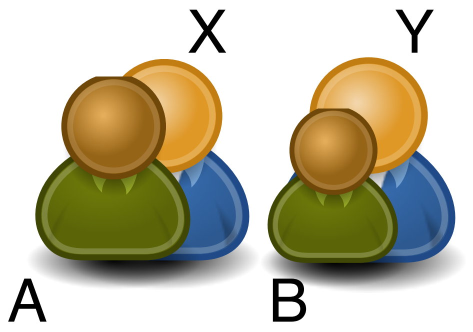
Sample Midterm Solutions

Problem 1:

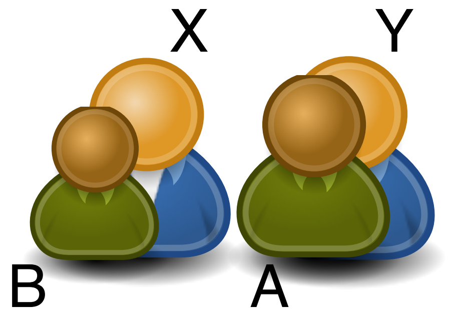
(a) There are $n!$ ways to order the first row and $n!$ ways to order the back row. There are two ways to place a team in front and the other one in the back. This gives a total of $2 \cdot n! \cdot n!$ arrangements. (The factor of 2 is arguable, as the problem can be read to say that one team has to 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) To extract the maximum and minimum of four elements, we place the four elements into two pairs and compare the pairs. The two larger of the pairs are compared to find the maximum and the two smaller of the pairs are compared to find the minimum. This gives four comparisons.

(b) If $n \equiv 0 \pmod 4$, then we have $\frac{n}{4} \cdot 4$ comparisons within the groups and twice $\frac{n}{4} - 1$ comparisons for a total of $\frac{6n}{4} - 2 = \frac{3n}{2} - 1$.
If $n \equiv 1 \pmod 4$, then we still have $\frac{n-1}{4} \cdot 4$ comparisons within the groups, but now twice $\frac{n}{4}$ comparisons between the group maxima and minima. This gives a total of $\frac{3}{2}(n-1)$ comparisons.

If $n \equiv 2 \pmod 4$, then the last group has two elements and we need one comparison to determine maxima and minima. Thus, within the groups we have $\frac{n-2}{4} \cdot 4 + 1$ comparisons and twice $\frac{n-2}{4}$ comparisons among the maxima and minima, respectively. This gives a total of $\frac{3n}{2} - 2$.

If $n \equiv 3 \pmod 4$, then the last group has three elements. To extract both maximum and minimum of this group, we need three comparisons. Within the groups, we have $\frac{n-3}{4} \cdot 4$ comparisons. As we have $\frac{n-3}{4} + 1$ groups, we have an additional $2 \cdot \frac{n-3}{4}$ comparisons among the group maxima and minima. This gives a total of $\frac{3n}{2} - \frac{3}{2}$ comparisons.

Problem 3:

The subsets of the states of the NFA are the possible states of the DFA.

State	On input 0	On Input 1
{A}	{B}	{A,B,C}

State	On input 0	On Input 1
{B}	{A}	{A,C}
{A,C}	{B}	{A,C}
{A,B,C}	{A,B}	{A,B,C}
{A,B}	{A,B}	{A,B,C}

Problem 4:

Let $T(n)$ be the runtime of the algorithm with n being the difference hi-lo. Then $T(0) =$ const. Otherwise, $T(n) = nT(n-1)$ as there are $n-1$ recursive calls on an array with n elements.

Problem 5:

(a) The MT applies with $a = 27$ and $b = 3$. We compare $f(n) = n^2$ with n^c where $c = \log_3(27) = 3$. We set $\epsilon = 1/2$ and compare with $n^{2.5}$. As

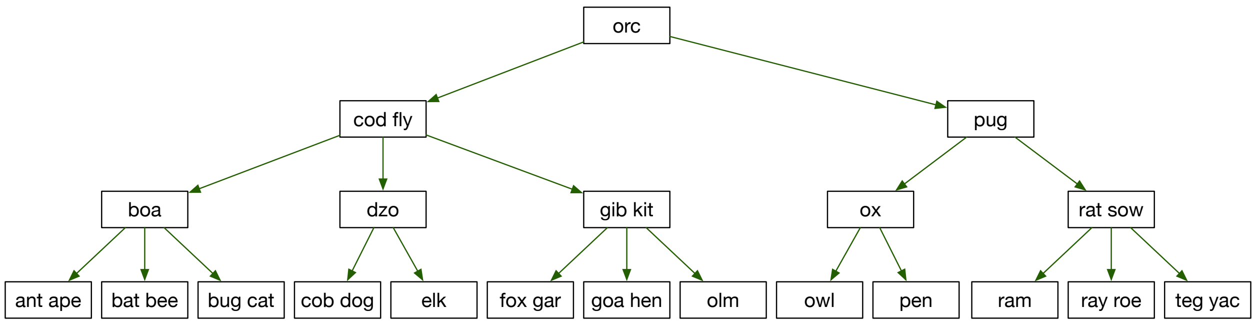
$$\lim_{n\to\infty} \frac{n^2 \log(n)}{n^{2.5}} = \lim_{n\to\infty} \frac{\log(n)}{\sqrt{n}} =^{LH} \lim_{n\to\infty} \frac{\sqrt{n}}{\frac{1}{2}n} = \lim_{n\to\infty} \frac{1}{2\sqrt{n}} = 0,$$

$f(n) = O(n^{3-\epsilon})$ and Case 1 of the MT applies. Thus, $T(n) = \Theta(n^3)$

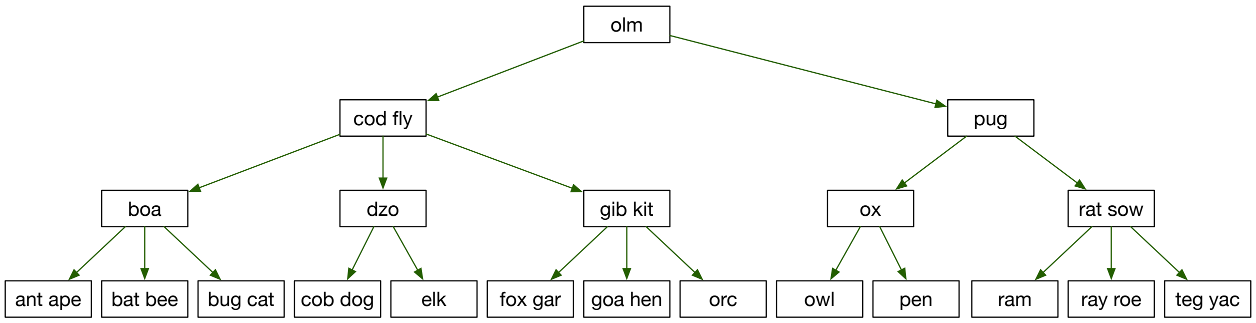
(b) The recurrence is not of the form of the MT.

(c) The MT applies with $a = 2$ and $b = 4$. The critical exponent is $c = \log_4(2) = \frac{1}{2}$. Thus, Case 2 of the MT applies and $T(n) = \Theta\left(\sqrt{n}\log(n)\right)$.

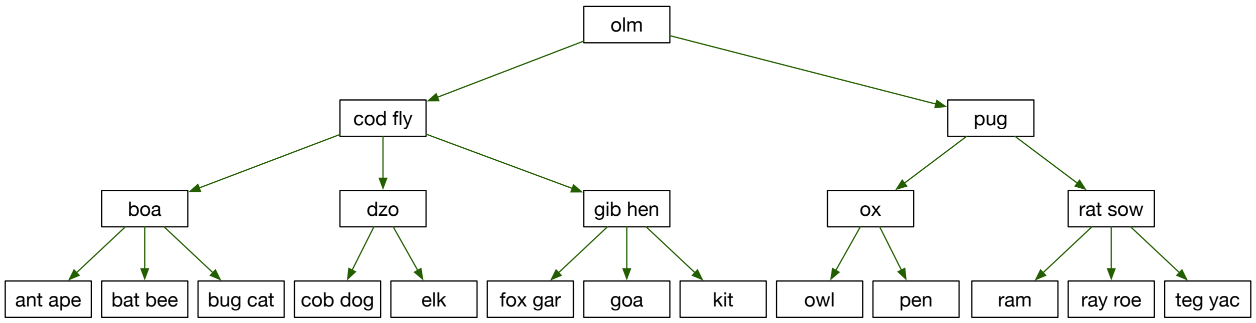
Problem 6:



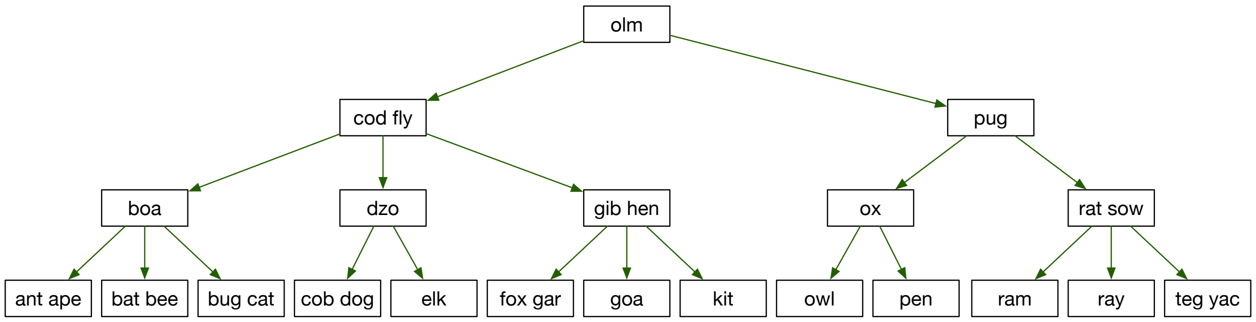
To delete orc, we exchange it with its predecessor:



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:

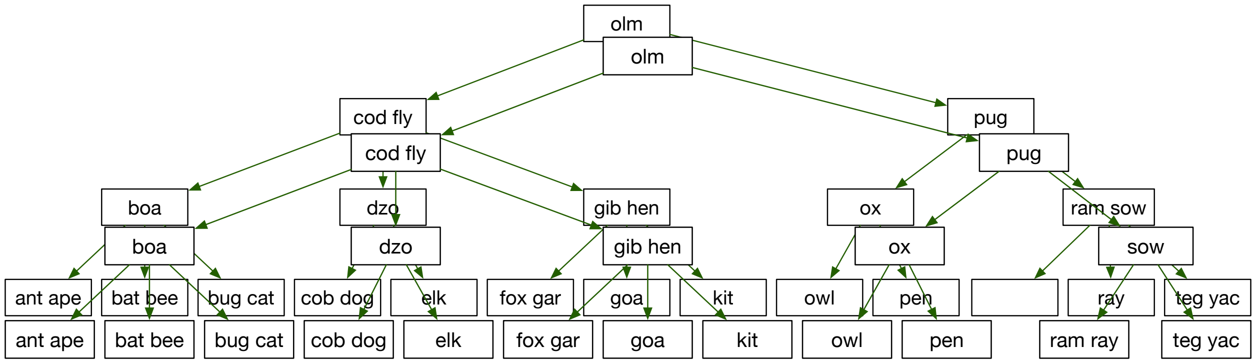


We can delete “roe” directly, as it is located in a leaf node. No restructuring is needed.



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.



Problem 7:

$10 = 2^3 + 2$, so level is 3 and split pointer is 2. 5 —> Bucket 5, 6 —> Bucket 6, 7 —>Bucket 7, 8—> Bucket 8, 9 —> Bucket 9, 10 —> Bucket 2.

Extra Problems

Problem 1:

(1) The number of comparisons needed to find the maximum of n values is $n-1$.
(2) Divide the array into groups of eight (linear time work). Then determine the maxima of each group (time is proportional to $n/8$) and put those maxima into an array. The apply the algorithm recursively to the new array of maxima.
(3) This recurrence relation is given by $T(n) = T(n/8) + Cn$ with some constant C.
(4) The critical value is $c = \log_8(1) = 0$. We need to compare Cn with $n^0 = 1$. Obviously, for any ϵ, $0 < \epsilon < 1$, $Cn = \Omega(n^\epsilon)$. For regularity, we just observe that

$$af(\frac{n}{b}) = C\frac{n}{8} \leq \frac{1}{2}Cn = \frac{1}{2}f(n).$$

Thus, Case 3 of the MT applies and we have $T(n) = \Theta(n)$. This is no surprise. More interestingly, if we count the number of steps using the sorting network we get

$$Cn + C\frac{n}{8} + C\frac{n}{8^2} + \ldots \approx Cn \cdot \sum_{i=0}^{\infty}(\frac{1}{8})^i = \frac{8}{7}Cn.$$

Problem 2:

The recursive call is on arrays of size 1/3 of the original array and there are five such calls. This gives the recurrence

$$T(n) = 5T(n/3) + Cn.$$

The addend on the right reflects the breaking up of the array into sub-arrays and the final concatenation. The MT looks at the critical value $c = \log_3(5) \approx 1.465$. Obviously, $f(n) = O(n^d)$ for $d > 1$, so that we are in Case 1 of the MT. Thus, $T(n) = \Theta(n^c)$.

By the way, the algorithm does not sort the array correctly. If the maximum is located in the first sixth of the array, it never appears in the result.