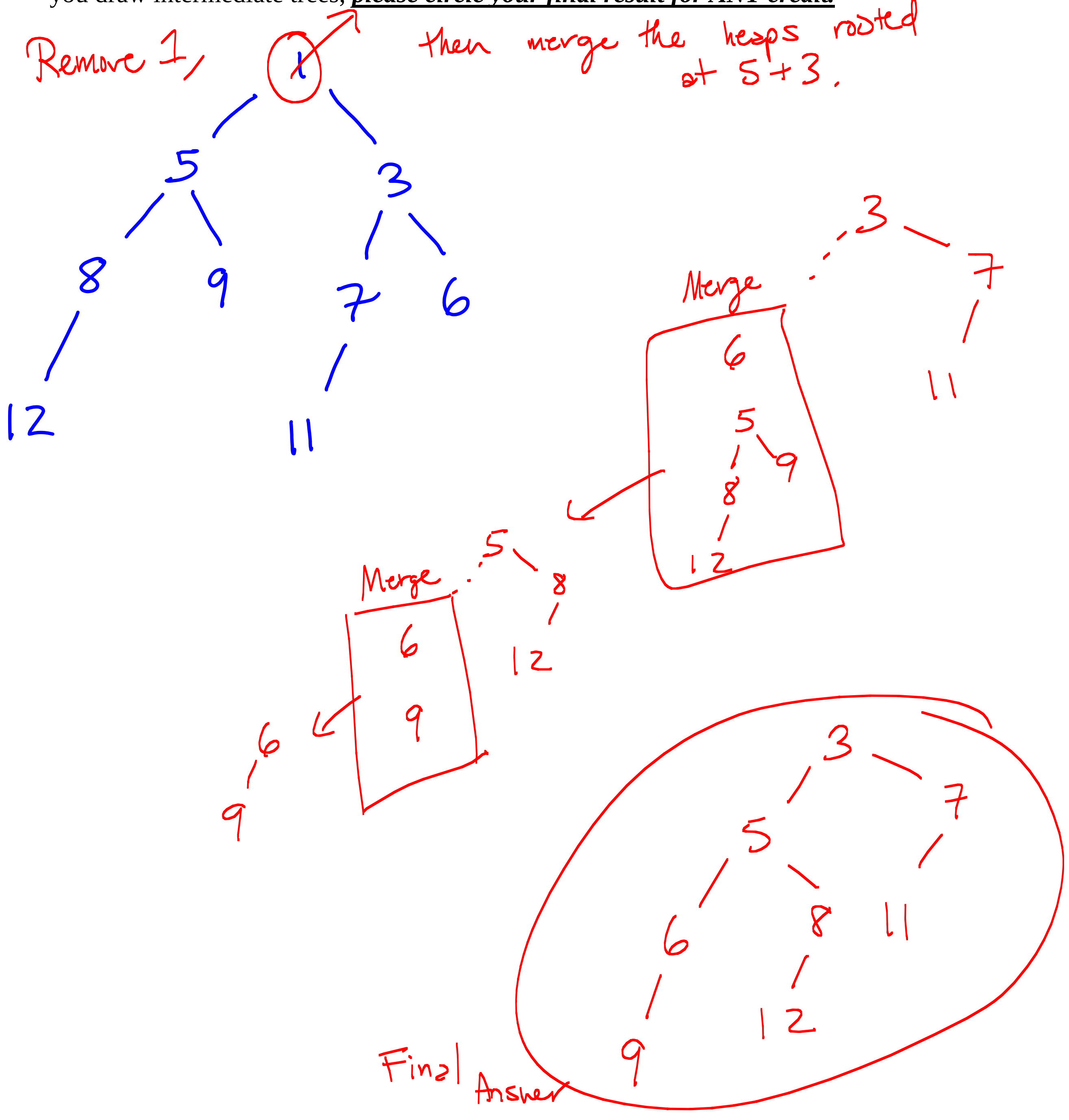
(6) Recursive

(7) Quadratic

(8) Semantic

[✓ (9) Symmetric]

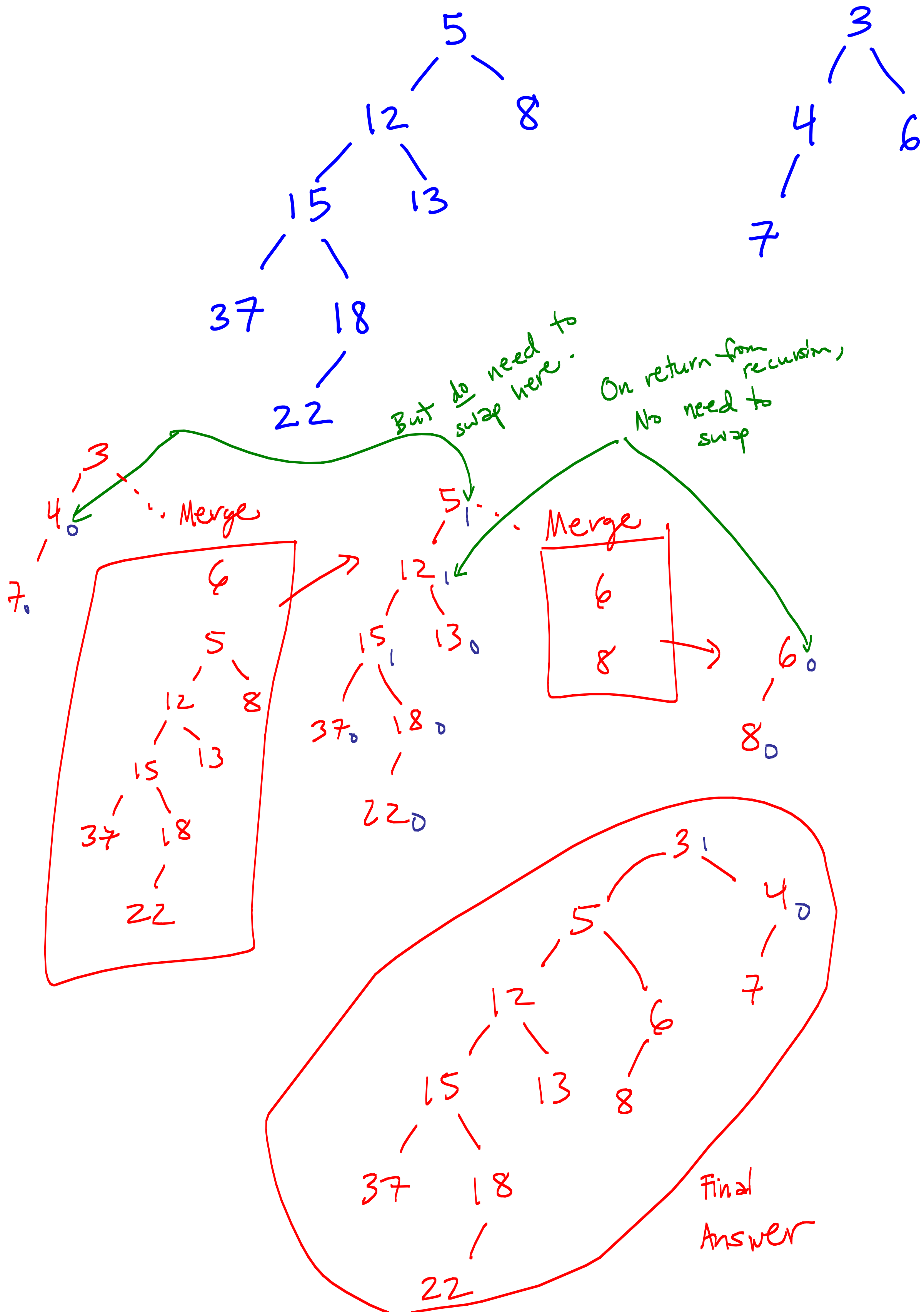
5) [6 pts total] **Skew Heaps** Draw the skew heap that results from doing a **deletemin** on the skew heap shown below. You are only required to show the final tree, although if you draw intermediate trees, ***please circle your final result for ANY credit.***



Remove 1, then merge the heaps rooted at 5+3.

6) [11 pts total] **Leftist Heaps**:

a) [6 pts] Draw the ***leftist*** heap that results from merging the two leftist heaps shown below. You are only required to show the final heap, although if you draw intermediate heaps, ***please circle your final result for ANY credit.***

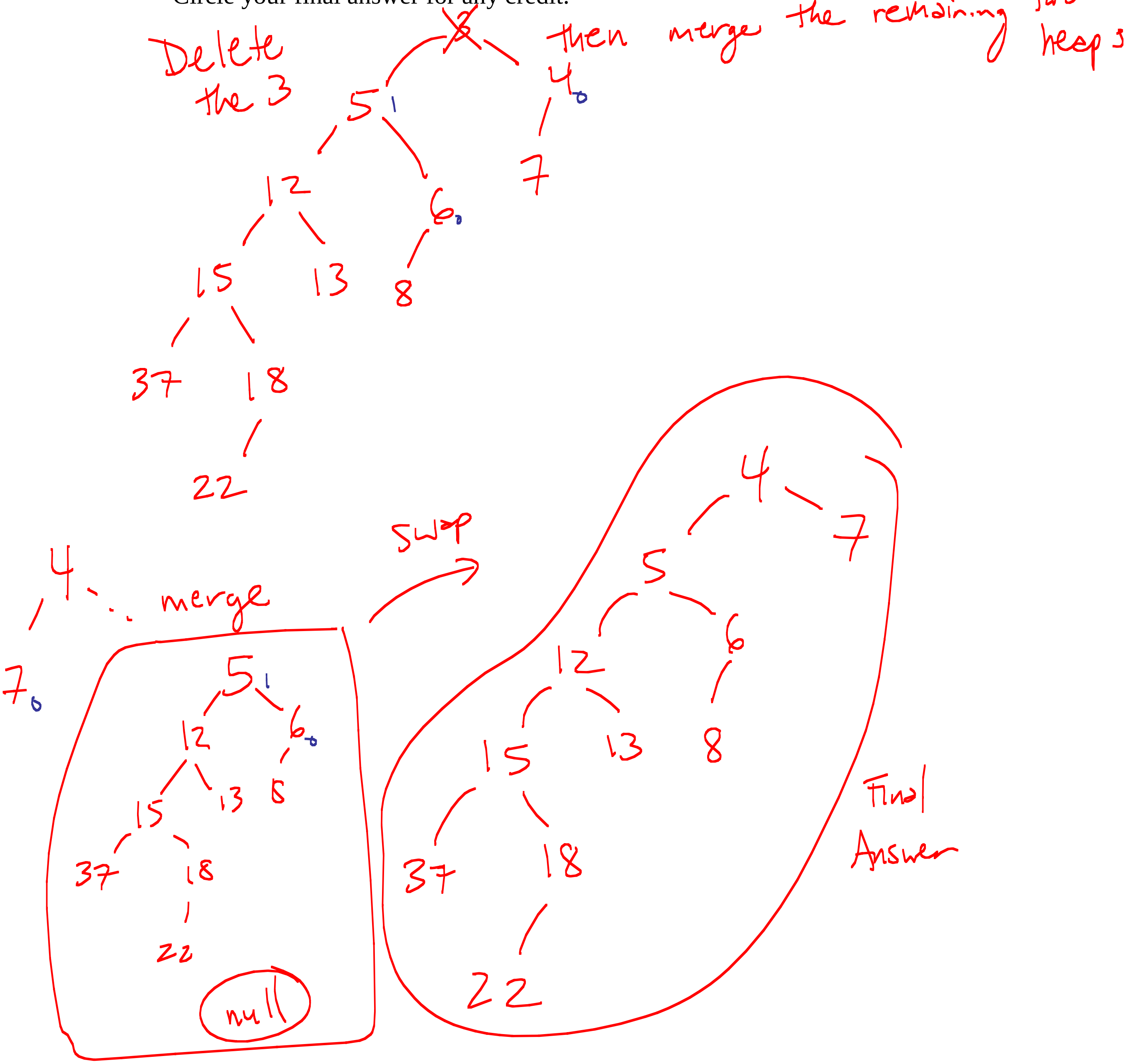


5
12
8
15
13
37
18
22
3
4
6
7
But do need to swap here.
On return from recursion, No need to swap
Merge
Merge
Final Answer

b) [2 pts] What is the null path length of the **root** in your final heap from part a)?

1

c) [3 pts] Draw the result of doing one **deletemin** on the heap you created in part a). Circle your final answer for any credit.



Delete the 3 then merge the remaining two heaps

Final Answer

7) [15 pts total] **Worst Case Running Time Analysis:** Give the tightest possible upper bound for the ***worst case*** running time for each of the following in terms of *N*. You MUST choose your answer from the following (not given in any particular order), each of which could be re-used (could be the answer for more than one of a)-e)):

$O(N^2)$, $O(N \log N)$, $O(N)$, $O(N^2 \log N)$, $O(2^N)$, $O(N^3)$, $O(\log N)$, $O(N^N)$, $O(1)$, $O(N^4)$

****For any credit, you must explain how you got your answer – be specific as to why the bound you give is appropriate.** (eg. Worst case running time for Find in a binary search tree is O(N) because you might need to traverse from root down to lowest level of tree to find the value, and worst case depth of node is N.)

Assume that the most time-efficient implementation is used and that all keys are distinct. Use N to represent the total number of elements. ***Bases of logarithms are assumed to be 2 unless otherwise specified.***

a) Print out all values in an **AVL tree** containing N elements ***from largest to smallest***.
Explanation: This could be done with a traversal that visited right, print self, then visit left. The total cost for a traversal in O(N) as each element will be visited at most a constant number of times.

a) $\boxed{O(N)}$

b) Delete the minimum value in a **skew heap** containing N elements. **Explanation:** O(log N) bound for skew heap operations is an amortized bound & does not apply for an individual operation. For a single operation the worst case time bound (due to height of tree) in O(N)

b) $\boxed{O(N)}$

c) Insert a value into a **binary min heap** (implemented using an array) containing N elements. **Explanation:** Inserting involves putting value in next available space on the bottom row & then percolating up. In the worst case, must compare with parent O(log N) times until reach the root, as a complete tree w. N elements has height log N.

c) $\boxed{O(\log N)}$

d) Finding (but not removing) the *minimum* value in a **leftist heap** containing N elements. **Explanation:** The minimum is stored at the root – this takes constant time to access. No modifications to the heap needed.

d) $\boxed{O(1)}$

e) Finding the *minimum* value in an **AVL tree** containing N elements. **Explanation:** Finding the minimum requires going left from the root until it is no longer possible to go left. This is the minimum value. Since the height of an AVL tree is at most O(log N), you must go down at most O(log N) levels to find the minimum.

e) $\boxed{O(\log N)}$