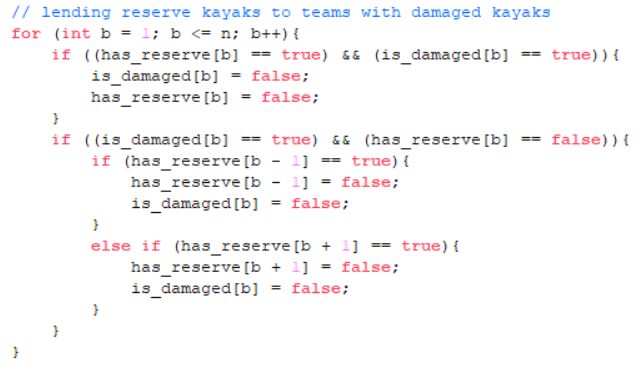
Jennica Ramones

Natjecanje



In this problem, you are hosting a kayaking competition and must figure out what is the minimum number of teams that can’t participate in the race. The first line of input is three consecutive numbers (n, s, r). **n** is the total number of teams, **s** is the number of teams with damaged kayaks, and **r** is the number of teams with reserve kayaks. If a team has a damaged kayak and its own reserve kayak, it will use its own kayak regardless. If a team has a spare kayak, it will *only* lend to the teams besides it. This is the part of the program where the core algorithm is implemented.

n in W(n) would be the size of the array. n is bounded, 2 <= n <= 10, s is bounded, 2 <= s <= n, and r is bounded, 2 <= r <= n. In a best-case scenario (besides one where there’s no damaged kayaks at all) where all teams have damaged kayaks and need to use their own reserve, the amount of work is W(n) = n (where n is the size of array).

Now say there are teams with damaged kayaks and reserved kayaks and teams with damaged kayaks and no reserves. To optimize the algorithm, I would put the first if statement in a separate for loop to reduce the amount of work for the second if statement. Now it’s been optimized, and there are only teams with damaged kayaks and no reserves (teams that are seeking reserves).

If it’s seeking a reserve, it first looks to the team on the left and then looks to the team on the right. The worst-case scenario would be a race with majority of teams having damaged kayaks. The work would be an additional (2^n) since it looks in both directions. Ultimately, W(n) = n + (2^n)