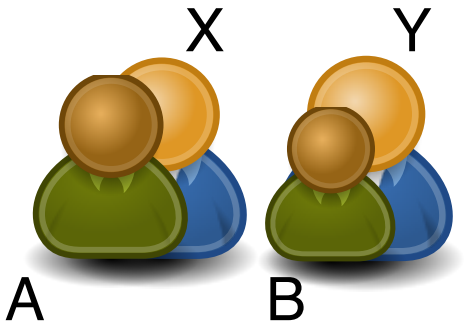
Homework 5 Solutions

Problem 1:

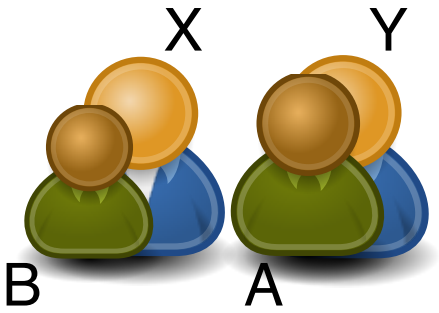
(a) There are $n!$ ways to order the first row and $n!$ ways to order the back row. There are two ways to place a team in front and the other one in the back. This gives a total of $2 \cdot n! \cdot n!$ arrangements. (The factor of 2 is arguable, as the problem can be read to say that one team has to be in the front.)

(b) Assume we have a valid arrangement. If we interchange a pair of players consisting of the front and the back player with another pair, then the new arrangement is also valid. Thus, we can order the back rank and still have a valid arrangement.

(c) Starting with any valid arrangement, we can order the back rank. Assume we have two pairs of players, A and B from the front-row team and X and Y from the back-row team. Assume that A is standing in front of X and B is standing in front of Y. Thus, $A < X$ and $B < Y$. Assume further that $X < Y$. If $B \leq A$, then we have the picture below.



First, $B \leq A$ and $A < X$, so $B < X$. Second, $A < X$ and $X \leq Y$, so $A < Y$. Thus, we can interchange the positions of players A and B and still have a valid arrangement.

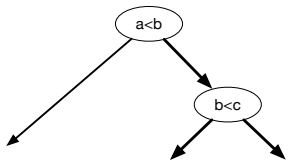


This means we can order the first row as well.

(d) Verifying all $n! \cdot n!$ or $2 \cdot n! \cdot n!$ possible arrangements takes super-exponential time. If we order the two teams by height we use time $\Theta(n \log(n))$ for sorting each team and time $\Theta(n)$ for comparing the heights of the two players in position i.

Problem 2:

(a) If we are given three elements a, b, c, then there are six possibilities for selecting the maximum and the minimum, namely abc; acb, bac, bca, cab, and cba, corresponding to all the permutations of the three elements. As seen in class, if two comparisons were possible to extract both maximum and minimum of the three elements, then we would have an algorithm that orders the three elements in a binary tree with two interior nodes. However, a binary tree with two interior nodes can only have three leaves, so it would not be able to distinguish the six possible cases.



We can easily write an algorithm with three comparisons per runs.

```
def minmax(a,b,c):
      if a<b<c:
            return a,c
      elif a<c<b:
            return a,b
      elif b<a<c:
            return b,c
      elif b<c<a:
            return b,a
      elif c<a<b:
            return c,b
      elif c<b<a:
            return c,a
```

(b) Assume $n \equiv 0 \pmod 3$, so $n = 3m$ for an integer m. This gives us $3m$ comparisons within each group, and then $m - 1$ comparisons to find the maximum of the group maxima and $m - 1$ comparisons to find the minimum of the group minima. This gives us a total of $5m - 2 = \frac{5n}{3} - 2$ comparisons in this case.

Assume now $n \equiv 1 \pmod 3$, so that $n = 3m + 1$. We still have $3m$ comparisons within each group, but now have $m + 1$ possible maxima and $m + 1$ possible minima, so that we have now $5m = \frac{5(n-1)}{3} = \frac{5n}{3} - \frac{5}{3}$ comparisons.

Assume now $n \equiv 2 \pmod 3$, so $n = 3m + 2$ for an integer m. There are $3m$ comparisons within the groups and one comparison for the remaining two elements. As before, there are m

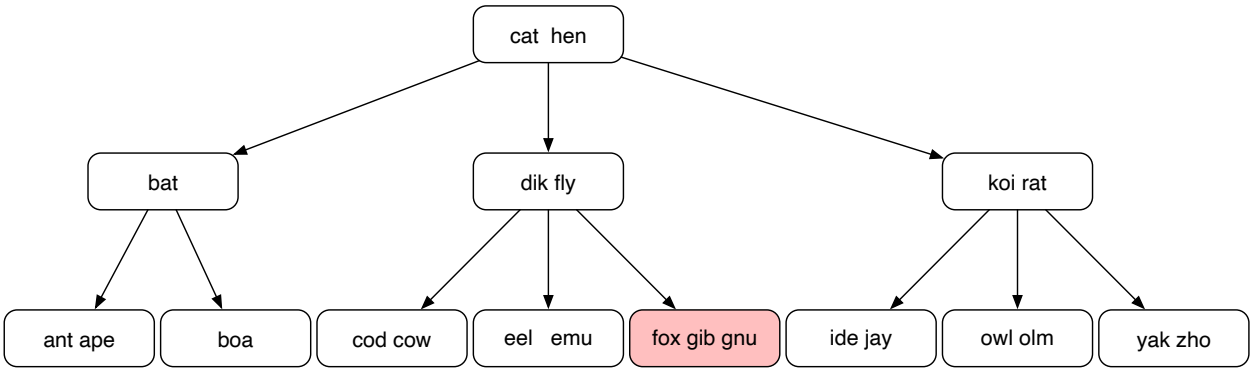
comparisons to determine the maximum of the group maxima and as well m comparisons to determine the minimum of the group minima. This gives a total of

$$3m + 1 + m + m = 5m + 1 = \frac{5(n-2)+3}{3} = \frac{5n}{3} + \frac{7}{3}$$

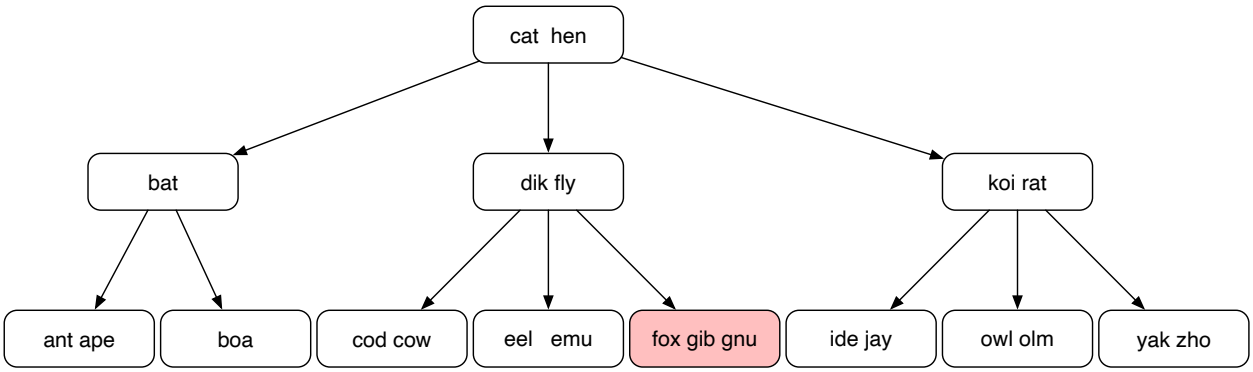
comparisons.

Problem 3:

Inserting gib

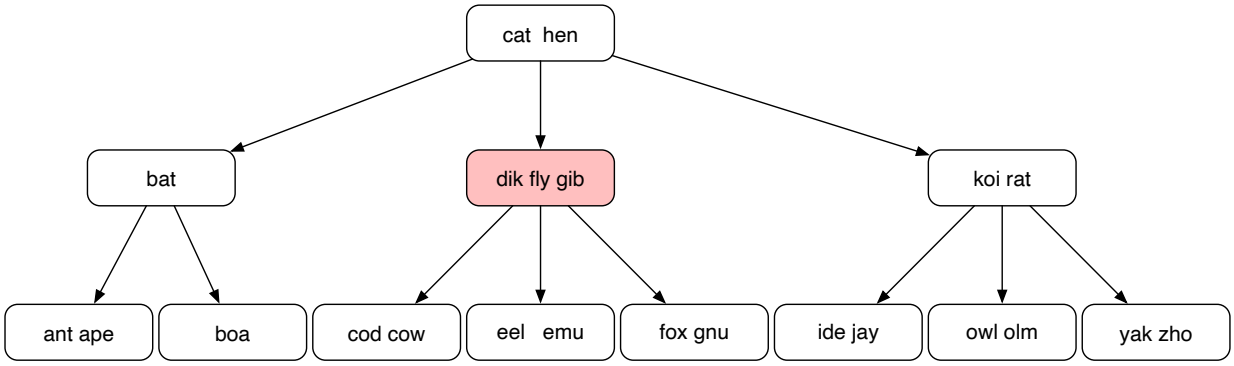


Overflow. Rotates are impossible, split.

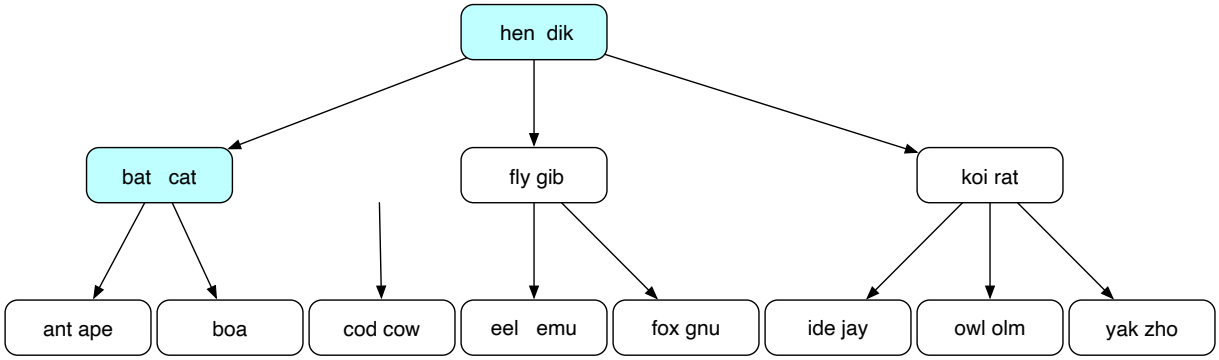


Overflow. Only left rotate is possible.

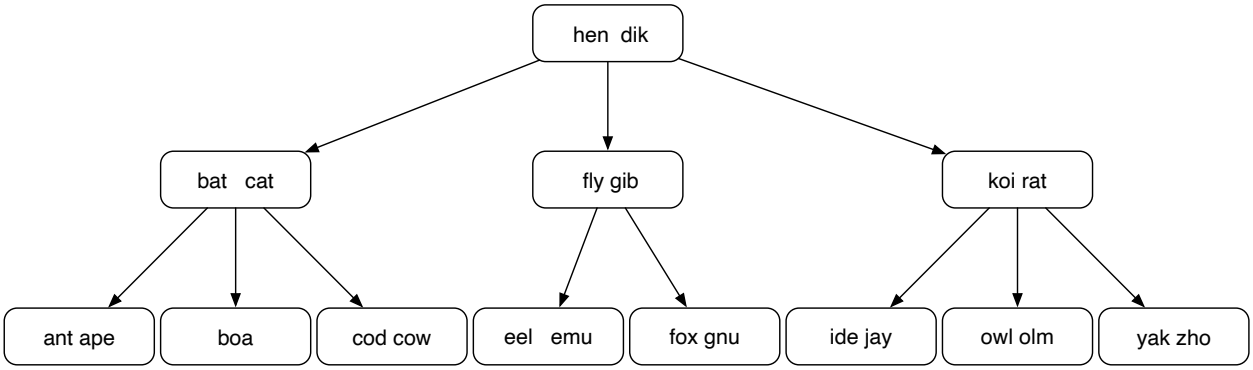
"dik" goes up, "cat" goes down



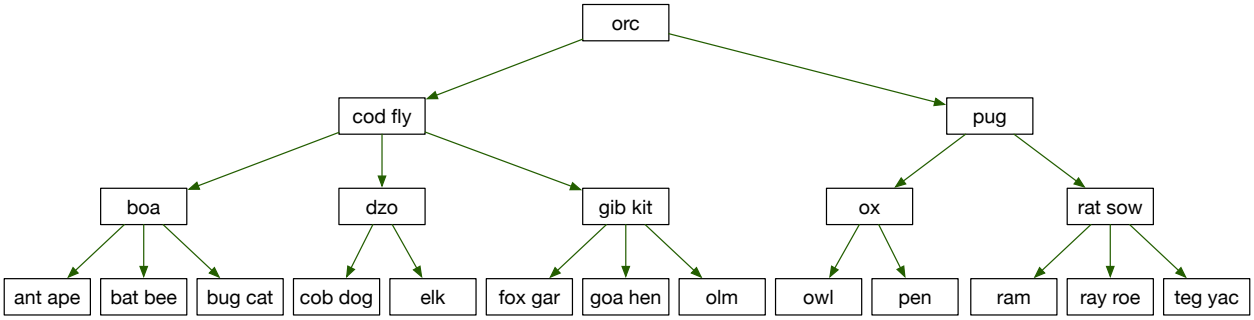
Need to reattach the dangling pointer.



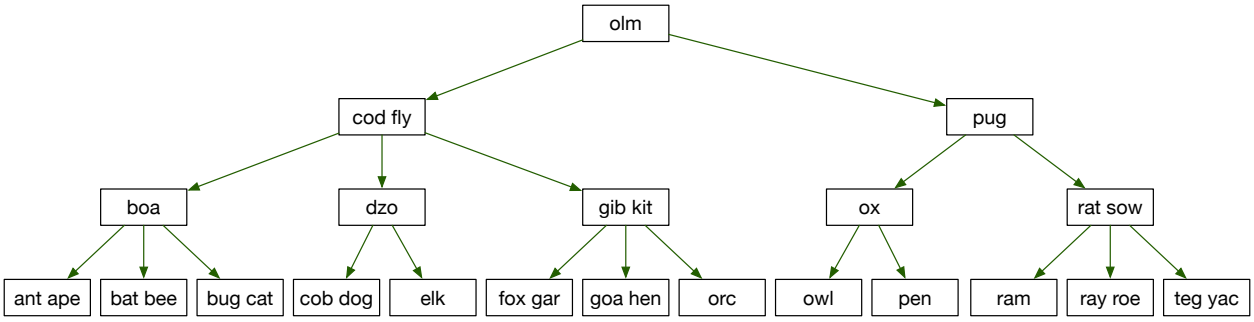
final result:



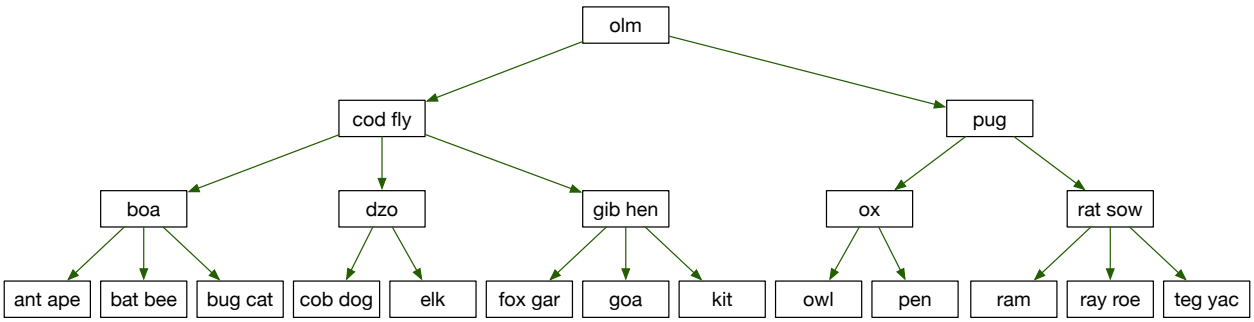
Problem 4:



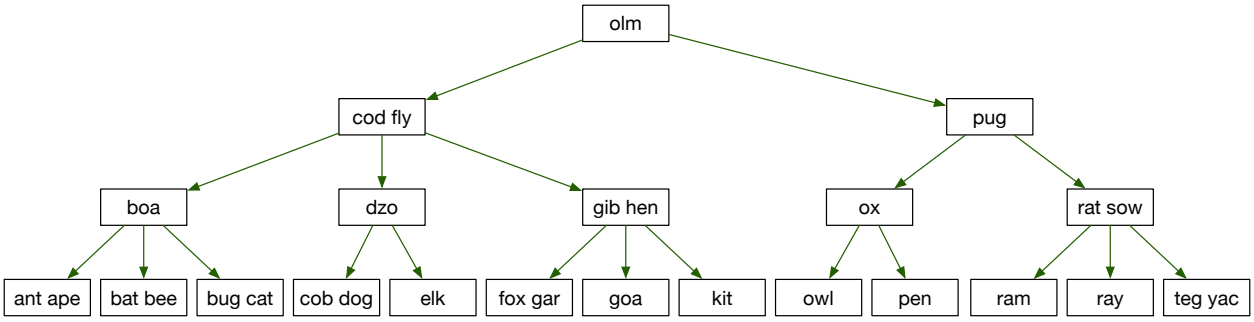
To delete orc, we exchange it with its predecessor:



After deleting orc, we have an underflow. The only way to cure the underflow is by a right rotate, where hen goes up and kit goes down:



We can delete “roe” directly, as it is located in a leaf node. No restructuring is needed.



To delete rat, we exchange it with its predecessor and then delete. This gives an underflow.

To remedy the underflow, we can try to rotate, but the only sibling has only one key in it. We therefore need to merge the empty leaf and the leaf with ‘ray’. This is done by moving ‘ram’ downwards into the merged leaf.