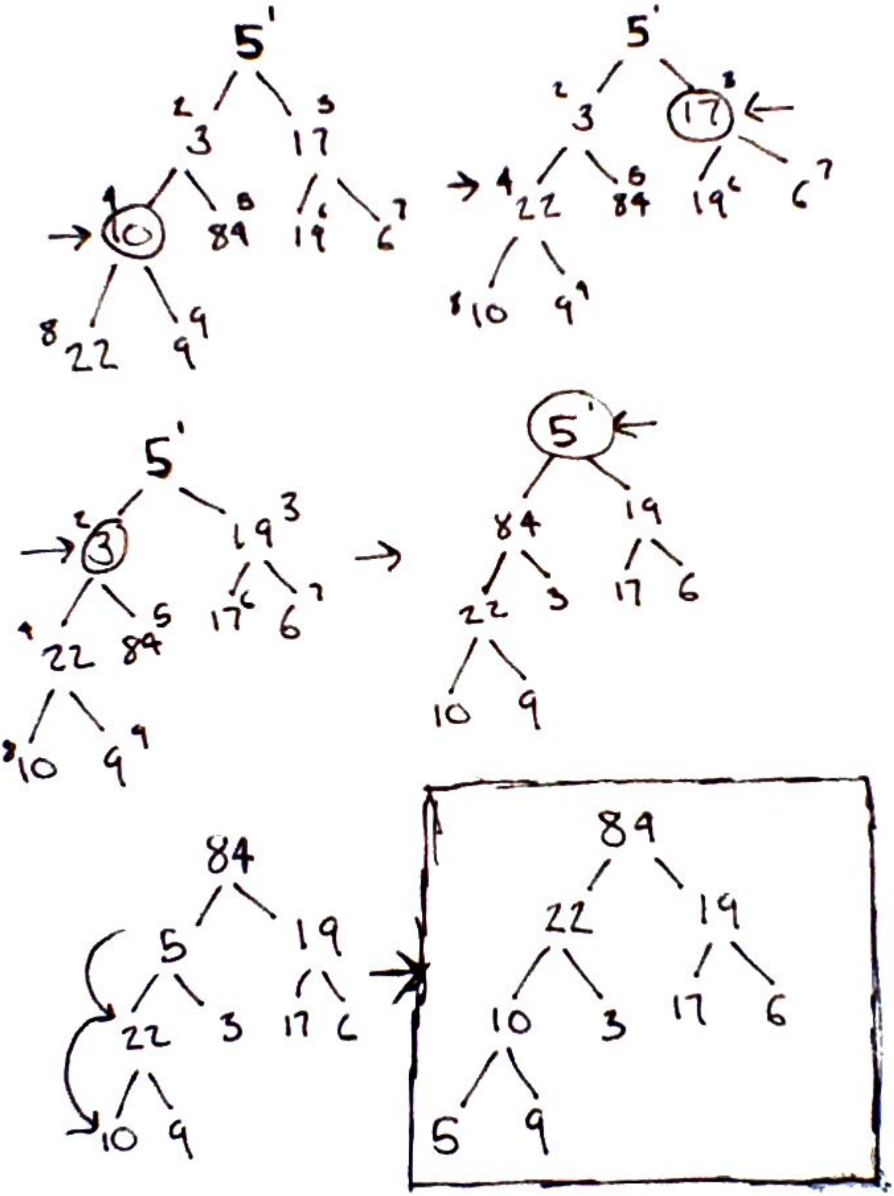
Assignment 2

Cole Pendergraft

1) 5 3 17 10 84 19 6 22 9



2) $O(n\log k)$ alg to merge k sorted lists. Use min-heap!

→ Build Min-Heap has a known complexity of $O(n)$

→ If we want to use Build min heap, we also will need min-heapify, which has a known complexity of $O(\log n)$

→ we have k sorted lists

1. We start by building our min-heap. If we use the first element of each of the k lists, we will have a heap size of k, which means build min heap will have complexity of $O(k)$. We now have a min-heap with the minimum at the root.

2. Now we put the minimum into the final sorted list. We accomplish this with heap-minimum, which has a complexity of $O(1)$. The root will then be replaced with the next element of the same list.

3. After this, our min-heap properties may have been violated, so we should call Min-Heapify on the root. This has a complexity of $O(\log k)$

4. Once the current list is empty, repeat from step 2 with a new sorted list.

Step 1 will only execute 1 time, so we have a complexity of $O(k)$ for that.

Step 2 and further will execute n times depending on the list sizes. Thus, step 2 has a complexity of $n \cdot O(\log k)$ and step 3 has a complexity of $n \cdot O(1)$.

So, we have an overall complexity of

$$O(k) + O(n) + O(n \log k)$$

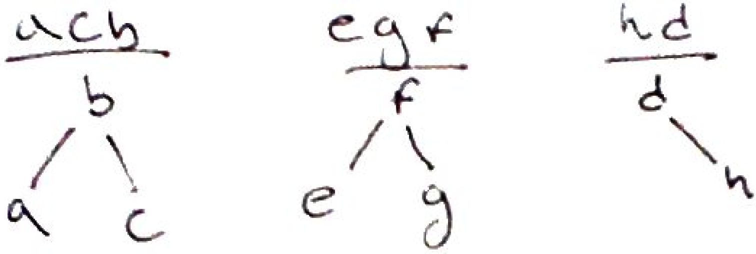
$$\boxed{= O(n \log k)}$$

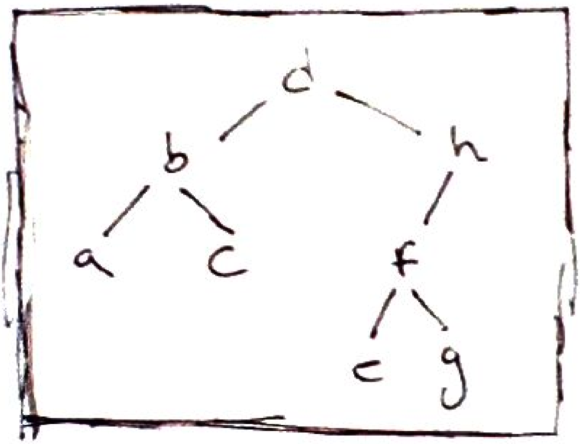
* Was stuck on this one so I did use online resources for help

3) Inorder: abcdef (left, root, right)

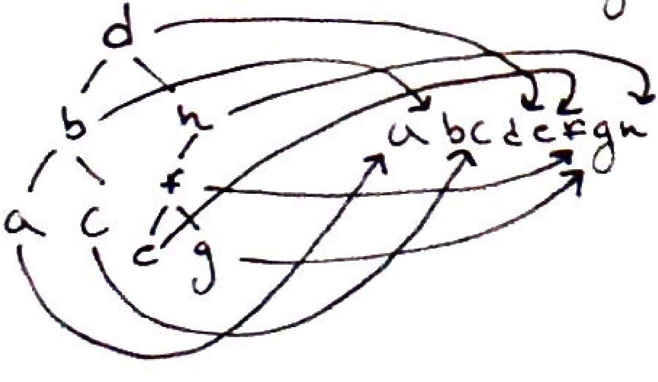
Postorder: acbegfhd (left, right, root)

*Pretty much just trial and error.





what is ~~the~~ inorder traversal of this? (left, root, right)



4) From ordered tree T, we get a binary tree T' using the first-child next-sibling as:

1. Each node u of T is associated with an internal node u' of T'

2. In the case where an external node u of T does not have any children, then the children of the node u' of T' must be external nodes

3. <u>First-Child</u>: For an internal node u of T, if v is the first child of u, then v' is the left child of u' in T'

4. <u>Next-Sibling</u>: For any sibling w that follows v, w' must be the <u>right</u> child of <u>v'</u> in T'.

From this, we can determine that an inorder traversal of T' is a <u>Postorder</u> traversal of T.

Why? In an inorder traversal, first the left subtree is traversed, then the root is visited, and lastly the right subtree is traversed.
With Postorder, we traverse the left subtr

then we traverse the right subtree, and lastly we visit the root. Since we build T' using first-child next-sibling, visiting the left subtree, then the root, and then the right subtree of T should be equivalent to visiting the left subtree, then the right subtree, and finally the root of T'. The opposite will also be true, so this means that an inorder traversal of T' is a postorder traversal of T.

* Another problem where I got stuck and had to use online resources as a guide.

5) * The process I use here is adapted from some code I have for computing insertion order perms.

Start: ~~5~~ ~~3~~ ~~9~~ ~~1~~ ~~4~~ ~~7~~ ~~10~~ 2 6 8

First root: **5**

R_1 (Elements ≥ 5) = 9, 7, 10, 6, 8

L_1 (Elements < 5) = 3, 1, 4, 2

size of $R_1 = 5$

size of $L_1 = 4$

Number of perms with 5 as the root:

$$\frac{(5+4)!}{(5)!\cdot 4!} = 126$$

Now repeat the process using L_1:

3 1 4 2

Root: 3

$R_2 (\geq 3)$: 4

$L_2 (< 3) = 1, 2$

size of $R_2 = 1$

size of $L_2 = 2$

number of perms of this seq with 3 as the root:

$$\frac{(1+2)!}{1!\cdot 2!} = \frac{3!}{2} = 3$$

Both L_2 and R_2 are too small to repeat the procedure, so instead we repeat using R_1

R_1: 9 7 10 6 8

Root: 9

$R_3 (\geq 9)$: 10

$L_3 (< 9)$: 7 6 8

size of $R_3 = 1$

size of $L_3 = 3$

Number of permutations of R_1 with root 9:

$$\frac{(3+1)!}{3! \cdot 1} = \frac{24}{6} = 4$$

R_3 is too small to repeat the procedure with, but we can do it on L_3:

$L_3 = 7\ 6\ 8$

Root $= 7$

$R_4 (\geq 7)$: 8

$L_4 (< 7)$: 6

size of $R_4 = 1$

Size of $L_4 = 1$

Number of Perms of L_3 with root 7:

$$\frac{(1+1)!}{1!\cdot 1!} = 2$$

All the remaining subsets we have not executed the procedure on (L_2, R_2, R_3, R_4, L_4) are all too small in size for the procedure to work, so at this point we are done with the algorithm.

Number of Perms of the vals 1 through 10 when inserted in that order yields the given tree:

$$\boxed{126 \cdot 3 \cdot 4 \cdot 2 = 3024}$$

6. Permutations of 1, 2, ... n that yield a skew tree.

A skew tree is the tree of maximum length. The max height is n-1. The tree can only be obtained if elements arrive in a specific order.

We have two possible skew trees from 1, 2, 3, ... n:

1. Left skew tree if elements arrive as n, n-1, n-2, ..., 1
2. Right skew tree if elements arrive as 1, 2, 3, ..., n

- If we have to use all n elements, only 2 skew trees are possible.

- However, if we <u>don't</u> have to use all n elements, but can take a subset, then any given subset will generate 2 possible skew trees, one in increasing order and one in decreasing order.

  - The total number of subsets with 2 or more elements is $2^n - n - 1$, (n subsets w/ 1 element, 1 subset with 0 elements)

Because there is only one tree per subset possible for single element subsets, we have that the

$$\text{total skew trees} = (2^n - n - 1) \cdot 2 + n$$
$$= 2^{n+1} - n - 2$$

* Again, I ended up needing the assitance of online resources for help