Midterm Algorithms Solutions

Problem 1:

The algorithm makes four recursive calls on an array of size $\frac{2n}{3}$. This gives a recurrence of

$$T(n) = 4(\frac{n}{3/2}) + \text{const.}$$

The critical value is $c = \log_{\frac{3}{2}}(4) \approx 3.419$, which is much larger than 2. Thus, we compare a constant with $n^{c-\epsilon}$ and are in Case 1 of the MT. Therefore, $T(n) = \Theta(n^c)$.

Problem 2:

(a) There are $\frac{n(n-1)}{2}$ pairs of indices in an array of length n, so that comparing all elements of the array takes time $O(n^2)$.

(b) We walk through the array. We insert the key-value pair (word — index of the word) into the LH file. If the word is already there, we calculate the distance between the two indices and compare it to the smallest distance seen. Afterwards, we update the index of the word to the new index. Since inserts and lookups into an LH file take probabilistically constant time, the algorithm runs in time $O(n)$.

Problem 3:

We notice that the cap is not uniquely determined, but can be chosen to be equal to an actual salary. We start out by ordering the array of salaries from smallest to largest. This takes $\Theta(n \log(n))$ time. Let this array be $[s_1, s_2, s_3, \ldots, s_n]$. If the cap is smaller than s_1, the payroll is $nC < ns_1$. If the cap is s_1, then the payroll is $P_1 = ns_1$. If the cap is s_2, then the payroll is

$$P_2 = s_1 + (n-1)s_2,$$

which is equal or larger than before. If the cap is s_3, then the payroll is

$$P_3 = s_1 + s_2 + (n-2)s_3.$$

In general, if the cap is s_i, then the pay-roll is

$$P_i = s_1 + s_2 + \ldots + s_{i-1} + (n-i+1)s_i.$$

These potential payrolls increase with i: $P_1 \leq P_2 \leq P_3 \leq \ldots \leq P_n$. Furthermore,

$$P_i = P_{i-1} + (s_i - s_{i-1})$$

so that we can calculate all potential payrolls in linear time. Thus, given the payroll, we calculate the largest index P_i that is smaller or equal to the target payroll. This gives us the cap as s_i.

Problem 4:

```
def minmax(array):
```

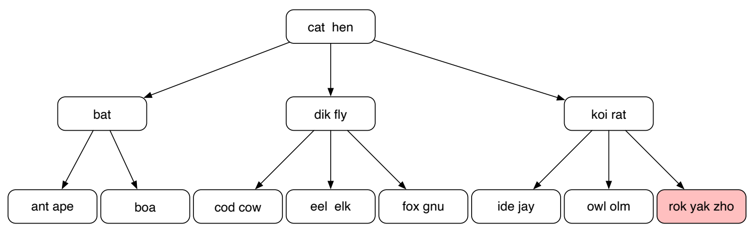
```
n = len(array)
if len(array) == 1:
    return array[0], array[0]
min1, max1 = minmax(array[:n//2])
min2, max2 = minmax(array[n//2:])
return min(min1, min2), max(max1, max2)
```

The runtime is given by the recurrence $T(n) = 2T(n/2)$+const. The critical value is $\log_2(2) = 1$. As const $= O(n^{1-1})$, Case 1 of the MT applies and $T(n) = \Theta(n^1)$.

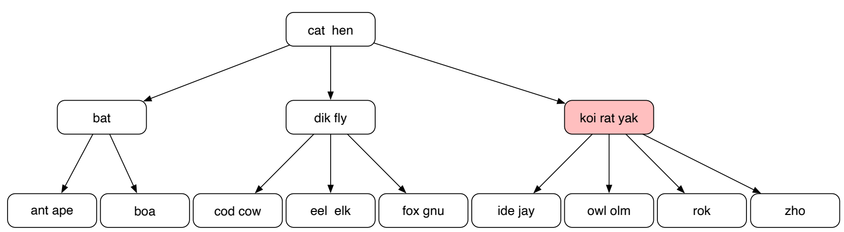
Problem 5:

Since $9 = 2^3 + 1$, the split pointer is 1 and the level is 3. The buckets are 20 —> Bucket 4, 25 —> Bucket 1, 30 —> Bucket 6, 35 —> Bucket 3, and 40 —> Bucket 8.

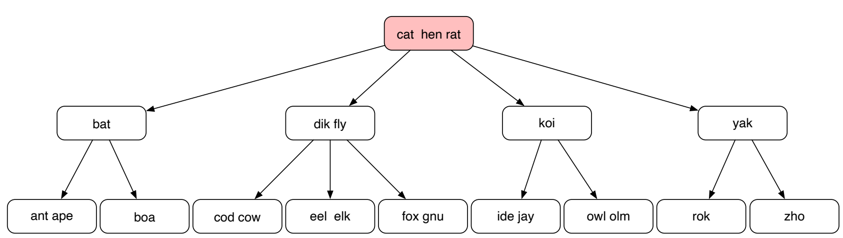
Problem 6:



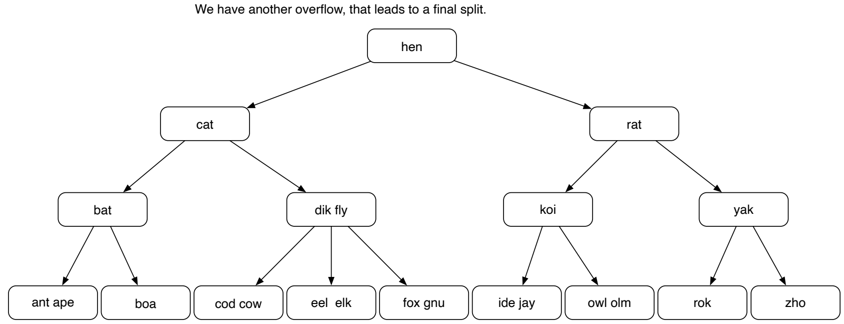
Insert into a leaf, which leads to an overflow. The overflow can only be remedied by a split

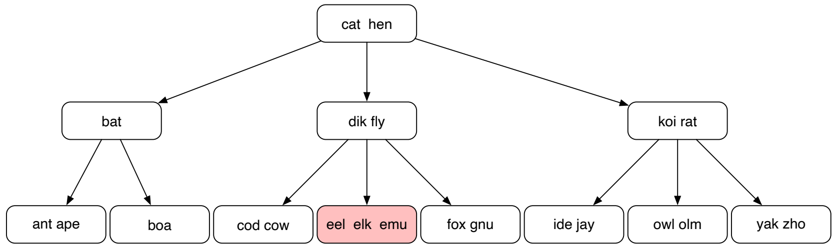


The split also leads to an overflow, that cannot be remedied by a rotation, so we have another split.

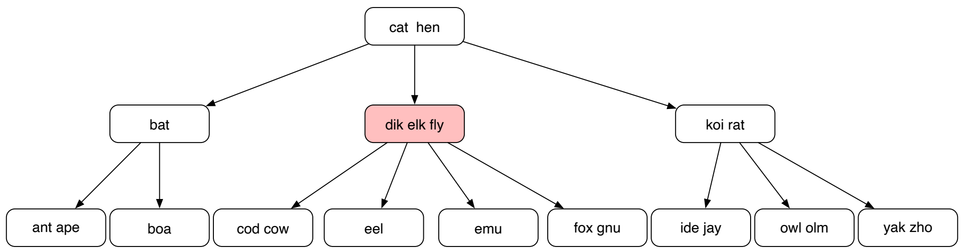


We have another overflow, that leads to a final split.





After insertion in the leaf, we have an overflow. Rotates are not possible, so we have a split.



We have another overflow, but this time, we can rotate: dik goes up, cat goes down