COMP285: Analysis of Algorithms
North Carolina A&T State University
Dr. Allison Sullivan

Homework #3
Due: September 14, 2018 (*in class quiz*)

Homework #3

You should try to solve these problems by yourself. I recommend that you start early and get help in office hours if needed. If you find it helpful to discuss problems with other students, go for it. **You do not need to turn in these problems. The goal is to be ready for the in class quiz that will cover the same or similar problems.**

Problem 1: Recurrences, one more crazy time! (And not the last time...)

In this problem, you are to analyze the asymptotic time complexity $T(n)$ for the following divide and conquer algorithm. Assume that `bar(k, m)` performs a constant time operation on whatever input it receives (that is, don't worry about what exactly it does). Show your work. Write the complete recurrence, and solve it using either the master method or a recursion tree.

```
int[] foo (int [] A){
   n = A.size()
   int B [] = new int[n]
   if ( n <= 1 ) { return }
   int even = new int[n/2]
   int odd = new int[n/2]
   int j = 0
   for(int i = 0; i < n; i = i + 2){
      even[j] = A[i]
      j++
   }
   int j = 0
   for(int i = 1; i < n; i = i + 2){
      odd[j] = A[i]
      j++
   }
   foo(even)
   foo(odd)
   for(int k = 0; k < n/2; k++){
      for(int m = 0; m <= k; m++){
         int t = bar(k, m) * odd[k]
         B[k] = even[k] + t
         B[k*n/2] = even[k] - t
      }
   }
   return B
}
```

Problem 2: LexicoSort

Let $S =$ '$s_1, s_2, \dots, s_a$' and $T =$ '$t_1, t_2, \dots, t_b$' be strings of length a and b, respectively (we call s_i the i^{th} letter of S). We say that S is lexicographically less than T, denoting $S <_{lex} T$, if either

- $a < b$ and $s_i = t_i$ for all $i = 1, 2, \dots, a$, or
- There exists an index $i \leq min\{a, b\}$ such that $s_j = t_j$ for all $j = 1, 2, \dots i - 1$ and $s_i < t_i$.

A lexicographic sorting algorithm aims to sort a given set of n strings into lexicographically ascending order (in case of ties due to identical strings, then in non-descending order). For each of the following, describe your algorithm clearly, and analyze its running time.

(a) Give an O(n) lexicographic sorting algorithm for n strings where each string consists of exactly one letter from the set $a - z$ (that is, each string is of length 1). (Hint: think about the CountingSort)

(b) Give an $O(mn)$ lexicographic sorting algorithm for n strings where each string contains letters from the set $a - z$ and is of length at most m. (Hint: you'll need to pad words of $length < m$.)

Problem 3: Goldilocks' Nightmare

Goldilocks is trapped in her worst nightmare. She is in a room with n beds in a line, ordered from shortest to tallest. And she needs to find the perfect bed. Her only saving grace is that she knows exactly how long the perfect bed is: it is the same length as her measuring stick. She wants to find the bed which is the same length as the stick, or else report that there is no such bed. The only operation you are allowed to do is `compareToStick(b)`, where `b` is a bed, which returns taller if `b` is taller than the stick, shorter if `b` is shorter than the stick, and the same if `b` is the same height as the stick. Additionally, since she is stuck in a nightmare, she does not have a piece of paper, thus she can only store up to seven integers in $0, \dots, n - 1$ at a time.

She could solve this iteratively, but its a nightmare, and she wants to wake up as fast as possible. Give an algorithm which finds a bed of the same height as the stick, or else returns "No such bed" that runs in $O(\log n)$ time.

Problem 4: Dean's Dilemma

Detailed-Oriented Dean sometimes does consulting work on the side. (This work is mostly pest control related, i.e. he'll eat your bugs). There are n households who are interested in Dean's work. Dean can work for at most one household during a given day. Each household has a range of times when they are interested in Dean's work, and an amount they are willing to pay: between a_i and b_i (including a_i and not including b_i), household i is willing to pay Dean p_i potato chips[1]. Here, a_i, b_i, p_i are all positive integers and $a_i \leq b_i$. Each day, Dean chooses to work for the highest bidder. If there is no household interested in Dean's work on a day, Dean gets zero potato chips that day.

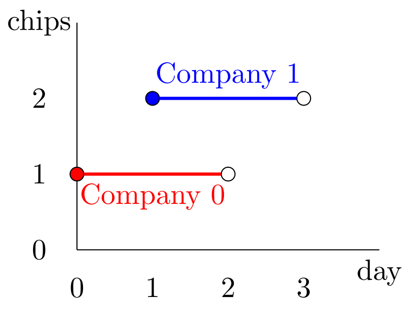
Dean gets the bids (a_i, b_i, p_i) as inputs, and wants to make a plot of how many treats he will receive each day. To understand the format he wants the output in, see the example below.

[1]Dean actually loves potato chips, so much so there is never peace if you open a bag near him.

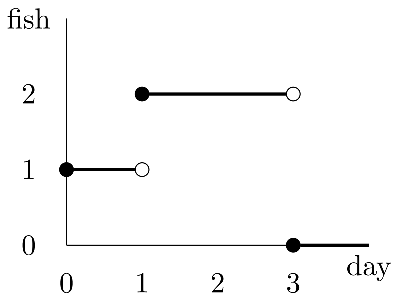
Example

Suppose that $n = 2$. As input, Dean would get the following data, which can be visualized as the graph below.

Company i	a_i	b_i	p_i
Company 0	0	2	1
Company 1	1	3	2



In this example, Dean would work for Company 0 at day 0, and receive one chip. He'd work for Company 1 on days 1, 2, and receive two chips on each of those days. On days 3 and onwards, no company was interested in Dean's work, so he works for no company and receives zero chips. So his output plot would look like this:



To return this plot, Dean will return a sequence (t_0, p_0), (t_1, p_1), ..., with $t_i \leq t_{i+1}$, which we interpret as meaning starting on day t_i and ending on day t_{i+1} - 1, Dean makes p_i chips. In the example above, the return value would be $(t_0 = 0,\ p_0 = 1)$, $(t_1 = 1,\ p_1 = 2)$, $(t_2 = 3,\ p_2 = 0)$.

Takeaways from this example:

- The last p-value will always be 0.
- In the example above, it would also be correct to return $(t_0 = 0,\ p_0 = 1)$, $(t_1 = 1,\ p_1 = 2)$, $(t_2 = 2,\ p_2 = 2)$, $(t_3 = 3,\ p_3 = 0)$, that is, breaking an interval into two smaller intervals is still correct.

In this problem you'll design an algorithm for Dean. Your algorithm should take as input a list of n bids $(a_i,\ b_i,\ f_i)$, one for each company $i \in \{0, \ldots, n-1\}$, and return a list `chipPlot` of $(t_i,\ p_i)$ pairs as described in the example above. Design a divide-and-conquer algorithm that takes time $O(n \log(n))$.