Recursion + Dynamic Programming

1. A most curious game

Sam (Player 1) and Terra (Player 2) take turns playing a game, with Sam starting first.

Initially, there is a number N on a whiteboard.  On each player's turn, that player makes a move consisting of:

1. Choosing any x with 0 < x < N and N % x == 0.
2. Replacing the number N on the chalkboard with N - x.

If a player cannot make a move, they lose the game.

Write a recursive function solveGame(N) that simulates gameplay and returns True if and only if Sam wins the game, assuming both players play optimally.

Examples:

solveGame(2) → true (Sam picks x = 1, Terra has no moves and loses)  
solveGame(3) → false (Sam picks x = 1, Terra picks x = 1, Sam has no moves and loses)

solveGame(4) → true (Sam picks x = 1, Terra picks x = 1, Sam picks x = 1, Terra has no moves and loses)

**Answer:**

public static boolean solveGame (int N) {

if (N<=1) return false;

if (N==2) return true;

for (int i=1; i<N; i++){ //Sam's turn

if (N%i==0) {

int x=i;

N=N-x;

if (!solveGame(N)) return true;

}

}

return false;

}

### 2. The game Continues

Write solveGame as a Dynamic Programming algorithm. Your algorithm must run in O(N) time where N is the initial number on the whiteboard.

public static boolean solveGame (int N) {

if (N<=1) return false;

if (N==2) return true;

for (int x=1; x<N; x++){ //Sam's turn

if (N%x==0) {

N=N-x;

if (!solveGamedp(N)) {

return true;

}

}

}

return false;

}

public static boolean solveGamedp (int N){

boolean [] dp = new boolean [N+1];

for (int x=1; x<=N; x++){

if (x%2==0) dp[x]=true;

if (x%2==1) dp[x]=false;

}

return dp[N];

}

## Orders of Growth/Runtime

These questions are more conceptual. The last worksheet will be entirely "find the runtime of a function".

### 3. Multiple Runtimes

An algorithm has two different runtimes, say O(log(n) and O(n), how is this possible? For full credit, list two distinct possibilities.

An algorithm has worst, average, and best case runtimes. Sometimes the runtimes can differ, with best case runtimes being significantly faster than the average and worst case.

Binary Search Tree has a worst case runtime of O(n) and best can runtime of O(log(n)).

We could have different runtimes depending on what we define n as

### 4. Runtime Ordering

Group the following runtimes from **most** efficient to **least** efficient. Indicate any complexities that are equivalent (i.e. O(x) = O(y)).

O(c), where c is a constant.

O(n)

O(n log n)

O(n log c), where c is a constant.

O(cn), where c is a constant.

O(log n)

O(n!)

O(1)

O(nc), where c is a constant.

O(nn)

O(c) = O(1) Most efficient

O(log n)

O(n) = O(n log c)

O(n log n)

O(nc)

O(Cn)

O(n!)

O(nn) Least efficient

Trees

5. Identity crisis

For the following Tree, select all that apply (bold and color red the bullet point):

Diagram

Description automatically generated

* **It is a binary tree**
* **It is a complete tree**
* It is a binary search tree
* It is a min binary heap
* **It is a max binary heap**
* It is an AVL tree

6. Counting odd subtrees

Write a method countOddSubtrees that takes in a BinaryTreeNode root and returns the number of odd subtrees that exist within the tree. Hint: consider writing "sizeOfTree" first as a helper. **Extra challenging version:** write this in O(n) time, where n is the number of nodes in the tree.

public static int countOddSubtrees(BinaryTreeNode <Integer> root){

  if (root == null) return 0;

  return countOddSubtreesHelper(root.getLeftChild()) + countOddSubtreesHelper(root.getRightChild());

}

private static int countOddSubtreesHelper (BinaryTreeNode <Integer> root){

  if (root==null) return 0;

int count = 0;

if (sizeOfTree(root) % 2 ==1) {

  count = 1;

}

return count + countOddSubtreesHelper (root.getLeftChild()) + countOddSubtreesHelper(root.getRightChild());

}

private static int sizeOfTree (BinaryTreeNode <Integer> root){

if (root == null)

return 0;

else

return sizeOfTree(root.getLeftChild()) + 1 + sizeOfTree(root.getRightChild());

}

Graphs

7. Example Graph

Given the following graph, answer the following questions:

Diagram

Description automatically generated

* What would be the print order if you ran BFS starting on node D? Assume you order neighbors by the smallest labeled neighbor first. D, B, E, A, C
* What would be the print order if you ran DFS starting on node D? Assuming you order neighbors by the smallest labeled neighbor first. D, B,A, C, E
* What would be the Minimum Spanning Tree of this graph? List all the edges used.

A-C, B-C, B-D, D-E

* Running Dijkstra's on this Graph starting at node B, which node would we visit first after B? Which node would we visit last? We visit C first, We visit E last

P and NP

8. Fact Sheet

Place an x in the cell for each statement that is ALWAYS true for that complexity class.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | P | NP | NP-Complete | NP-Hard |
| All problems can be verified in polynomial time. | X | X | X |  |
| All problems can be solved in polynomial time (assuming P = NP) | X | X | X |  |
| All problems can be solved in polynomial time solutions (assuming P != NP) | X |  |  |  |
| All problems are at least as hard as the hardest problems in NP |  |  | X | X |
| Contains problems that are unsolvable |  |  |  | X |
| Is completely within NP | X | X | X |  |
| Are actually the same complexity class, assuming P = NP | X | X | X |  |

Misc.

9. Name that Algorithm

Name one algorithm or data structure (of the ones taught in class) that you would use to

solve the following problems most

1. BFS
2. Finding the shortest driving route home from class. Djikstra’s
3. Figuring out a valid order to do chores, when a bunch of chores have dependencies on various other things that need to be done first. Topological sort
4. Sorting large numbers of bank accounts by the amount of money held within the account, knowing that each account has between $10 and $200 in it, but is not uniformly distributed. Counting Sort
5. Building a digital dictionary that supports insertion, searching, and deletion of words. Tries
6. Building a fiber optic data network using the least amount of cable possible. Prim’s Algorithm