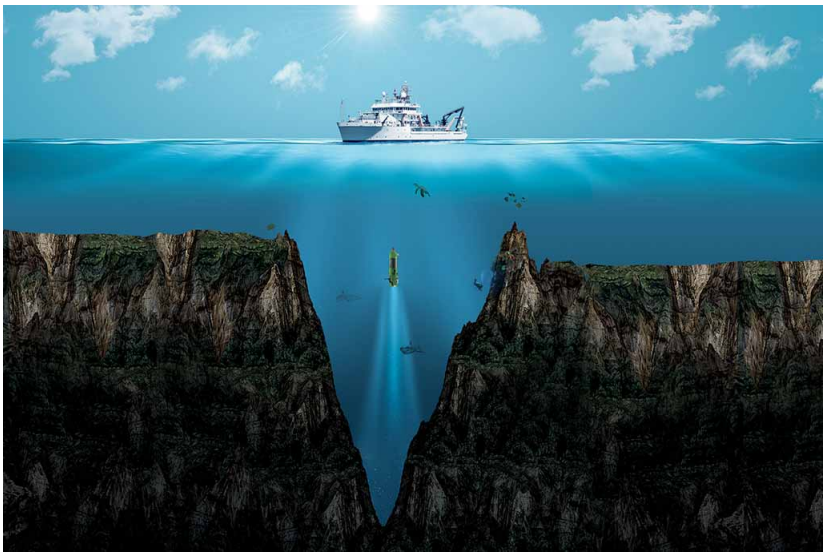
PA3: Depth Finder

This programming assignment is to be completed in either Java or Python (your choice) and submitted on Gradescope.

Collaboration is restricted to just 2 people (other than yourself), and it must be "Whiteboard Only". In summary, this means that you may discuss the assignment with 2 others, but you cannot preserve any artifacts (digital or physical) from these discussions. In particular, this means you may not take and keep photos, notes, video, audio recordings, whiteboard contents, etc. from any discussions.

All collaborators and other resources consulted should be listed as a comment at the top of your code submission. Course staff and official course materials do not need to be listed.



Motivation

A new board called "Depth Finder" is hitting the shelves. The goal of this game is to find a trench in your opponent's sea floor. Each player's board is a list of n distinct floating point values representing the depth of the sea floor at various locations. You and your opponent alternate turns "pinging" locations on your opponent's sea floor, with the goal of finding a trench (i.e. a local maximum depth). To guarantee that a local minimum exists, the first two and last two depths must be 1, 2, 4, and 3 respectively. The winner is the first to find and declare a trench of their opponent's.

Problem Statement

You will write a divide and conquer algorithm which efficiently finds a local maximum in your opponent's list of depths. Specifically, you will write an algorithm which can find and declare a local maximum value in your opponent's board in no more than $3 \cdot \lceil \log_2(n) \rceil$ "pings", where a ping is one request for the depth of a particular location. To find a local maximum depth you must have pinged three consecutive locations where the middle location is larger than the other two. To declare a local maximum you must clearly state this found index as your "final answer".

For example, if we had the board below of size 10, indices 4 and 6 each contain a local maximum. You would win the game by either pinging locations 3, 4, and 5, then declaring 4 as your final answer, or else by pinging locations 5, 6, and 7 then declaring 6 as your final answer.

1	2	4.5	5.7	7	6.2	9	8.6	4	3
0	1	2	3	4	5	6	7	8	9

Starter Code Summary The starter code provided consists of three files:

– **GameBoard.java / gameboard.py**: This is implements an object to represent the gameboard. When constructing a gameboard object you simply provide an integer value to constructor (this value must be at least 5). The program then sets up the board, randomly selecting one location to be the local maximum. To ensure you don't peek, the board generates depths when pinged, rather than in advance. There are only 2 methods/attributes that your submission may access in Gameboard:

  – `ping(location)`: This is the ping method. The argument is the index you would like to ping, the output is a float(python)/double(java) representing the depth at that location. If you exceed maximum allotted ping count then the method will print a statement indicating this, and then only return 0.

  – `size`: This attribute represents the size of the board (i.e. the length of the list of depths)

  For debugging purposes a `toString()`(java)/`str`(python) method is provided for nice printing of the board. *You will not submit this file.*

– **PlayGame.java / playgame.py**: `This is the file you will modify and submit.` In here there is the header of just one method: `playgame_dc(gb, left, right)`. You should implement this function to be a divide and conquer algorithm to play the game. `gb` is the gameboard, `left` represents the leftmost index at which a local maximum may appear, and `right` represents the rightmore index at which a local maximum may appear. Your algorithm should return the index of a local maximum in the given gameboard. Your algorithm must also follow the divide and conquer paradigm. In particular, it must include a conquer step consisting of recursive calls on one or more subproblems.

– **GamePlayer.java / gameplayer.py**: This is the file you will run. It takes an integer as input in the console to represent the size of the board, creates a gameboard object using that size, calls the divide and conquer algorithm you will implement using that gameboard with `left = 0` and `right=size`, then gives the valued retured as the "final answer". You should see "trench found!" printed to the console if you correctly found the trench within the allotted pings. Remember, to correctly find the trench you must have also pinged on both sides of it to see that the values are smaller. *You will not submit this file.*

Input Your input will be a single integer represening the board size.

Output The function you implement will return a single integer representing the index of a local maximum in the given gameboard.

Running Time Your algorithm must find the local maximum using no more than $3 \cdot \lceil \log_2(n) \rceil$ pings.