Name: Key

Email address:

CSE 373 Spring 2008: 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: 67 points. Time: 50 minutes.

Question	Max Points	Score
1	15	
2	8	
3	12	
4	12	
5	8	
6	6	
7	6	
Total	67	

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) Insert an element into a **hash table** of tablesize N, containing N elements where separate chaining (each bucket points to an unsorted linked list) is used.

Explanation: Even if all values are in one bucket, insert just requires doing the hash function to find the correct bucket (O(1)) + inserting at front of an unsorted list (O(1)).
[Find could take O(N) in worst case]

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

b) Inserting an element into a **binary min heap** (implemented using an array) containing N elements.

Explanation: Add element at end of array (O(1)), then call percolate up. For a complete tree, height is O(logN). So at most O(logN) comparisons/swaps are required.

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

c) Merging two **leftist heaps**, each containing N elements.

Explanation: In leftist heaps, the right paths are guaranteed to be at most length O(logN). Merging performs all work on the right paths, (where "work" = checking NPL + swapping if needed). So total work is at most O(logN).

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

d) Deleting the minimum value from a **skew heap** containing N elements.

Explanation: In skew heaps we have no worst case guarantee on path lengths, so we could have paths of length O(N). "work" is similar to leftist heaps, except no checking + updating NPL is needed. So total work is O(N)

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

e) Finding the *maximum* value in a **binary min heap** (implemented using an array) containing N elements.

Explanation: Need to examine values in the "bottom row" of the heap (= ~$\frac{1}{2}$N values). (Even if we examine all values in the heap (why ever look @ your parent though?), still takes time N.
So time is O(N)

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

2) [8 points] **Hashing**: Draw the contents of the hash table in the boxes below given the following conditions:

The size of the hash table is 9.
Open addressing and quadratic probing is used to resolve collisions.
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.

10, 35, 18, 19, 26

0	18_0	26_1
1	10_0	19_0
2	19_1	
3	26_1	
4		
5		
6		
7		
8	35_0	26_0

10 mod 9 = 1

35 mod 9 = 8

18 mod 9 = 0

19 mod 9 = 1

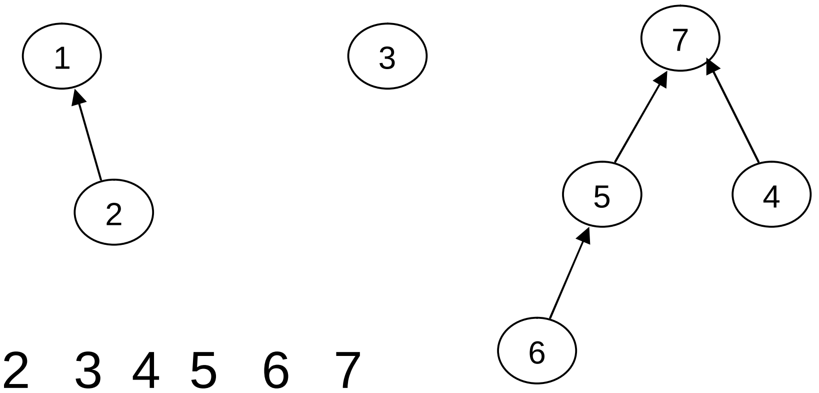
$+1 \to 2$

26 mod 9 = 8

$+1 \to 9 \bmod 9 = 0$

$+4 \to 12 \bmod 9 = 3$

3) [12 points total] **Disjoint Sets**: Given the array representation of up-trees discussed in class (where `size[x]` = total number of values in set x, and `up[x]` == -1 indicates that node x is a root node), for example (`size` is unspecified for non-root nodes):



	1	2	3	4	5	6	7
up	-1	1	-1	7	7	5	-1
size	2		1				4

a) [8 points] Fill in the remainder of the code below for **Union by Size** (weighted Union) and **Find** (without path compression) below (does not have to be perfect Java code).

```
int up[N];
int size[N];

void Union(int x, int y) {
     // Assuming x and y are the roots of two sets,
     // fill in code to implement union by size (weight)
     int wx = size[x];
     int wy = size[y];
    if (wx < wy) { // x should point to y
        up[x] = y;
        size[y] = wx + wy;
     } else {
        up[y] = x;
        size[x] = wx+wy;
     }
}
```

```
int Find(int x) {
     // Assuming x is an element, fill in code to
     // return the root of the set x belongs to.
```

```
//Iterative:

while ( up[x] != -1) {
     x = up[x];
}
return x;
```

```
// Recursive:
if (up[x] = -1){
     return x;
}else{
     return Find(up[x]);
}
```

```
}
```

b) [2 points] What is the **worst case** running time of a single **Union**-by-size (weighted Union) operation (assuming you are given roots as parameters as in your code above) where N = total # of elements in all sets?

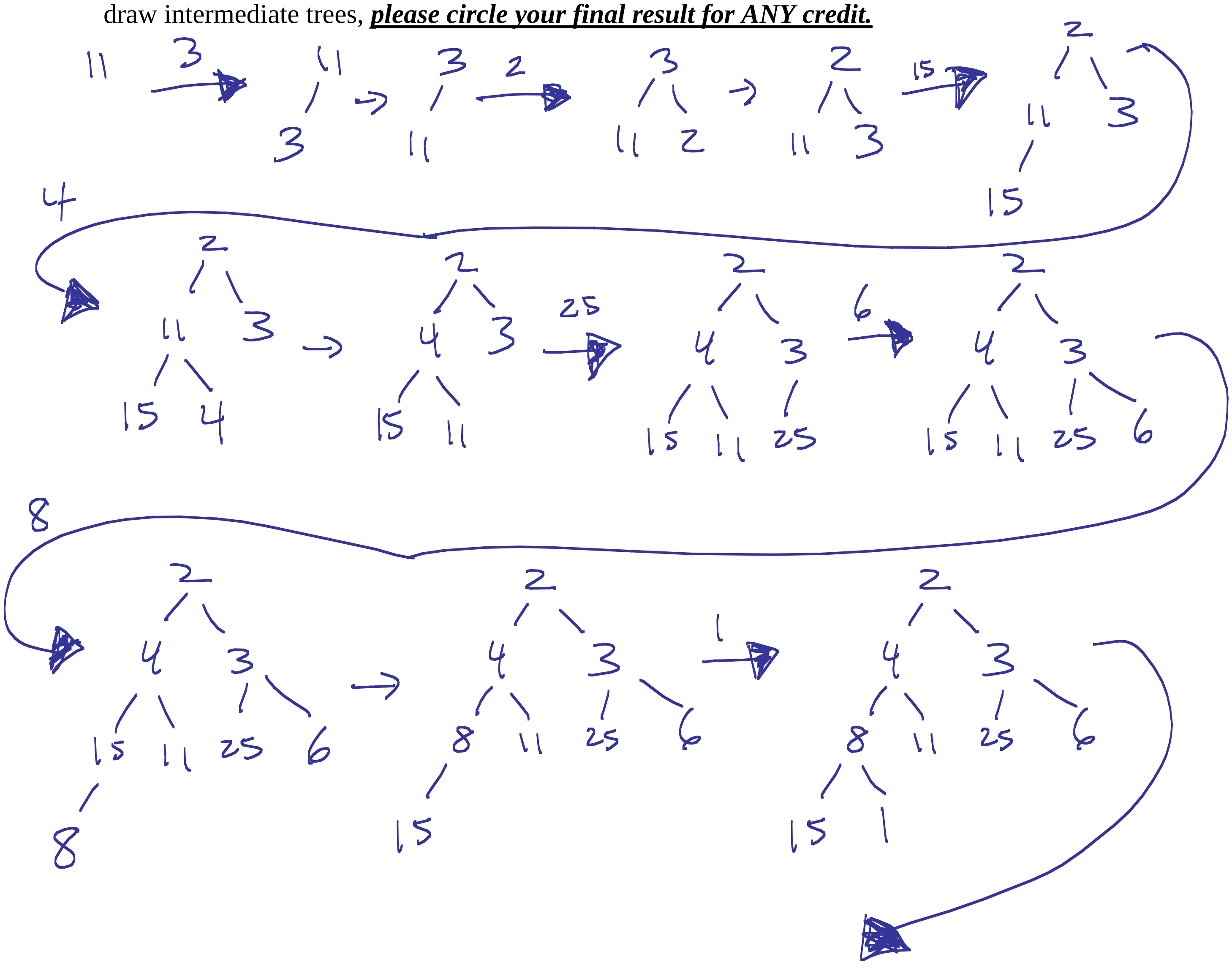
O(1) - If given roots, all operations are constant time!

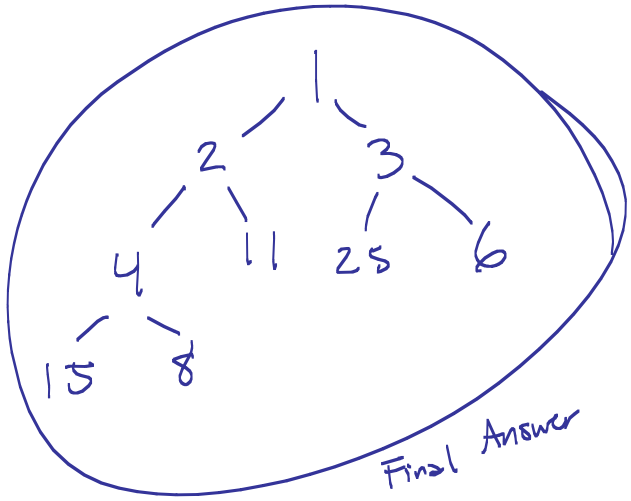
c) [2 points] What is the **worst case** running time of a single **Find** operation (without path compression), *assuming that Union-by-size has been used*, where N = total # of elements in all sets?

O(log N) - If Union-by-size is used, the depth of any node is never worse than O(log N)
(i.e. O(log N) recursive calls or O(log N) loop iterations in the code above.)

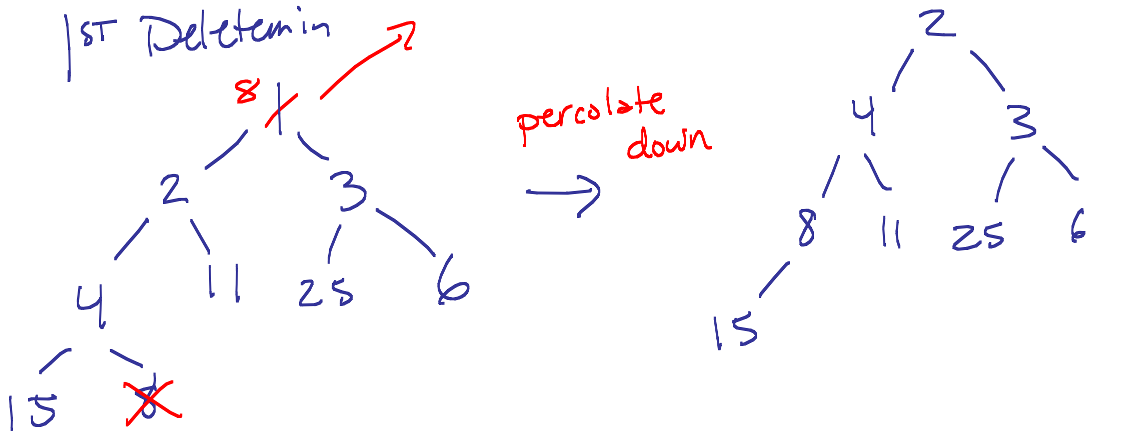
4) [12 points total] **Binary Min Heaps**

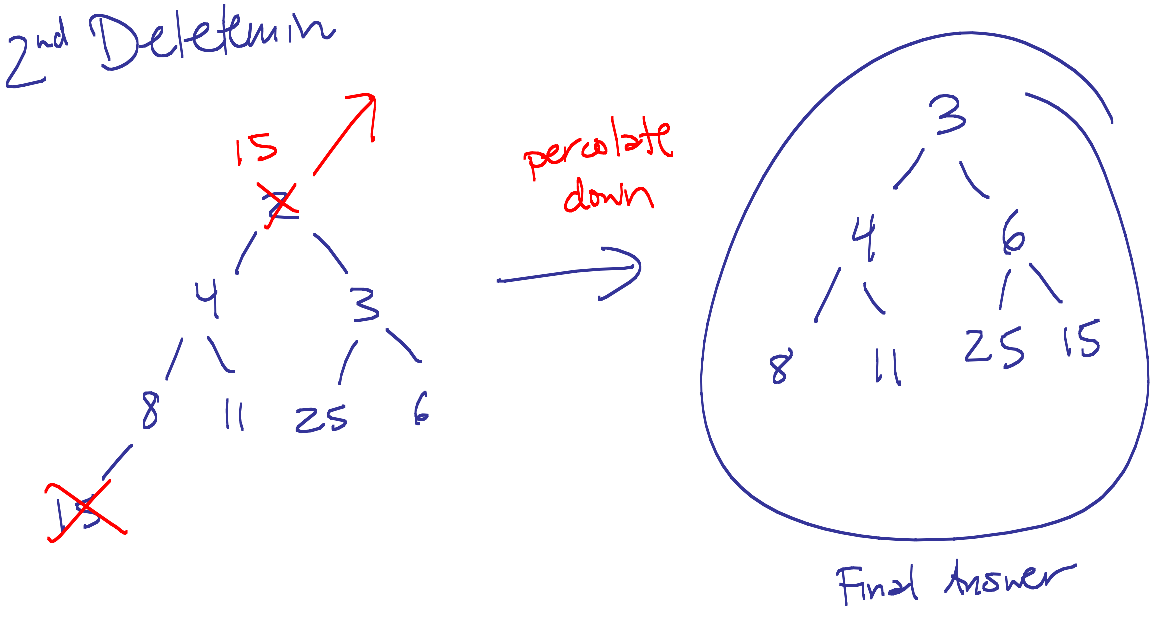
a) [8 points] Draw the binary min heap that results from inserting 11, 3, 2, 15, 4, 25, 6, 8, 1 in that order into an **initially empty binary heap**. You do not need to show the array representation of the heap. You are only required to show the final tree, although if you draw intermediate trees, ***please circle your final result for ANY credit.***





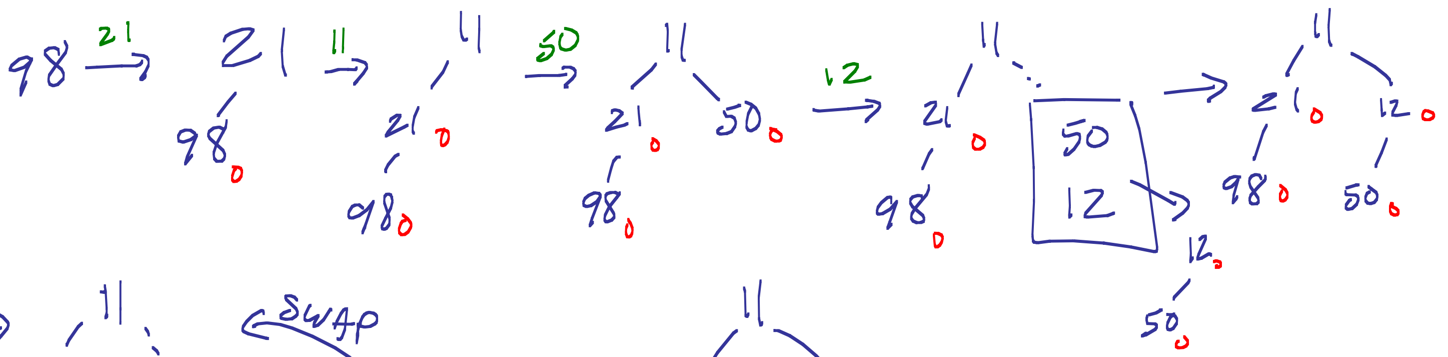
b) [4 points] Draw the result of doing 2 deletemins on the heap you created in part a. Please show what the tree would look like *after each deletion*. ***In addition, please circle your final result for ANY credit.***

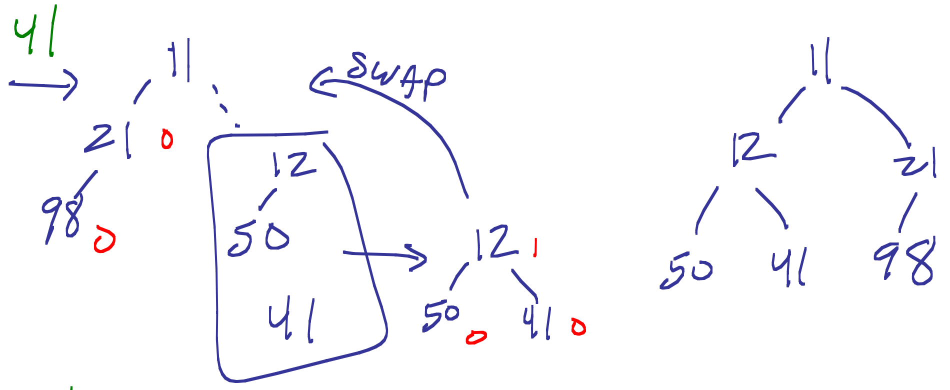


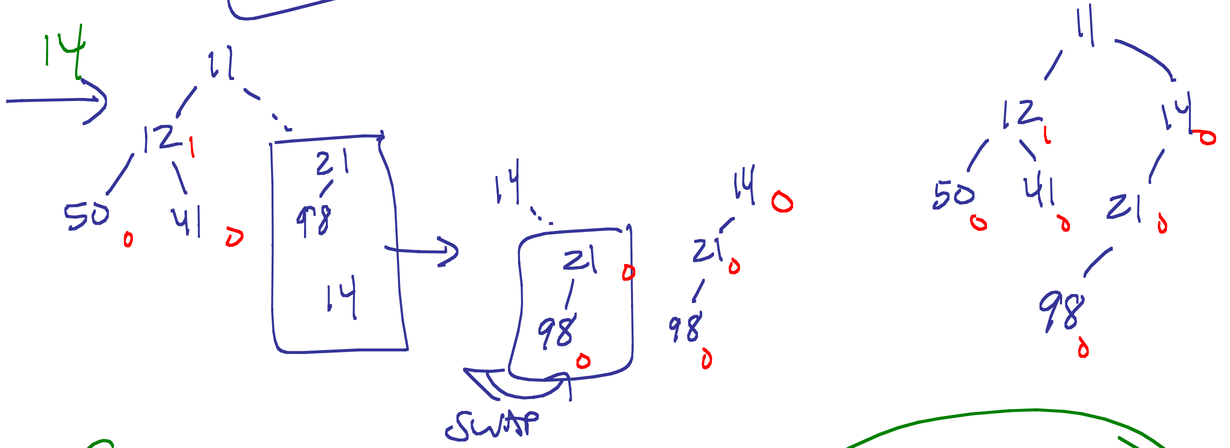


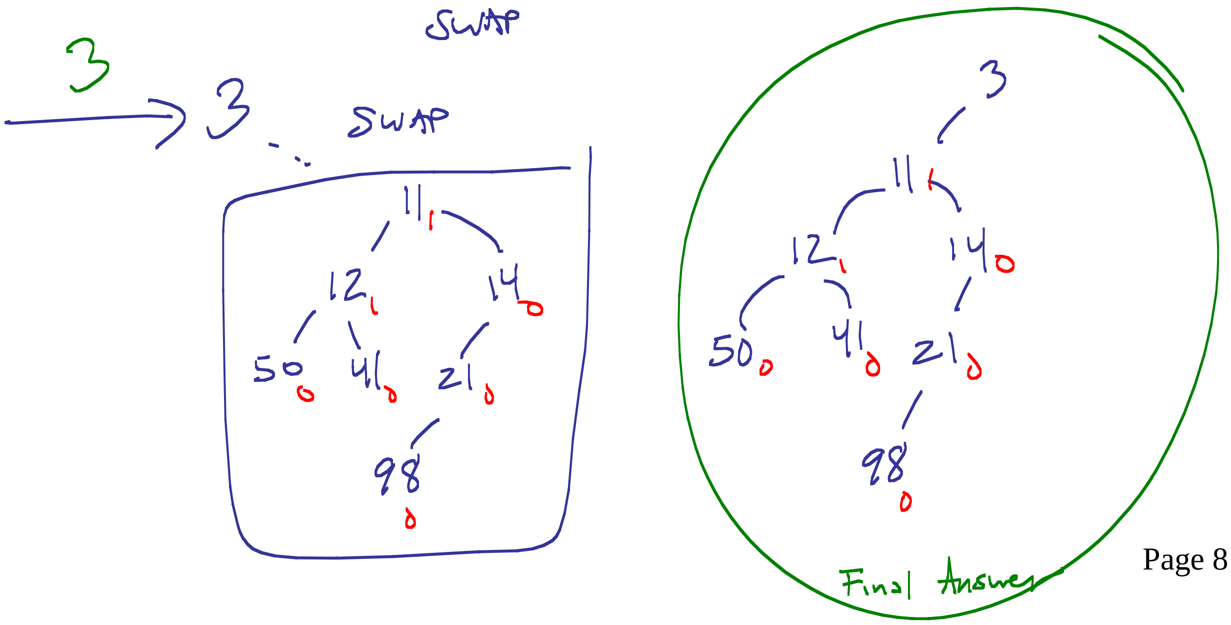
(NPL in Red)

5) [8 points] **Leftist Heaps**:
Draw the ***leftist*** heap that results from inserting: 98, 21, 11, 50, 12, 41, 14, 3 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.***



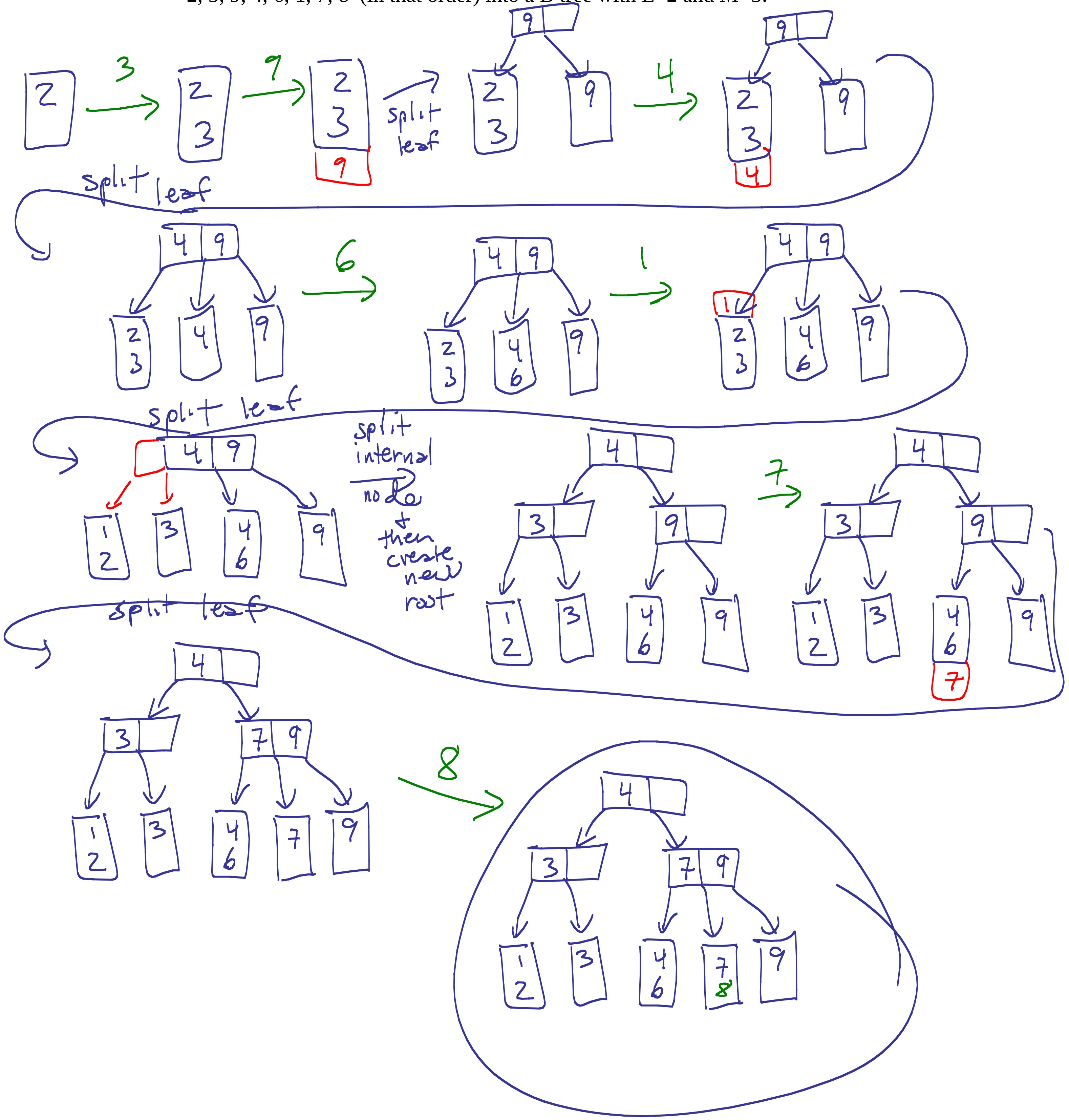






Note: other answers are possible depending on where values go when a leaf node splits. Both leaves must be half full, however.

6) [6 points total] **B-trees**:
Using the method described in class and in the book, insert the values:
2, 3, 9, 4, 6, 1, 7, 8 (in that order) into a B tree with L=2 and M=3.



7) [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.]

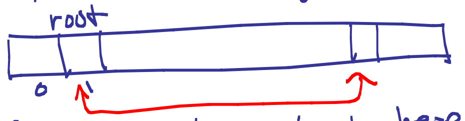
Option 1 [6 points]: Describe how **D-heaps** can have both spatial and temporal locality (on data). Be specific about *what* exactly in the D-heap has locality and *when* (on what operations).

Spatial locality on data:

On a deletemin, we replace root w. a new value & percolate down. When pecolating down, you must examine all D children to find the smallest. All of these children will be next to each other in memory, so you should get spatial locality when accessing them sequentially (the first child might be a cache miss, but its siblings should be cache hits.)

Temporal Locality on data:

Deletemin operations always access the first & last elements in the heap:



If you do several deletemins without much percolation, the root in a row (either with a short heap, or is likely to remain in cache. If you just did an insert before the deletemin, the last value in the heap is likely to still be in cache.

(Binary min heaps would have a similar locality effect.)

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 (i.e. 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.