Please enter your name and uID below.

Name: Jason Crandall

uID: u0726408

Submission notes

- **(PS 1 specific) This is a warm-up assignment. Full points will be given for answers that have some correct elements and that clearly been given a legitimate effort.**

- *Solutions must be typeset* using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the semester, so please retain the original files.
- Please remember that for problem sets, collaboration with other students must be limited to a high-level discussion of solution strategies. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

1.

For my solution to the programing task of problem set 3, I began with an initial set of all the islands found in the problem. With that as the setup, for the parameters of my recursive function, I have a subset of the islands representing those included in the solution, a subset of islands representing the islands that have connections, a number representing the current island we are on, and finally a number representing the number of islands found in the solution thus far. Of these parameters, the exploration state is the subset of island connections, as it is only when this subset matches the original set that we know we have found a correct solution. With these in place, the first call made to the recursive function will be an empty set for the partial solution, an empty set for the connections, 1 for the current island, and 0 for the current island count.

Once in the execution of the function, a minimal solution that can be found is if the partial solution is if the subset of connections equals the original island set, then the program is complete. Some of the non-solution base cases are if the current island is greater than the max island value, meaning we went out of bounds. Another non-solution is if the current island count is greater than the best count we have found thus far. To further bind and prune the solution I'll first go over the decisions that my function makes. It essentially breaks it down to will it include the current island in the partial solution, or will it not include it in the partial solution. Knowing this, a pruning condition I have on whether or not to add the current island is to check whether the connections of the current island are already included in the subset of connections. If it is, then it would be redundant to do so again. The next check I have to prune further before deciding not to include an island is to check if it is an orphan island. If it is an orphan, then it must be added to the partial solution and thus cannot go down the path of not including it.

2.

To successfully complete the algorithm for the game, the inputs to the recursive function will be an array of choices, a score, and an index of the array of the cards. The exploration state in this example is represented by both the score and the index of the array past in, as they are how we determine if a score is a minimum score given a position in the card array. The parameters to the first call of the function in this case would be an empty array, 0, and 1. Potential base cases for this algorithm would be if the length of the card array is 0, then the score would be 0. Next, if the index of the array is greater than the size of the array, then the score would be the smallest score found thus far. In order for this to work, the decision is based around the previous choices, as if the previous 3 choices are not HI, then choose HI, else if the previous 3 choices are not LO, then choose LO. There is no need for pruning in this tree as we are trying to find the absolute minimum score, which does require us to travers through the entirety of the tree, as any value of a card could potentially cause the lowest score.

As a clearer representation of this algorithm, below is a representation for a recursive formula:

$$findLowestScore(response, score, C, i,) \begin{cases} 0 \ if\ |C| = 0 \\ score \text{ if } i > |C| \text{ and score } \leq \text{ lowest} \\ \text{if previous 3 responses are not HI:} \\ findLowestScore(response + HI, score - C[i], i + 1) \\ \text{if previous 3 responses are not LO:} \\ indLowestScore(response + LO, score + C[i], i + 1) \end{cases}$$

Essentially this algorithm will loop through the entirety of the tree, which in this case is a binary search tree, with the left node being the case where HI is selected, and the right case being if LO is selected. The only branches of the tree that we do not travers are those where we selected the same decision for the past 3 times, which is why we keep track of it in the response array as our partial. As we traverse, when we reach a leaf node is when it will check if our score is lower than the lowest found score. This will thus go down each possible path, then backtrack up and proceed to the next path until every one has been traversed, giving us the absolute lowest possible score.