Due: Wednesday 9/5 before 9:15

1. (2.6) You're given an integer array $A = (A[1], A[2], \ldots, A[n])$. Your goal is to output an n by n array $B = [B[i, j]]$, where $B[i, j] = A[i] + A[i + 1] + \cdots + A[j]$ for $i < j$ (for $i \geqslant j$ the value of $B[i, j]$ is not specified, so it doesn't matter what is output for these values).

   Here's an algorithm to solve this problem:

   ```
   for i = 1, 2, ..., n:
      for j = i+1, i+2, ..., n:
         add up array entries A[i] through A[j]
         store the result in B[i,j]
   ```

   (a) Pick a function f and show that the running time of the algorithm is $O(f(n))$ for an input of size n.

   (b) For the same function f show that the running time of the algorithm is also $\Omega(f(n))$. (Here you may have to go back to the previous part and correct your answer if this is not true for the function you had picked.

   (c) The algorithm you've just analyzed is, in fact, not very efficient. Give a different algorithm to solve the same problem, with an asymptotically better running time: if your algorithm has running time $O(g(n))$, then it should be the case that $g \in o(f)$.

2. (2.8) You're doing stress-testing of glass jars to determine the maximum safe height from which they can be dropped without breaking. The experiment (for a particular fixed type of jar) works as follows. You have a ladder with n rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the *highest safe rung*.

   How to find the highest safe rung? A natural approach is *binary search*: try rung $n/2$, then if the jar has broken, rung $n/4$, if not then $3n/4$ etc. Such a procedure will end after $O(\log n)$ steps, but you may break as many as $\log n$ jars in the process.

   If the jars were expensive, you might start from the bottom rung and keep increasing the height by one until you finally break the jar. This way you would break only one jar, but may have to try the experiment a linear number of times.

   Is there a trade-off? If you allow a few more jars to be broken, how many tries are required?

   (a) Suppose you are given a budget of $k = 2$ jars. Explain how to find the highest safe rung while dropping a jar at most $f(n)$ times, for some functin $f \in o(n)$.

   (b) Now suppose you have a budget of k jars, for some given $k > 2$. Describe a strategy for finding the highest safe rung using at most k jars. If your strategy requires $f_k(n)$ jar drops, then the functions $f_1, f_2, \ldots$ should satisfy $f_k \in o(f_{k+1}$ for each k.

3. (3.4) Your friends go on a field trip and catch n butterflies. They believe each of the butterflies belongs to one of only two different species, A and B. It is very difficult to label any one specimen directly as belonging to A or B, but a comparison between two butterflies makes it easier to say if they belong to the same species or not. For example, by studying butterfly i side-by-side with butterfly j, they may be able to label the pair (i, j) as "same" or "different". Occasionally, they cannot decide on the label and leave the pair labeled as "ambiguous".

   Now we have a collection of n specimens and m "judgments" for the pairs that were not declared ambiguous. Is this data set consistent with the assumption that the butterflies all come from only two distinct species? Let's say the data set is *consistent* if we can label each butterfly as belonging either to A or to B in such a way that an unambiguous pair (i, j) is labeled "same" if and only if i and j both belong to the same species.

   Give an algorithm with running time $O(n+m)$ that determines whether the m judgments are consistent.

4. (3.6) Let $G = (V, E)$ be a connected graph and $u \in V$ a vertex. Let T be the depth-first search tree of G rooted at u (because G is connected, T spans all its vertices). Suppose that a breadth-first search tree of G is computed from u and we get the same tree T. Prove that $G = T$.

5. **[Implementation question]** Implement the flood fill algorithm using breadth-first search. More precisely, you should write a function `floodfill` that takes as argument a list of strings. The interpretation of the list is the following: imagine printing the strings as lines of text on a sheet of paper, using a monospaced font (one where every letter takes up the same amount of horizontal space). The strings are guaranteed to contain only the characters "X" and " " (space). It is also guaranteed that the characters "X" make up a cycle in the square grid graph whose vertices are the possible locations of the characters and whose edges join the neighboring locations (left-right and up-down). Your algorithm should find a point interior to this cycle and then fill the interior with "X"s, using a variant of the breadth-first search algorithm.

   For example, the list composed of the three strings "`XXXXX`", "`X   X`", "`XXXXX`" defines a cycle that contains 12 vertices, and encloses three additional points of the grid:

```
XXXXX
X...X
XXXXX
```

   (In this illustration, the dots stand for blank space in the second string, or equivalently for the vertices of the grid interior to the cycle.)

   Your function should return a list of strings that in this same way describes the initially given region filled by "X"s.

   Hint: first make sure you can accurately locate a point interior to the cycle.