## 13sp

**1. (18 pts) Big-Oh**

(2 pts each) For each of the functions *f(N)* given below, indicate the tightest bound possible (in other words, giving $O(2^N)$ as the answer to every question is not likely to result in many points). Unless otherwise specified, all logs are base 2. **Your answer should be as "tight" and "simple" as possible.**

You do not need to explain your answer.

a) $f(N) = N \log N + N \log \log N$ $O(N \log N)$

b) $f(N) = 50 \log N^2 + 100 (\log N)^2$ $O(\log^2 N)$

c) $f(N) = N \log_2(4^N)$ $O(N^2)$

d) *Push in a **stack** containing N elements implemented using linked list nodes (worst case)* $O(1)$

e) A *preorder traversal in a **binary search tree** containing N elements (worst case)* $O(N)$

f) *Insert in a **separate chaining hash table** containing N elements where each bucket points to an AVL tree (worst case)* $O(\log N)$

g) *IncreaseKey(k, v) on a **binary min heap** containing N elements. Assume you have a reference to the key k. v is the amount that k should be increased. (worst case)* $O(\log N)$

h) $T(N) = T(N-1) + N$ $O(N^2)$

i) $T(N) = T(N/2) + 100$ $O(\log N)$

## 13sp

**2. (6 pts) Big-Oh and Run Time Analysis:** Describe the worst case running time of the following pseudocode functions in Big-Oh notation in terms of the variable `n`. Your answer should be as "tight" and "simple" as possible.

***Showing your work is not required***

Runtime:

```
 I. int sunny (int n) {
        if (n < 10)
           return n - 1;
        else {
           return sunny (n / 2);
        }
    }
```

$O(\log n)$

```
II. int funny (int n, int sum) {
        for (int k = 0; k < n * n; ++k)
           for (int j = 0; j < k; j++)
              sum++;
        return sum;
      }
```

$O(n^4)$

```
III. int happy (int n, int sum) {
        for (int k = n; k > 0; k = k - 1) {
           for (int i = 0; i < k; i++)
              sum++;
           for (int j = n; j > 0; j--)
              sum++;

        }
        return sum;
    }
```

$O(n^2)$

**6. (9 pts) Recurrences**

Give a base case and a recurrence for the runtime of the following function. Use variables appropriately for constants (e.g. $c_1$, $c_2$, etc.) in your recurrence (you do not need to attempt to count the exact number of operations). **YOU DO NOT NEED TO SOLVE** this recurrence.

```
int onion(int n) {
  if (n < 10) {
    return n * n;
  }
  else {
    for (int i = 0; i < n; i++) {
      print "Keep trie-ing!";
      print "Onions rule!"
    }
    return n * onion(n / 3) + 10 * onion(n / 3);
  }
}
```

$T(n) = c_1$ For n < 10

$T(n) = c_2 + c_3 \cdot n + 2 \cdot T\left(\frac{n}{3}\right)$ For n >= 10

**Yipee!!!! YOU DO NOT NEED TO SOLVE** ***this*** recurrence…

**7. (10 pts) Solving Recurrences**

Suppose that the running time of an algorithm satisfies the recurrence relationship:

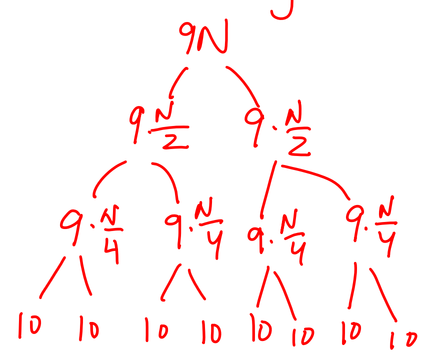
$$T(1) = 10.$$

and

$$T(N) = 2 * T(N/2) + 9N \qquad \text{for integers } N > 1$$

Find the closed form for T(N). **You may assume that N is a power of 2.** Your answer should *not* be in Big-Oh notation – show the relevant *exact* constants and bases of logarithms in your answer (e.g. do NOT use "$c_1$, $c_2$" in your answer). You should not have any summation symbols in your answer. The list of summations on the last page of the exam may be useful. ***You must show your work to receive any credit***.

$N = 8 \qquad h = \log_2 N$




$$T(N) = 2\cdot T\left(\frac{N}{2}\right) + 9\cdot N$$

$$= 9\cdot N \cdot \sum_{i=0}^{\log_2 N - 1} 2^i \cdot \frac{1}{2^i} + 10\cdot N$$

$$= 9\cdot N \cdot \sum_{i=0}^{\log_2 N - 1} 1 + 10N$$

$$= 9N\cdot\log_2 N + 10N$$

---

$$T(N) = 2\cdot T\left(\frac{N}{2}\right) + 9\cdot N$$

$$= 2\left[2\cdot T\left(\frac{N}{4}\right) + 9\cdot\frac{N}{2}\right] + 9N$$

$$= 2\left[2\cdot\left[2\cdot T\left(\frac{N}{8}\right) + 9\cdot\frac{N}{4}\right] + 9\cdot\frac{N}{2}\right] + 9\cdot N$$

$$= 2^3\cdot T\left(\frac{N}{8}\right) + 2^2\cdot 9\cdot\frac{N}{4} + 2\cdot 9\frac{N}{2} + 9N$$

$$= 2^3\cdot T\left(\frac{N}{2^3}\right) + 9N + 9\cdot N + 9\cdot N$$

$$= 2^3\cdot T\left(\frac{N}{2^3}\right) + 3\cdot 9N$$

$$= 2^k\cdot T\left(\frac{N}{2^k}\right) + k\cdot 9N$$

$$\frac{N}{2^k} = 1 \text{ when}$$

$$N = 2^k$$

$$k = \log_2 N$$

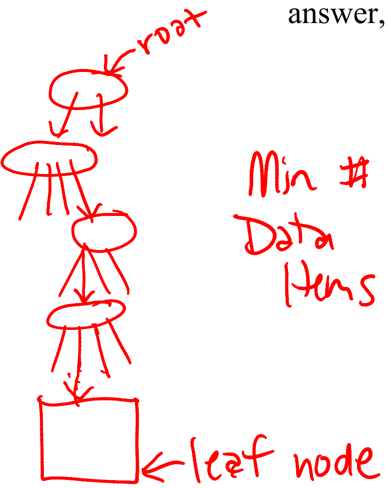
$$= 2^{\log_2 N}\cdot T(1) + \log_2 N\cdot 9N$$

$$= N\cdot 10 + 9N\cdot\log_2 N$$

**16au**

**8. (10 pts) B-Trees**

a) (5 pts) Given M=7 and L=20, what is the minimum number of data items in a B-Tree (as defined in lecture and in Weiss) of height 4? Give a single number for your answer, not a formula. **Explain briefly how you got your answer.**




$$\text{Min \# Data Items} = \underbrace{2}_{\text{Min \# pointers in root}} \cdot \underbrace{4 \cdot 4 \cdot 4}_{\text{Min \# pointers in other interior nodes}} \cdot \underbrace{10}_{\text{Min \# data items in a leaf}}$$

$$= 2^7 \cdot 10 = 128 \cdot 10 = 1280$$

b) (5 pts) Given the following parameters for a B-tree with M= 22 and L = 5:

Key Size = 8 bytes

Pointer Size = 4 bytes

Data Size = 50 bytes per record (*includes* the key)

Assuming that M and L were chosen appropriately, **what is the likely size of a disk block** on the machine where this implementation will be deployed? Give a numeric answer and **a short justification *based on two equations*** using the parameter values above.

$$\underbrace{50}_{\text{Data Size}} \cdot \underbrace{5}_{L} = 250 \text{ bytes} \le b$$

$$\underbrace{22 \cdot 4}_{\text{pointers}} + \underbrace{21 \cdot 8}_{\text{keys}} \le b$$

$$88 + 168 \le b$$

$$256 \le b$$

Block size is most likely 256 bytes. Since block size must be at least 256 bytes to be an interior node, and at least 250 bytes to be a leaf, it must be at least 256 bytes (and probably not more or else we would have made M larger - we need 12 more bytes to get to M=23).



Since 256 is also a power of 2, this is the likely block size.

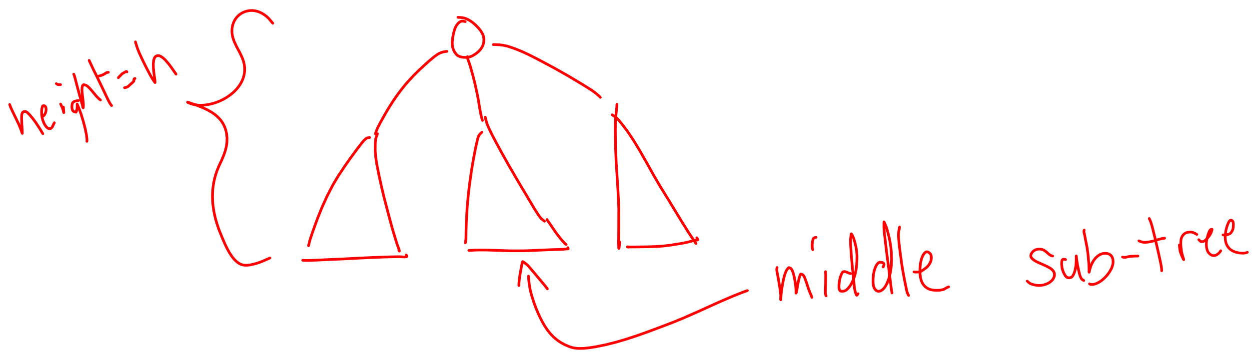
## 18au

**4. (8 pts) 3 Heaps**

Given a 3-heap of height h, what are the minimum and maximum number of nodes in the **middle sub-tree** of the root? Give your answer in closed form (there should not be any summation symbols).

Min nodes in middle sub-tree: $\frac{3^{h-1}-1}{2}$

Max nodes in middle sub-tree: $\frac{3^{h}-1}{2}$




Max Nodes: (max nodes in 3-heap of height h-1)

$$\sum_{i=0}^{h-1} 3^i = \frac{3^h-1}{2}$$

Min Nodes: (max nodes in a 3-heap of height h-2)

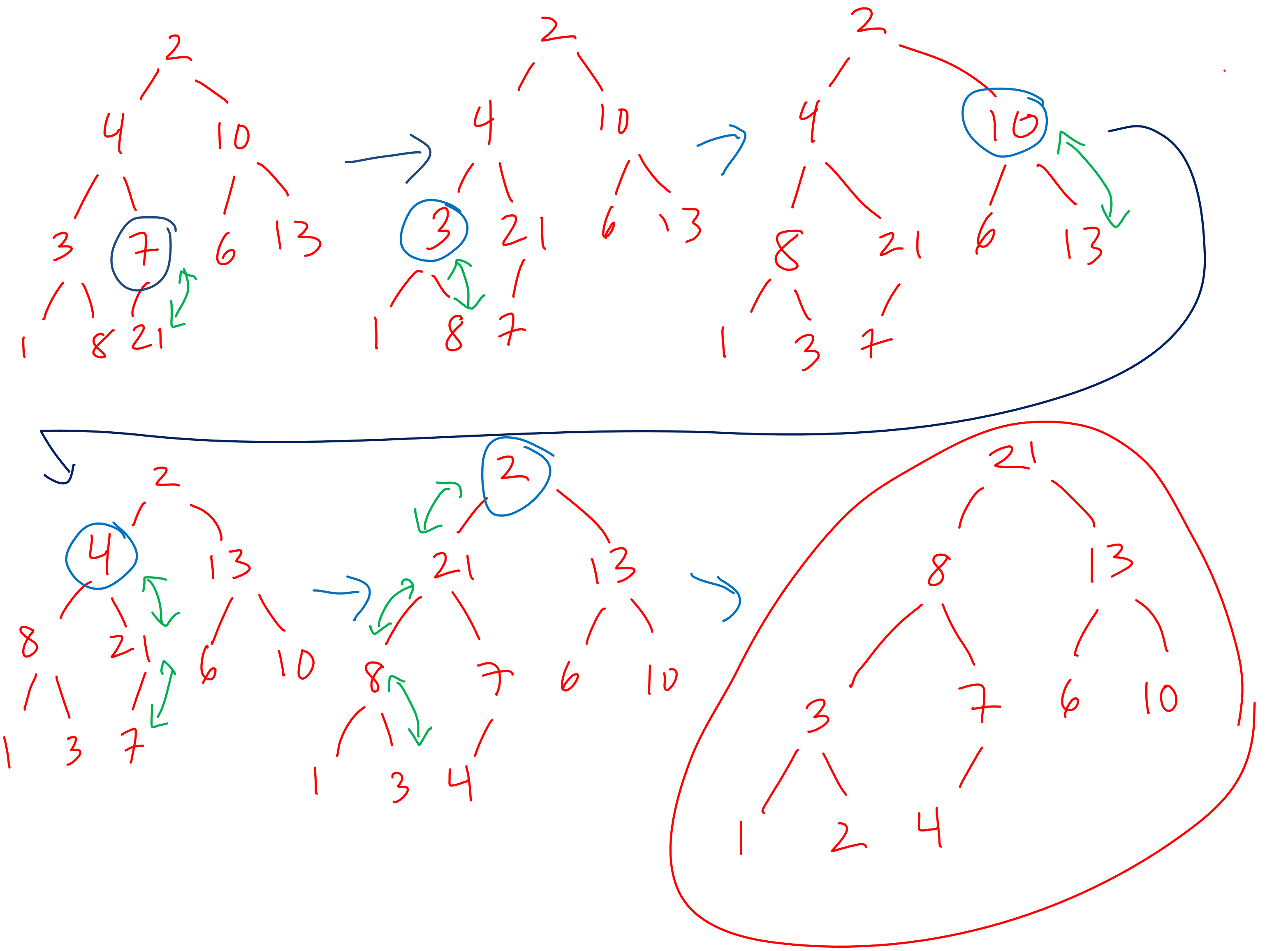
$$\sum_{i=0}^{h-2} 3^i = \frac{3^{h-1}-1}{2}$$

# 18au

**5. (10 pts) Binary Max Heaps**

Use Floyd's build heap to create a **Max** heap out of the following array. (Hint: a binary max heap would have the largest value at the root of the tree.) **For any credit, show your tree one step at a time.** You do not need to show the array. THIS IS A BINARY **MAX** HEAP!

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 10 | 3 | 7 | 6 | 13 | 1 | 8 | 21 |



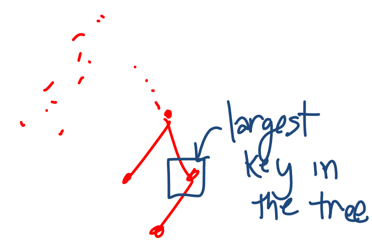
## 19wi

**5) (6 pts) AVL Trees**
You are given an AVL-Tree of height 5 and you are told what the largest key in the tree is. If you are doing a **pre-order traversal** of the tree, ***how many nodes in the tree will you visit before you encounter the largest key***? Give your answer as a ***single number*** (not a mathematical expression) but **explain your answer briefly**.

a) In the BEST case, what is the minimum number of nodes you would visit (include the largest key in your count of nodes)? **Explain your answer briefly**.

The minimum number of nodes in an AVL tree of height 5 is 20. However the right side of the tree could look like this:

$S(2) = 1 + 2 + 1 = 4$
$S(3) = 1 + 4 + 2 = 7$
$S(4) = 1 + 7 + 4 = 12$
$S(5) = 1 + 12 + 7 = 20$




Thus with a pre-order traversal you could stop as soon as you hit the largest key + not visit its left child. So a total of 19 nodes visited.

b) In the WORST case, what is the maximum number of nodes you would visit? (include the largest key in your count of nodes) **Explain your answer briefly**.

The maximum number of nodes in an AVL tree of height 5 would be in a perfect tree.

$$\sum_{i=0}^{5} 2^i = 2^6 - 1 = 64 - 1 = 63 \text{ nodes}$$

In the worst case you would need to visit all of these nodes