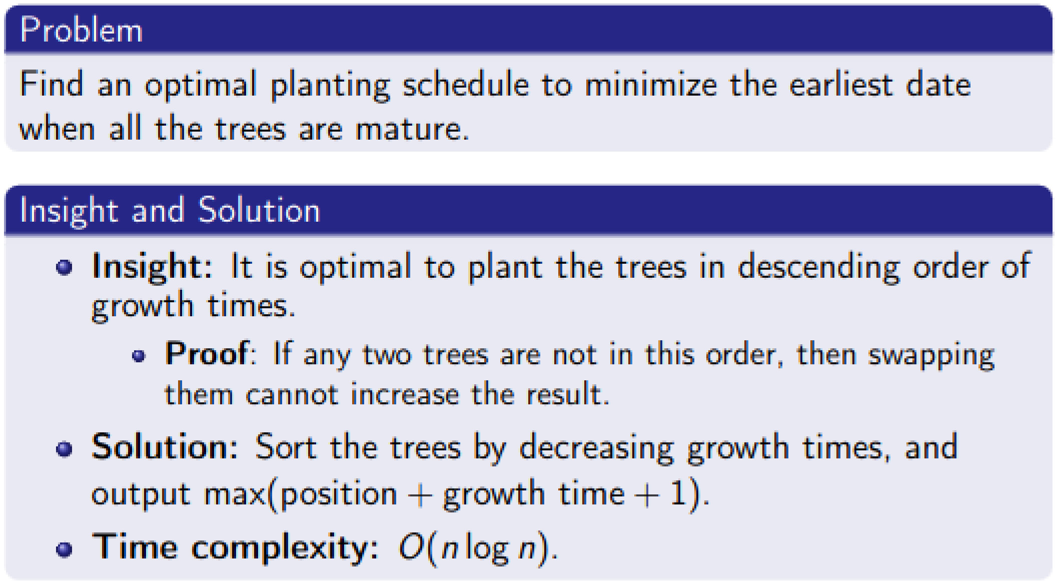
# Spring 24 Div I Week 2 - Solutions

(taken from editorials of original problems)

## Problem A

(Source: Nordic Collegiate Programming Contest 2013 Problem A)



**Problem**

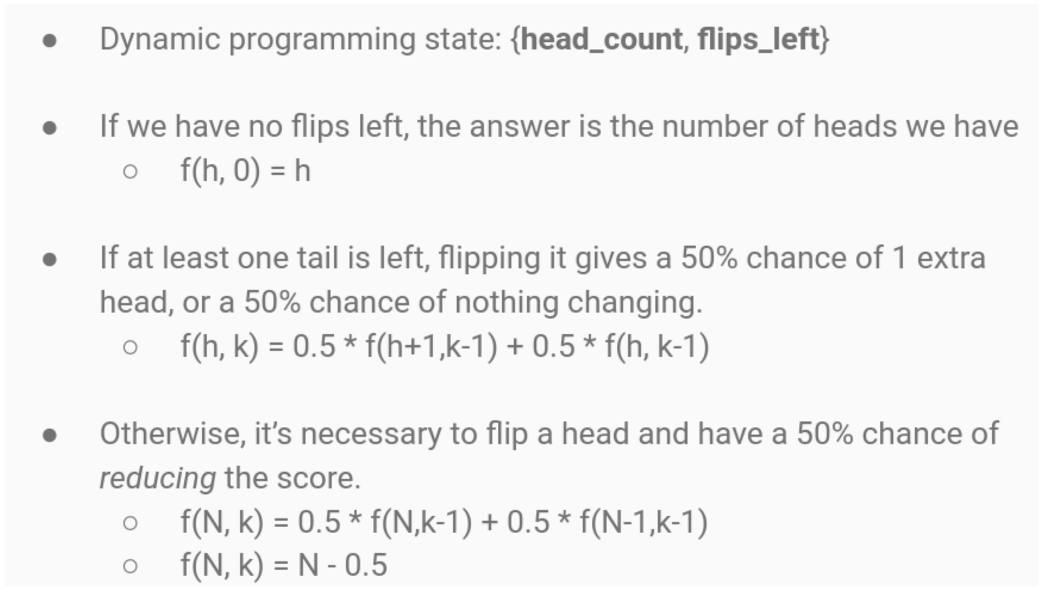
Find an optimal planting schedule to minimize the earliest date when all the trees are mature.

**Insight and Solution**

- **Insight:** It is optimal to plant the trees in descending order of growth times.
    - **Proof**: If any two trees are not in this order, then swapping them cannot increase the result.
- **Solution:** Sort the trees by decreasing growth times, and output max(position + growth time + 1).
- **Time complexity:** $O(n \log n)$.

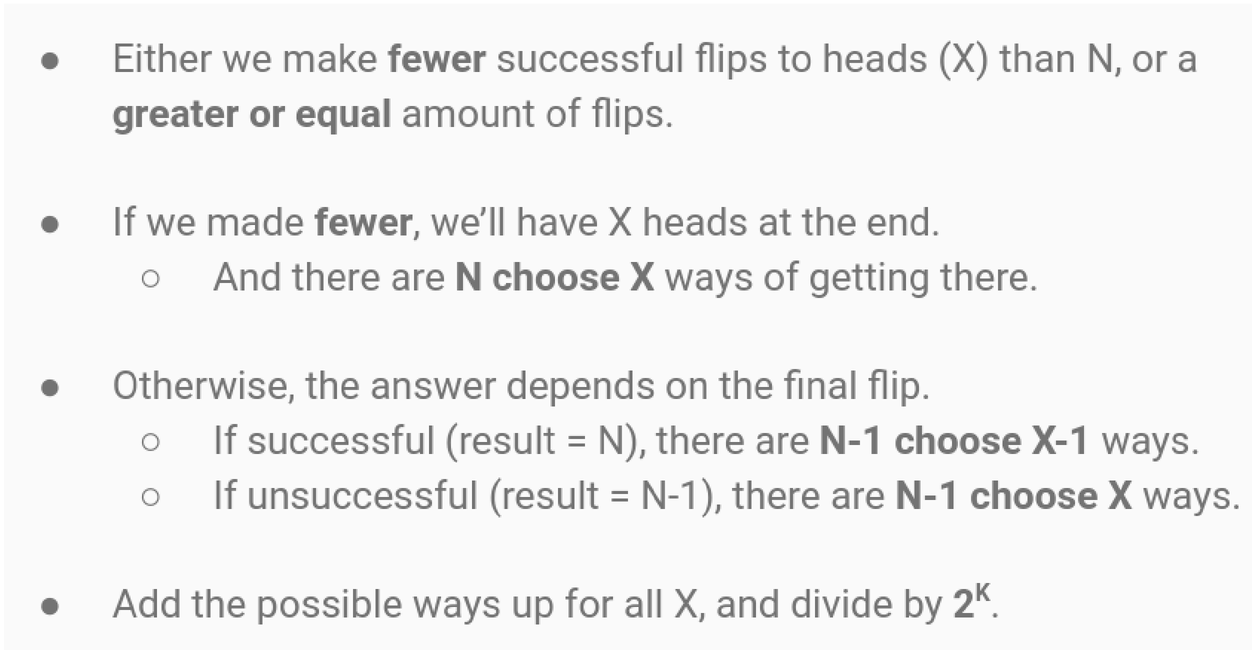
## Problem B

(Source: UK & Ireland Programming Contest 2017 Problem F)



- Dynamic programming state: {**head_count**, **flips_left**}
- If we have no flips left, the answer is the number of heads we have
    - f(h, 0) = h
- If at least one tail is left, flipping it gives a 50% chance of 1 extra head, or a 50% chance of nothing changing.
    - f(h, k) = 0.5 * f(h+1,k-1) + 0.5 * f(h, k-1)
- Otherwise, it's necessary to flip a head and have a 50% chance of *reducing* the score.
    - f(N, k) = 0.5 * f(N,k-1) + 0.5 * f(N-1,k-1)
    - f(N, k) = N - 0.5

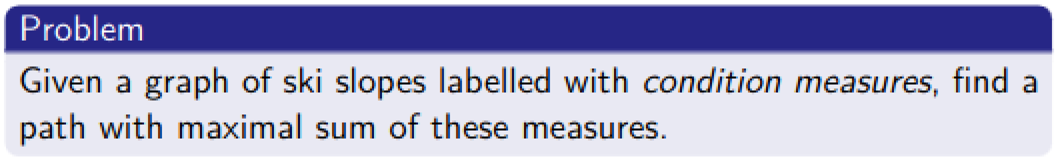
Alternative solution:



- Either we make **fewer** successful flips to heads (X) than N, or a **greater or equal** amount of flips.
- If we made **fewer**, we'll have X heads at the end.
  - And there are **N choose X** ways of getting there.
- Otherwise, the answer depends on the final flip.
  - If successful (result = N), there are **N-1 choose X-1** ways.
  - If unsuccessful (result = N-1), there are **N-1 choose X** ways.
- Add the possible ways up for all X, and divide by **2$^{K}$**.

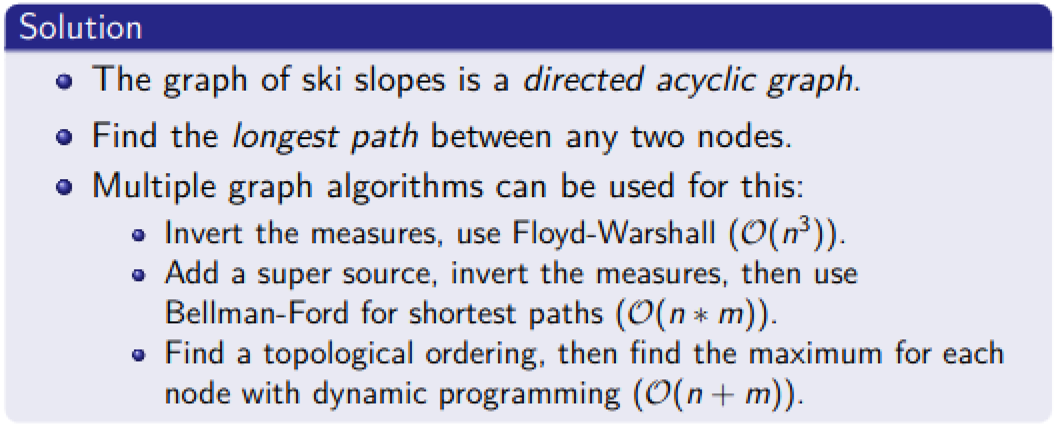
## Problem C

(Source: German Collegiate Programming Contest 2018 Problem C)



Problem

Given a graph of ski slopes labelled with *condition measures*, find a path with maximal sum of these measures.



Solution

- The graph of ski slopes is a *directed acyclic graph*.
- Find the *longest path* between any two nodes.
- Multiple graph algorithms can be used for this:
  - Invert the measures, use Floyd-Warshall ($\mathcal{O}(n^3)$).
  - Add a super source, invert the measures, then use Bellman-Ford for shortest paths ($\mathcal{O}(n * m)$).
  - Find a topological ordering, then find the maximum for each node with dynamic programming ($\mathcal{O}(n + m)$).

## Problem D

(Source: German Collegiate Programming Contest 2018 Problem M)

### Problem

Given are some pairs of points in a topographic map. For each pair find the least maximal height of a path connecting both ends.

### Intuition

- Imagine a rising water line throughout the mountain range.
- The answer for a pair is the lowest height at which it becomes possible to swim from one end to the other.

### Solution

- Store connected components in a union-find data structure. In each component, store a list of end points.
- Merge neighboring cells by increasing height.
- While merging:
  - Always merge the smaller list into the larger list.
  - If both ends of a pair are in the two lists, the current height is the answer for that pair.
- Total time complexity: $\mathcal{O}(k \cdot \log^2 k)$ where $k = \max(m \cdot n, q)$.

## Problem E

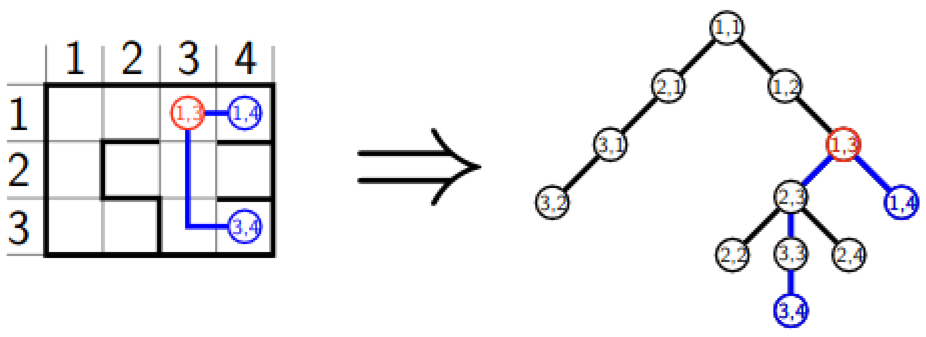
(Source: German Collegiate Programming Contest 2018 Problem A)

### Problem

Given a sequence of locations inside a maze, what is the minimal distance one has to travel to visit them all in given order?

### Observation

- The number of locations is too big for multiple iterations of any search algorithm.
- Since the maze has no loops, it can be seen as a tree.



**Solution**

- Transform the maze into a tree e.g. with *depth first search*.
- Find *Lowest Common Ancestors* to calculate the distance between two consecutive locations.
- $\Rightarrow$ Init: $\mathcal{O}(n \log n)$, Lookup: $\mathcal{O}(1)$

## Problem F

(source: XXII Open Cup, Grand Prix of Southeastern Europe Problem H)

The majority color, if it exists, is unique, because otherwise, the total number of nodes with one of the majority colors would be greater than the total number of nodes in the subtree. Let's iterate through the $n$ possible colors and calculate the number of subtrees with this color being a majority color.

Given a fixed candidate majority color, for all vertices set $s_v = +1$ if it is of this color, and $s_v = -1$ otherwise. The problem is now to find the number of subtrees whose sum of $s$ values is strictly positive. This can be done with a standard $dp[v][sum]$, meaning the number of subtrees of $T$ restricted to the subtree of $v$ which contain node $v$ and have their sum of $s$ values equal to $sum$. It seems like this works in $O(n^4)$, but let's look at it more closely.

Let's say that the number of nodes with color $c$ is $k_c$. Then, when we calculate the dp for this color, $sum \leq k_c$. Additionally, note that we only care about states with $sum > -k_c$, because we won't be able to reach a positive $sum$ from smaller values. Also $|sum| \leq sz_v$, where $sz_v$ is the size of the subtree of node $v$. Thus, we can only calculate $dp$ for $|sum| \leq \min(sz_v, k_c)$. It is well-known that such dp works in $O(n \cdot k_c)$. Therefore, the total time complexity is $\sum_c O(n \cdot k_c) = O\left(n \cdot \sum_c k_c\right) = O(n^2)$.