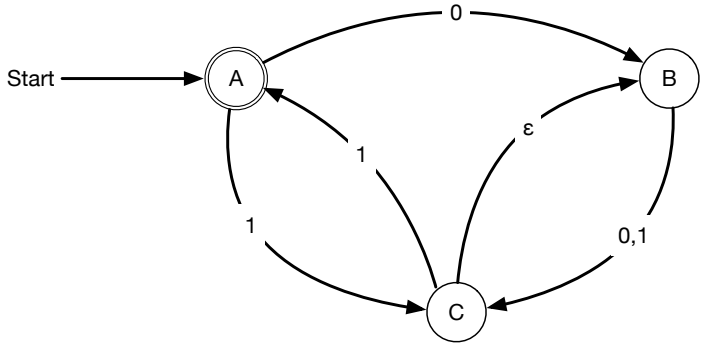
Solutions Midterm Algorithms

Problem 1: Convert the following NFA with epsilon moves to a DFA. For this, give the transition table, specify the starting state and the accepting states.



State	0	1
{A}	{B}	{B,C}
{B}	{B,C}	{B,C}
{B,C}	{B,C}	{A,B,C}
{A,B,C}	{B,C}	{A,B,C}

Starting state is {A}, accepting states are {A} and {A,B,C}.

Problem 2: Use the Master Theorem (if possible) on the following recursions: (If the MT cannot be applied, just say so.)

a. $T(n) = \sqrt{2}T(n/2) + \log(n)$

b. $T(n) = 3T(n/4) + n\log(n)$

(a) Compare $\log(n)$ with $n^{\log_2(\sqrt{2})} = n^{1/2}$. Since

$$\lim_{n\to\infty} \frac{\sqrt{n}}{\log n} = \lim_{n\to\infty} \frac{\frac{1}{2}n^{-\frac{1}{2}}}{n^{-1}} = \lim_{n\to\infty} \frac{1}{2}n^{\frac{1}{2}} = \infty,$$

we have $\log n = o(\sqrt{n})$. Select $0 < \epsilon < 1/2$ and set $c = \frac{1}{2} - \epsilon > 0$. Then

$$\lim_{n\to\infty} \frac{n^c}{\log n} = \lim_{n\to\infty} \frac{cn^{c-1}}{n^{-1}} = \lim_{n\to\infty} cn^c = \infty$$

and still $\log(n) = o(n^c)$. We are in Case 1 of the MT. This implies $T(n) = \Theta(\sqrt{n})$.

(b) We compare $n\log(n)$ with $n^{\log_4(3)}$. Notice that $\log_4(3) < 1$ and pick ϵ positive but small enough such that $c = \log_4(3) + \epsilon < 1$. Then

$$\lim_{n\to\infty} \frac{n\log(n)}{n^c} = \lim_{n\to\infty} \frac{n\frac{1}{n} + \log(n)}{cn^{c-1}} = \lim_{n\to\infty} \frac{1 + \log(n)}{cn^{c-1}} = \lim_{n\to\infty} (\frac{1}{c}n^{c-1} + \frac{\log(n)}{cn^{c-1}})$$

$$= 0 + \frac{1}{c}\lim_{n\to\infty} \frac{\log(n)}{n^{c-1}} = \frac{1}{c}\lim_{n\to\infty} \frac{n^{-1}}{(c-1)n^{c-2}} = \frac{1}{c(c-1)}\lim_{n\to\infty} n^{1-c} = \infty.$$

Therefore, $n^c = \Omega(n\log(n))$. Because $af(n/b) = 3\frac{n}{4}\log(\frac{n}{4}) \leq 3\frac{n}{4}\log(n) \leq \frac{3}{4}f(n)$, the regularity condition is fulfilled. Therefore, we are in Case 3 of the MT and have $T(n) = \Theta(n\log(n))$.

Problem 3: The maximum subarray problem is to find a contiguous subarray in an array of numbers whose sum is the largest. For example, for $[-2,-5,6,-2,-3,1,5,-6]$, we find the maximum subarray to be the subarray $[6,-2,-3,1,5]$. Just picking the positive numbers would not count, because the subarray has to be contiguous. In Python, the subarray has to be a simple slice.

First approach: We calculate the sum of each subarray.

```
def sub(array):
   best = - MAXINT
   besti, bestj = 0,0
   for i in range(len(array)):
      for j in range(i+1, len(array)):
          if(sum(array[i:j])> best:
              best = sum(array([i:j])
              besti, bestj = i, j
```

What is the runtime of this algorithm for an array of length n?

Let the length of the array be n. The inner for loop has first $n-1$ iterations, then $n-2$, and so on. Therefore, the outer loop has $(n-1)+(n-2)+\ldots+2+1=\frac{n(n-1)}{2}$ (Little Gauss) iterations and the algorithm is quadratic in n.

Second approach (Divide and conquer):

1. Divide the array A in two halves
2. Return the maximum of the following three numbers:
   - **(a)** The maximum subarray of the first half
   - **(b)** The maximum subarray of the second half
   - **(c)** The maximum subarray that starts in the first half and ends in the second half.
     - **i.** Start at the midpoint, given by index m. Then calculate for all indices r larger than m the sum of the array $A[m : r + 1]$ and find the maximum.
     - **ii.** Start at the midpoint, given by index m. Then calculate for all indices l smaller than m the sum of the array $A[l : m]$ and find the maximum.
     - **iii.** Combine the two maximizing subarrays for the maximum subarray that starts in the first half and ends in the second half.

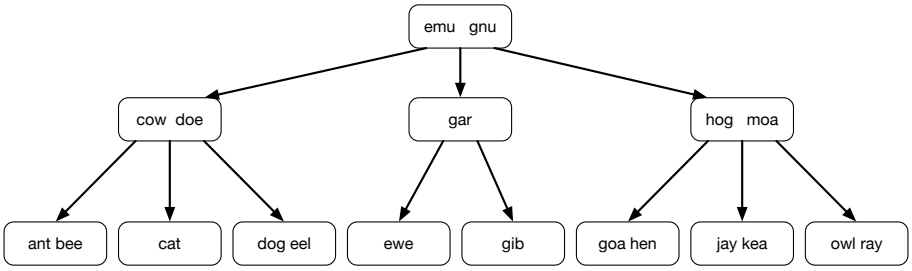
Write a recurrence relation for the runtime of this algorithm depending on the length of the array.

Part iii has one sum calculation for each index above and for each index below the midpoint, which is $\Theta(n)$. Parts i and ii are just recursive calls on arrays slightly smaller than the length divided by 2. We get

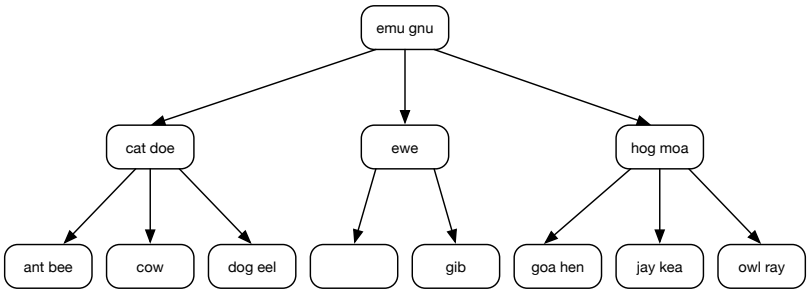
$$T(n) = \begin{cases} 2T(\lfloor n/2 \rfloor) + \Theta(n) & \text{if } n \text{ odd} \\ T(n/2) + T(n/2 - 1) + \Theta(n) & \text{if } n \text{ even} \end{cases}.$$

We can approximate by $T(n) = 2T(n/2) + \Theta(n)$ (and solve it with case two of the MT).

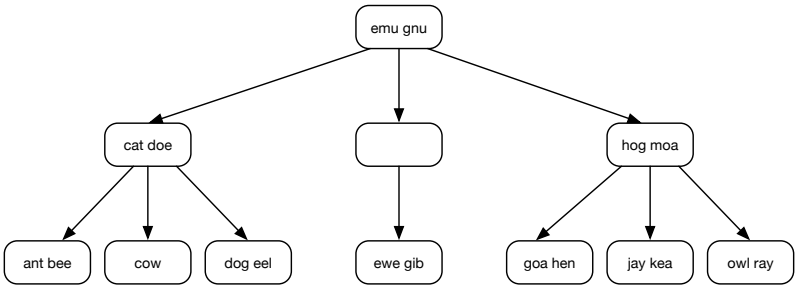
Problem 4: Delete 'gar' from the following B-tree. (Prefer left rotates over right rotates over splits over merges). Show each step of your restructuring.



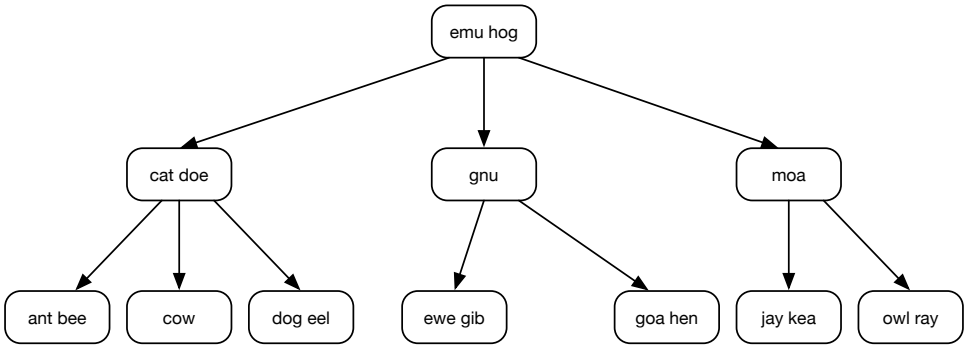
First, we exchange 'gar' with its predecessor 'ewe' and delete it:



We need to merge the two leaf nodes



Then we need to do a left rotate in order to get the final solution



Extra Credit Problem:

It takes a factor of 2 cycles to reach a nano-second, 1000 to reach a micro-second, another 1000 to reach milli-second, and then 10 to reach the average random access time for a block on the hard drive for a total of $2 \cdot 10^7$ cycles.