Name: Key

Email address: ______________________

CSE 373 Winter 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: 68 points. Time: 50 minutes.

Question	Max Points	Score
1	15	
2	14	
3	12	
4	8	
5	13	
6	6	
Total	68	

1) [15 points 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*. Choose your answer from the following, each of which could be re-used:

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

****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 the N to represent the total number of elements. ***Bases of logarithms are assumed to be 2 unless otherwise specified.***

a) Finding an element in a **hash table** of tablesize N, containing N elements where separate chaining is used and each bucket points to a sorted linked list.
Explanation: Worst case is all elements hashed to the same bucket. Even though the linked list is sorted, you would need to potentially visit all the nodes in the list.

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

b) Merging two **binary min heaps** (both implemented using an array) containing N elements each.
Explanation: Copy the two arrays into the new array $O(N)$ and use Floyd's buildheap $O(N)$ to convert the new array into a heap.

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

c) Insert a value into a **skew heap** containing N elements.
Explanation: No guarantee on time for an individual merge/insert operation. Height could still be $O(N)$.

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

You don't know what the minimum value is!

d) Finding the *minimum* value in a hash table currently containing N elements of table size = 3N. The hash table is implemented using open addressing with linear probing as the collision resolution strategy.
Explanation: Potentially must examine all 3N buckets until you have seen all N values. Just do a sequential traversal thru the table.

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

e) Finding the *maximum* value in a **leftist heap** containing N elements.
Explanation: Due to the linked structure of leftist heap, basically you must examine every node on your way to examining all leaves.

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

2) [14 points total] **Hashing**:

a) [7 pts] Draw the contents of the hash table in the boxes below given the following conditions:

The size of the hash table is 7.
Open addressing and quadratic probing is used to resolve collisions.
The hash function used is H(k) = k mod 7

What values will be in the hash table after the following sequence of insertions? Draw the values below, and show your work for partial credit.

10, 36, 17, 19, 24

0	24_2	
1	36	
2		
3	10	17_0 24_0
4	17_1	24_1
5	19	
6		

b) [3 pts] What is the **load factor** for the table above?

$\lambda = 5/7$

c) [4 pts] Is the value 7 a good choice for table size? Give one good reason for picking 7 and one bad reason for picking 7 for table size.

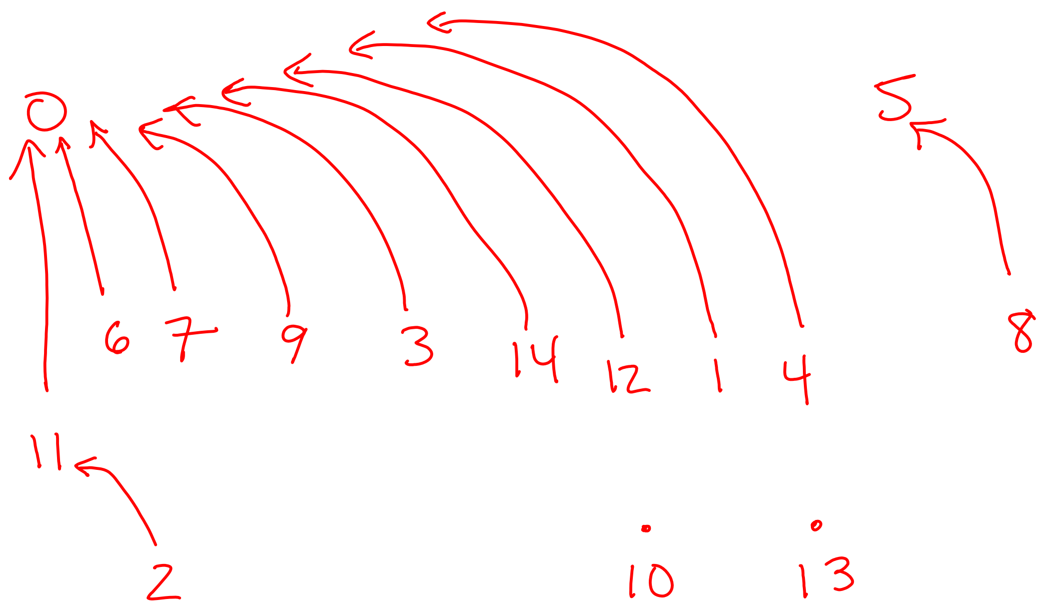
7 is a **good** choice because: 7 is prime

7 is a **bad** choice because: If inserting 5 values, this is $> \frac{1}{2}$ full. With quadratic probing there is no guarantee that you will find an empty spot

3) [12 points total] **Disjoint Sets:** Consider the set of initially unrelated elements: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14.

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. Perform a find (without path compression) if you need to find which set an element belongs to and then union the two sets as you normally would.

Union(0,6), Union(6,7), Union(7,9), Union(9,3), Union(0, 14), Union (5,8), Union(12,6), Union(1,12), Union(11, 2), Union(7,11), Union(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	13	14
-11	0	11	0	0	-2	0	0	5	0	-1	0	0	-1	0

c.) [2 pts] Draw the new forest of up-trees that results from doing a Find(6) with path compression on your forest of up-trees from (a).

Identical to part a).

d.) [2 points] In terms of big-O, how long **total** does it take to do the maximum number of unions possible and M find operations if union by size (weighted union) and path compression is used where N = total # of elements in all sets? (You may assume you are given roots as parameters to union)

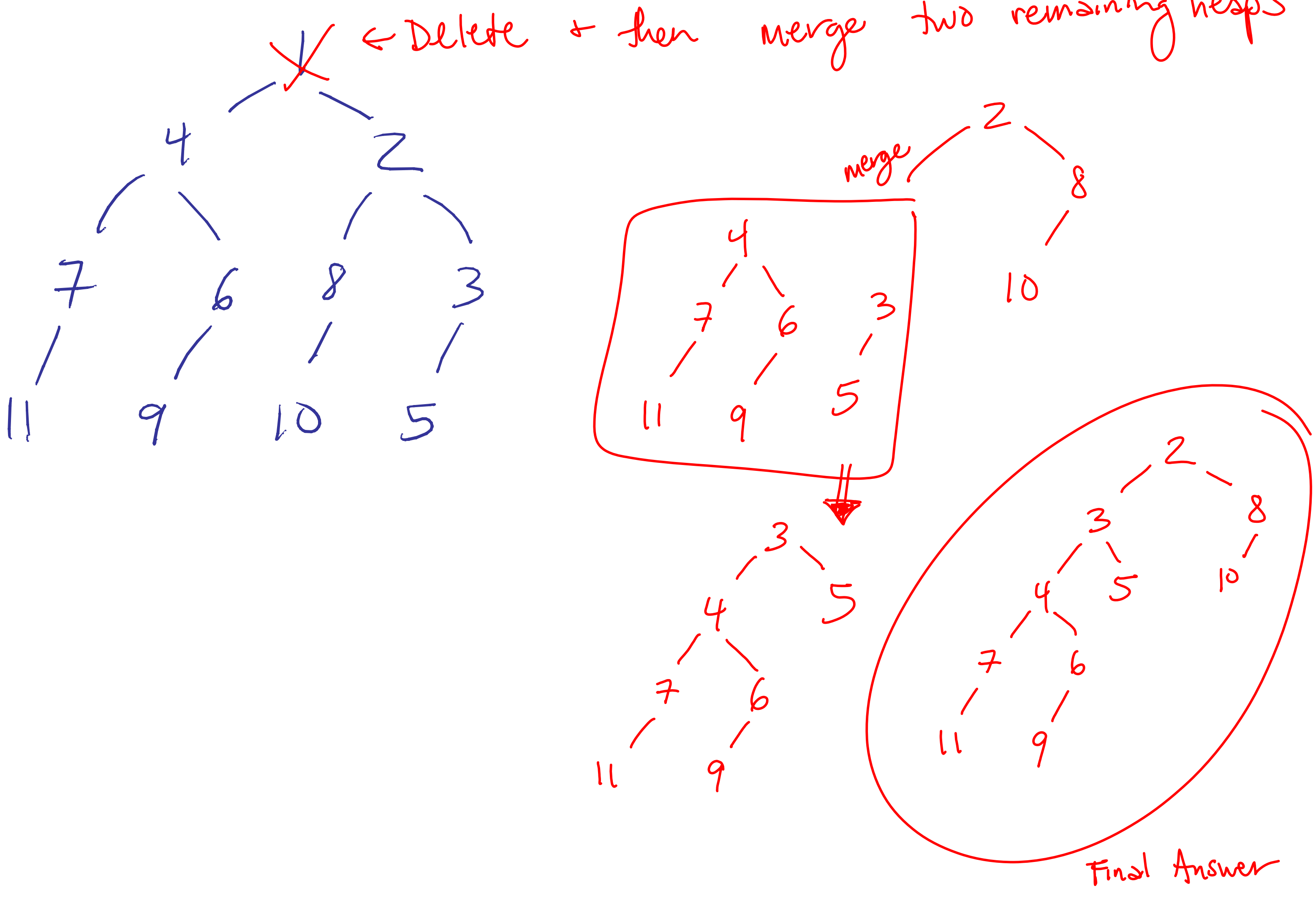
Max # of unions = N-1

Total cost of N-1 unions + M finds:

$= O(N + M)$

4) [8 points total] **Skew Heaps**

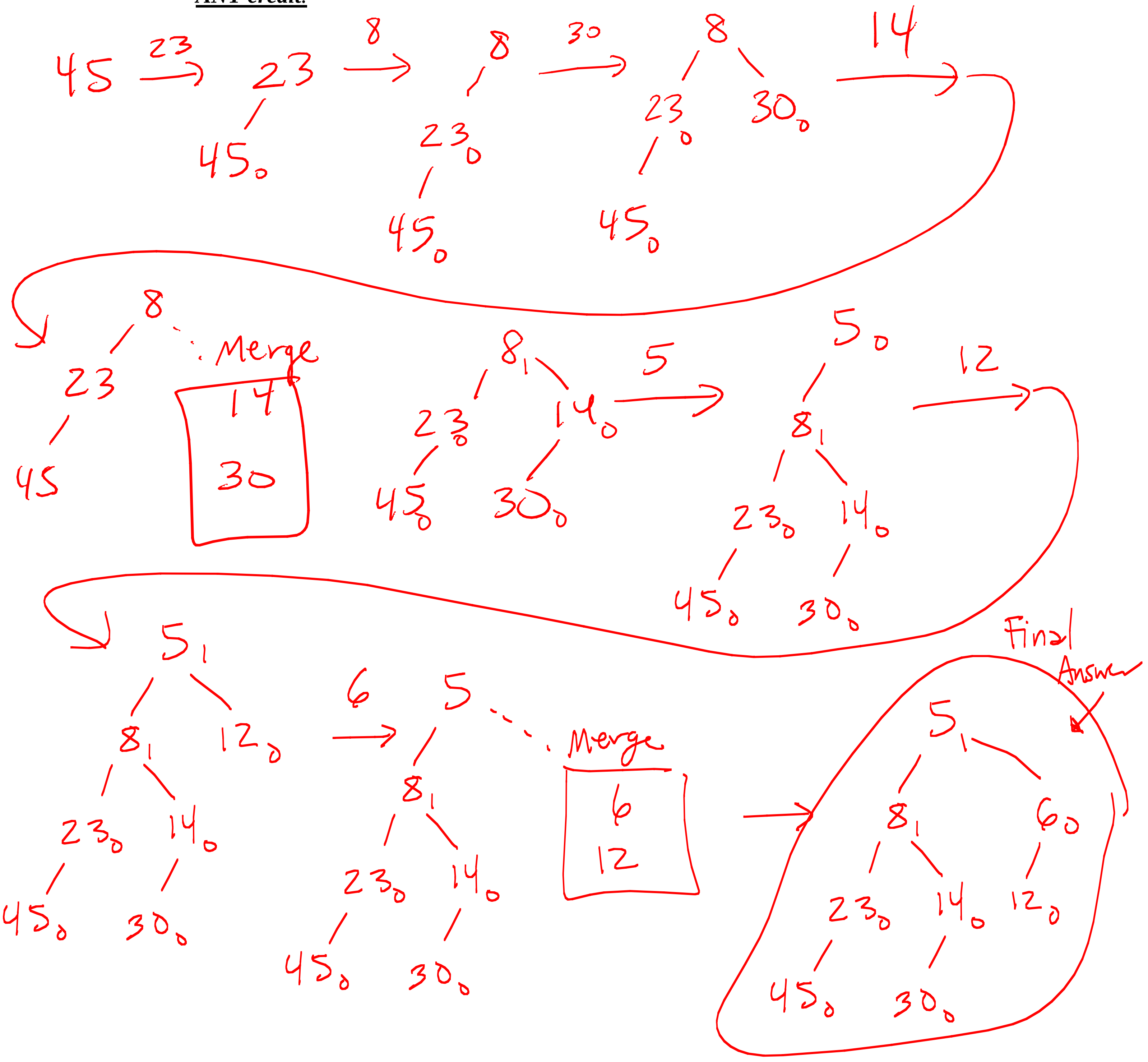
a) [6 points] Draw the skew heap that results from doing a **deletemin** on the skew heap shown below. You are only required to show the final tree, although if you draw intermediate trees, ***please circle your final result for ANY credit.***



b) [2 pts] Deletemin in a skew heap takes **worst case** time: $O(N)$

5) [13 points total] **Leftist Heaps**:

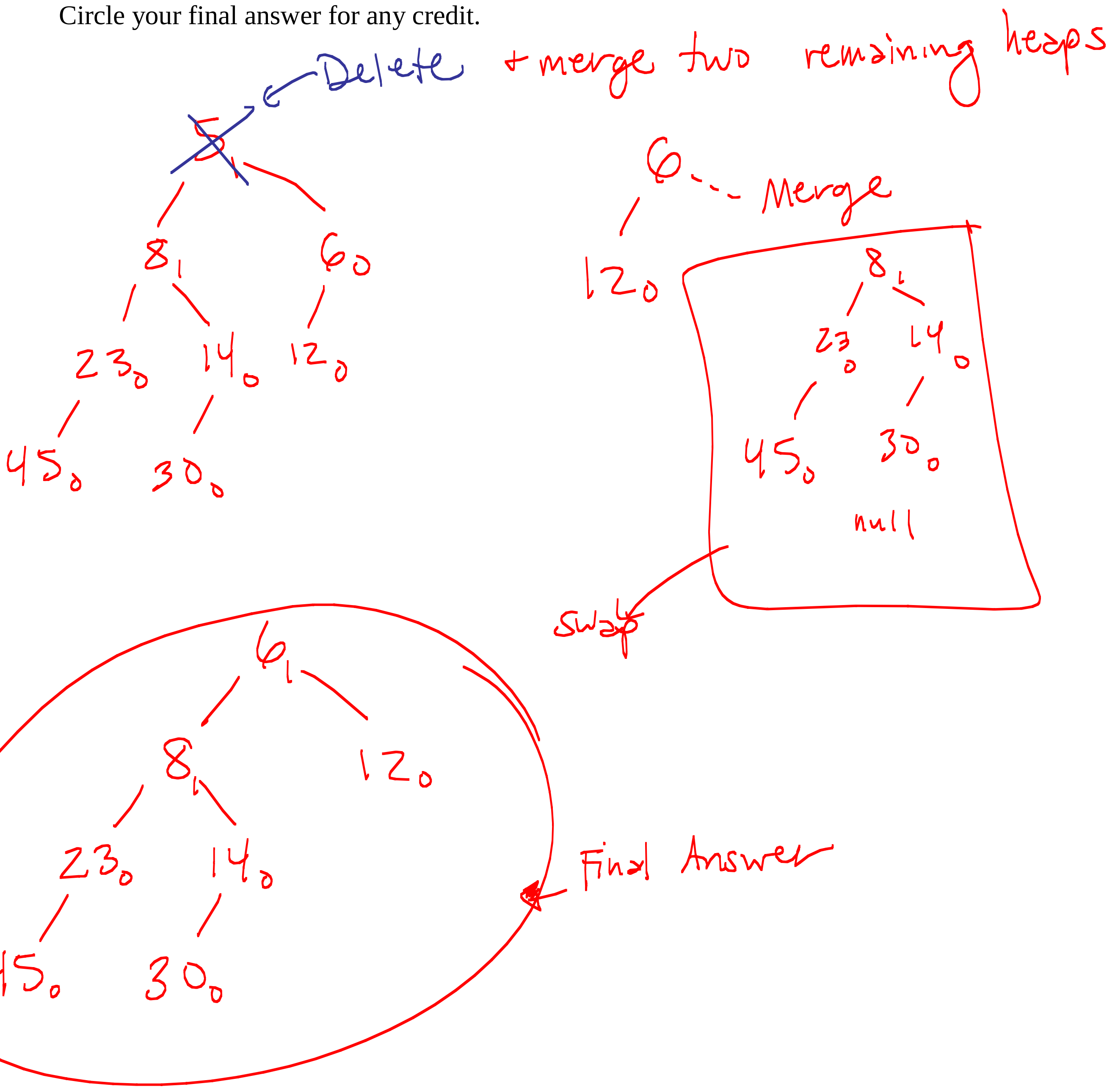
a) [8 pts] Draw the ***leftist*** heap that results from inserting: 45, 23, 8, 30, 14, 5, 12, 6 in that order into an initially empty heap. You are only required to show the final heap, although if you draw intermediate heaps, ***please circle your final result for ANY credit.***



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.



6) [6 points total] **Memory Hierarchy**: [Note: Feel free to answer both Options 1 and Options 2, you will however only get a maximum of 6 points for this question. Option 1 is the one I would prefer, but if you are not confident of your Option 1 answer, you can try Option 2 for up to 2 points. Points from the two options will not combine.]

*Option 1 [6 points]: Which has better **data** locality and why:
a) A hash table with separate chaining where each bucket is a sorted linked list.
b) A hash table with open addressing where linear probing is used as the collision resolution strategy.

I think choice (circle one) **a)** [✓ **b)**] would have better data locality.

It demonstrates (circle one) [✓ **spatial**] **temporal** locality in this situation:
(Be specific about *what* exactly has locality and *when* (on what operations).

Linear probing means that when a collision occurs the next sequential location is probed. This pattern of access demonstrates spatial locality. This will occur both on insertions + finds.
(There is no real spatial locality in a separate chaining hash table as described in a).)

Say something else here about the other type of locality (the one you did not circle above). Does your choice demonstrate this type of locality or not? If so when?

There is not really any temporal locality.
(Temporal locality might occur with the first element in each linked list in a separate chaining hash table described in a).)

*Option 2 [2 points]: ***Alternately***, for [2 points] define spatial and temporal locality (hint, a 3-4 word answer to each of these is not good enough to get the 2 points).

Spatial locality: (locality in space) If an item is accessed, items whose address is close by (close in space) are likely to be accessed soon.

Temporal Locality: (locality in time) If an item is accessed, that item is likely to be accessed soon.