Name: Sample Solution

Email address: ____________________

CSE 373 Spring 2010: 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	8	
2	12	
3	13	
4	6	
5	8	
6	16	
Total	63	

1) [8 points total] Disjoint Sets

The uptrees used to represent sets in the union-find algorithm can be stored in two *n*-element arrays: one giving the parent of each node (or -1 if the node has no parent), and the other giving the number of items in a set if the node is the root (representative node) of a set. For example, we can represent a collection of sets containing the numbers 1 through 14 as follows:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
up	6	5	4	-1	6	-1	11	6	10	11	5	11	-1	13
weight	-	-	-	2	-	10	-	-	-	-	-	-	2	-

a) Draw a picture of the uptrees represented by the data in the above arrays.

$$\begin{tikzcd}
 & & & 6 & & & 13 & 4 \\
 & & 5 \arrow[ur, bend left] & 8 \arrow[u] & 1 \arrow[ul] & & 14 \arrow[u] & 3 \arrow[u] \\
 & 11 \arrow[ur, bend left] & 2 \arrow[u] & & & & & \\
10 \arrow[ur, bend left] & 7 \arrow[u] & 12 \arrow[ul, bend right] & & & & & \\
9 \arrow[u] & & & & & & &
\end{tikzcd}$$

1) (cont)
b) Now, **draw** a new set of uptrees and **update** the data arrays as needed to show the results of executing the following two set operations:

```
          union(find(7), find(14));
          find(2);
```

You should assume that the find operations use **path compression** and that the union operation uses **union-by-size (aka union by weight)**. In case of ties, always make the higher numbered root point to the lower numbered one.

$$\begin{tikzcd}
 & & & 6 & & & & 4 \\
7 \arrow[urrr] & & 5 \arrow[ur] & 8 \arrow[u] & 1 \arrow[ul] & 2 \arrow[ull] & 13 \arrow[ulll] & 3 \arrow[u] \\
 & 11 \arrow[uurr] & & & & & 14 \arrow[u] & \\
10 \arrow[ur] & & 12 \arrow[ul] & & & & & \\
9 \arrow[u] & & & & & & &
\end{tikzcd}$$

c) Update the numbers in the arrays at the top of the previous page to reflect the picture after part b).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
up	6	6	4	-1	6	-1	6	6	10	11	6	11	6	13
weight	-	-	-	2	-	12	-	-	-	-	-	-	2	-

2) [12points total] Hashing

Draw the contents of the two open addressing hash tables in the boxes below. The size of the hash table is 9. The hash function used is H(k) = k mod 9

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

$$\begin{array}{ccccc} 18, & 16, & 10, & 7, & 26 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 7 & 1 & 7 & 8 \end{array}$$

a) Linear Probing

0	18	26_1
1	10	26_2
2	26_3	
3		
4		
5		
6		
7	16	7_0
8	7_1	26_0

b) Quadratic Probing

0	18	26_1
1	10	
2		
3	26_2	
4		
5		
6		
7	16	7_0
8	7_1	26_0

2) (cont)
c) What is the **load factor** for the table a)? 5/9

d) What is the **load factor** for the table b)? 5/9

e) Table a) will (circle one):

i. [✓ gradually degrade in performance as more values are inserted]

ii. may fail to find a location on the next insertion

iii. none of the above

e) Table b) will (circle one):

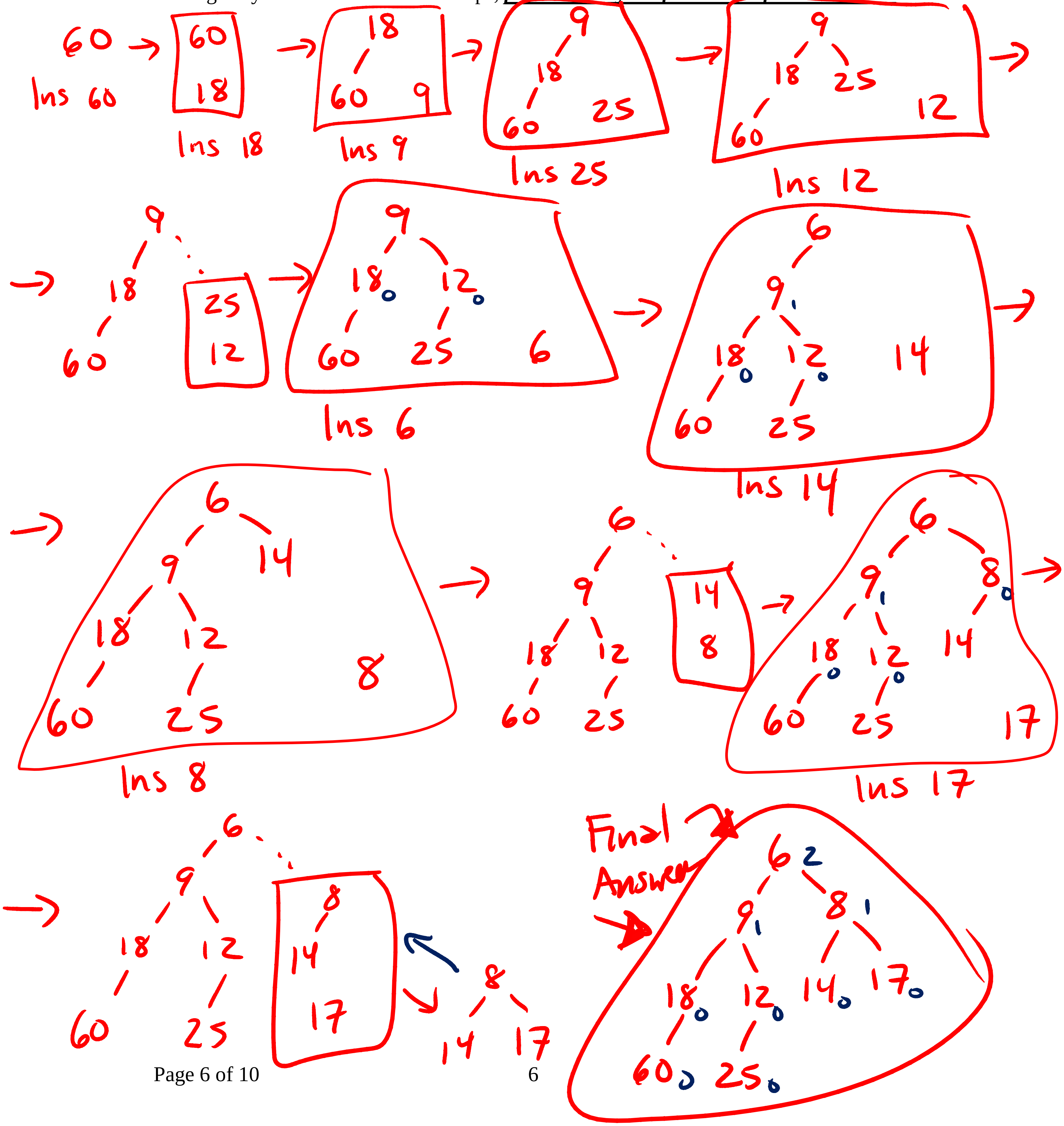
i. gradually degrade in performance as more values are inserted

ii. [✓ may fail to find a location on the next insertion]

iii. none of the above

3) [13 points total] Leftist Heaps:

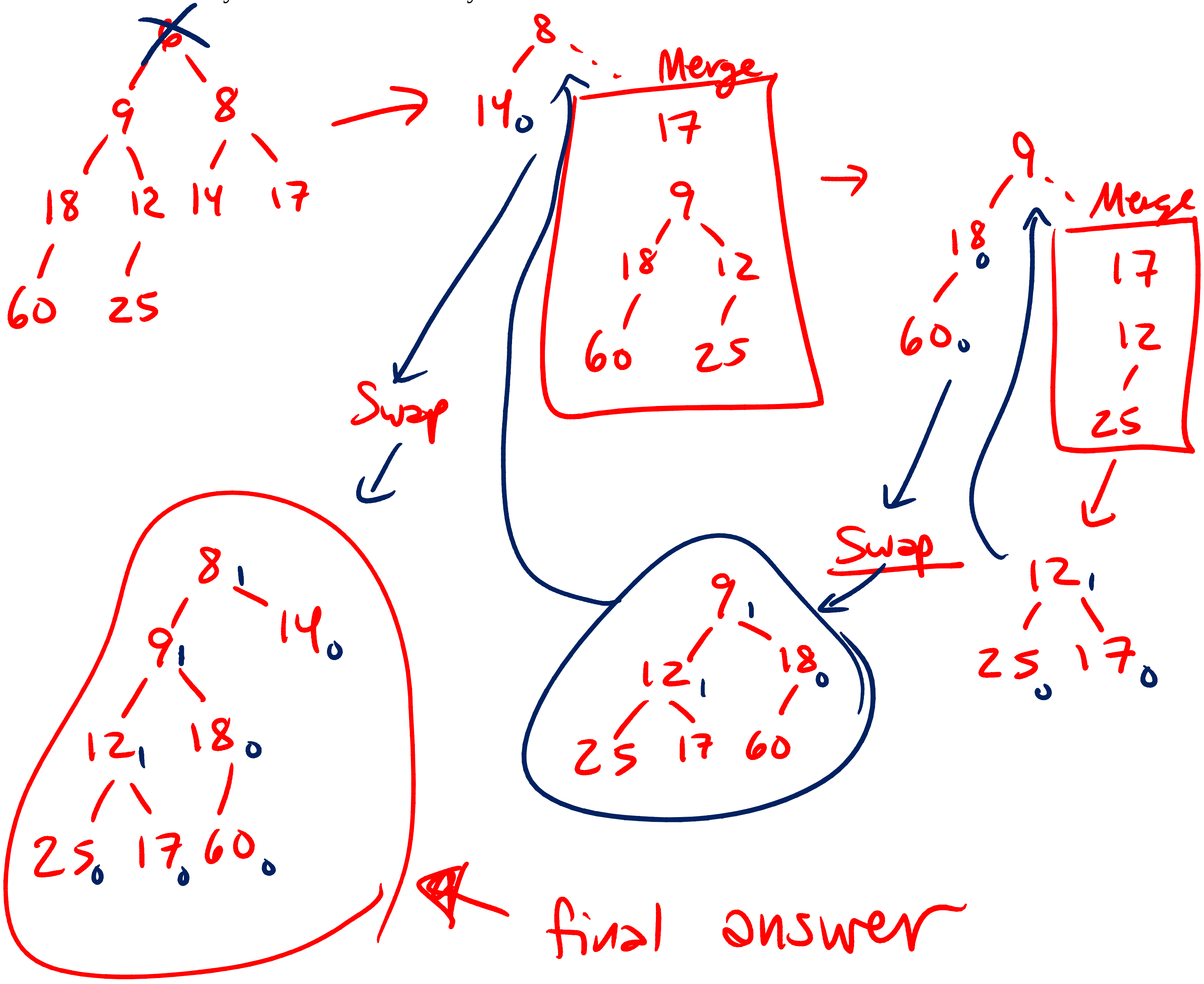
a) [8 pts] Draw the ***leftist*** heap that results <u>from inserting: ~~60, 18, 9, 25, 12, 6, 14, 8,~~ 17</u> <u>in that order</u> into an initially empty heap. You are only required to show the final heap, although if you draw intermediate heaps, ***<u>please circle your final result for ANY credit</u>***.



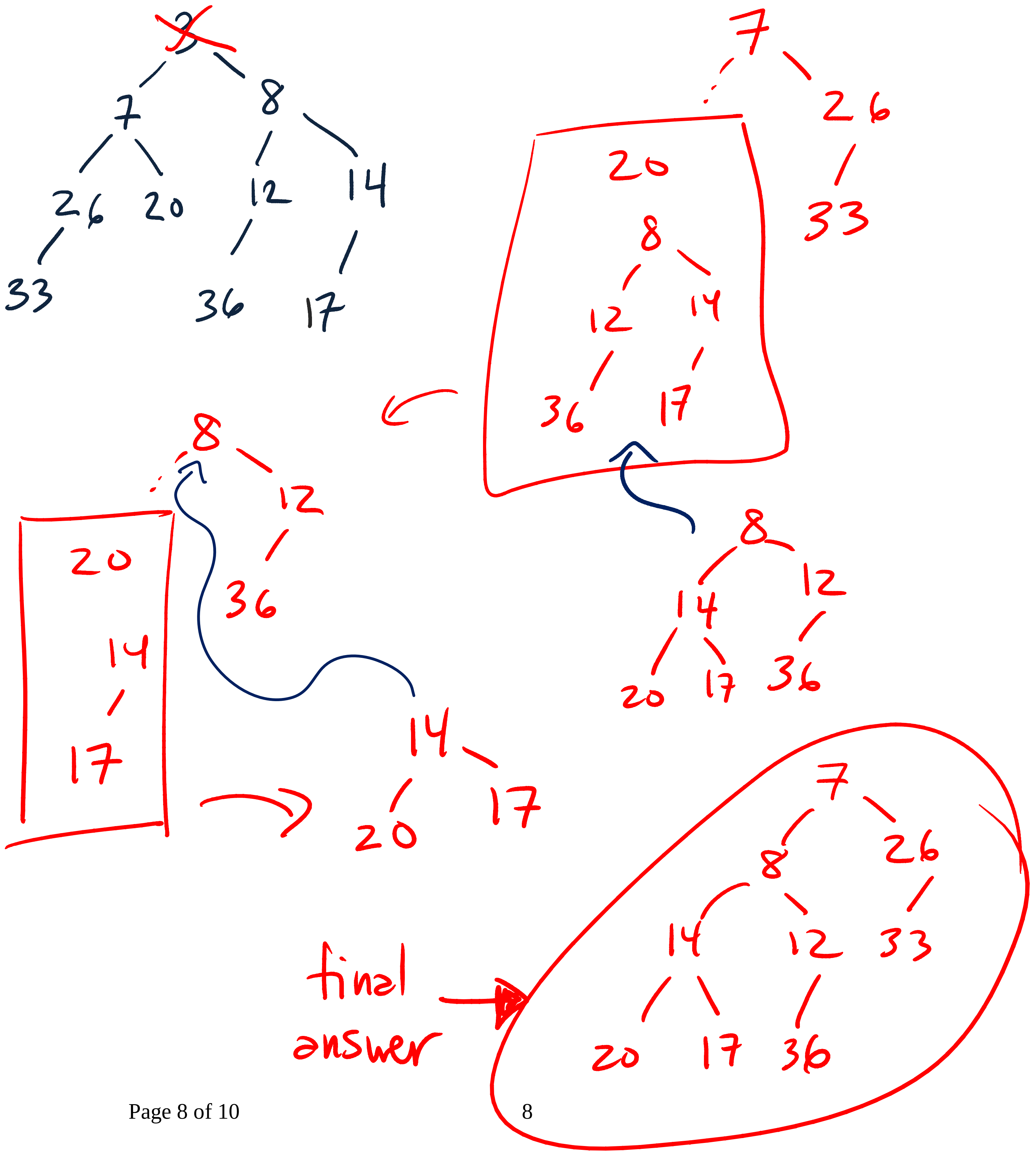
3) (cont)
b) [2 pts] What is the null path length of the **root** in your final heap from part a)?

2

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.



4) [6 points] 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.***



5) [8 points] Memory Hierarchy & Locality: Examine the code example below:

```
x = 7;
y = 0;
z = 400;
for (i = 1; i < 1000; i++) {
     y = y + 5;
     a[i] = a[i] + a[i+1];
     b[5] = b[5] + y;
     w += c[i] + 10;
     System.out.println(”z = ” + z);

}
```

Considering only their use in the code segment above, for each of the following variables, indicate below what type of locality (if any) is demonstrated. Please circle ***all that apply*** (you may circle more than one item for each variable):

a	[✓ spatial locality]	[✓ temporal locality]	no locality
b	spatial locality	[✓ temporal locality]	no locality
c	[✓ spatial locality]	temporal locality	no locality
i	spatial locality	[✓ temporal locality]	no locality
w	spatial locality	[✓ temporal locality]	no locality
x	spatial locality	temporal locality	[✓ no locality]
y	spatial locality	[✓ temporal locality]	no locality
z	spatial locality	[✓ temporal locality]	no locality

6) [16 points total] 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) – f)):

$O(N^2)$, $O(N^{½})$, $O(N \log N)$, $O(N)$, $O(N^2 \log N)$, $O(N^5)$, $O(2^N)$, $O(N^3)$, $O(\log N)$, $O(1)$, $O(N^4)$, $O(N^N)$, $O(N^6)$, $O(N (\log N)^2)$, $O(N^2 (\log N)^2)$

****For any credit, you must explain your answer.** *Assume that the most time-efficient implementation is used.* Assume no duplicate values and that you can implement the operation as a member function of the class – with access to the underlying data structure. **If you remove elements from a structure you must re-assemble it when done.**

a) Merging two **skew heaps** containing N elements each. **Explanation:**

The worst case length/height of the right hand path is O(N) in a skew heap. Since all operations are on the right path, WC for merge is O(N).

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

b) Finding an element in a **hash table** containing N elements where separate chaining is used and each bucket points to an AVL tree. The table size = N. **Explanation:**

Worst case is all N values hashed to the same bucket. Takes constant time to find the correct bucket, then do a find on the bucket. If the bucket is an AVL tree, Find is WC O(log N).

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

c) Finding the *median* value in a **leftist heap** containing N elements. (You don't know what the median value is ahead of time.) You may assume N is odd. **Explanation:**

Do N/2 deletemins. The median value is then at the root of the heap. Deletemin is O(log N) WC so $\frac{N}{2}\log N \rightarrow O(N \log N)$. You should also then re-insert the $\frac{N}{2}$ values you removed, Insert is O(log N) so this just adds another O(N log N) so overall O(N log N).

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

d) Moving the values from ***two* leftist heaps,** each containing N elements, into ***one*** initially empty array of size 2N. The final contents of the array should be sorted from low to high. **Explanation:** Do a deletemin on each heap, compare the two values, (lets say the smaller of the two values comes from heap A) write the smaller of the two values into the array. Now do another deletemin on heap A. Compare that value to the value you previously deleted from heap B. Write the smaller of those two values into the array + delete another value from the heap that value came from. Repeat deletemins (total of 2N of them) + comparisons until all values have been deleted. 2N deletemins total = 2N log N = O(N log N)

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