Quiz 2

This quiz is **open-book and closed-internet**. Feel free to use any physical materials you have brought, but you may not access resources online (except funprog). Proctors will be available to answer administrative questions and clarify specifications of coding problems, but they should not be relied on for coding help.

Problems are worth 30, 30 and 40 points, respectively, for a total of 100 points.

You *must* submit your quiz via funprog before the deadline in order to receive credit. This quiz assumes you have Python 2.7 installed on your machine.

The `resources` directory contains **Python documentation** for commonly-used data structures.

Problem 1: `remove_first_repeated`

Implement `remove_first_repeated`, which given an array `A` produces a new array `A_dedup` with the 2nd occurrence of the 1st repeated element of `A` removed. If none of the elements of `A` are repeated, `A_dedup` should equal `A`.

INPUTS:

- `A`, an array of integers.

OUTPUT:

A new array equal to `A`, but with the second occurrence of the first repeated element removed.

EXAMPLES:

- `remove_first_repeated([0,1,2,3,4]) = [0,1,2,3,4]`
- `remove_first_repeated([0,1,2,1,3,4]) = [0,1,2,3,4]`
- `remove_first_repeated([0,1,2,3,1,1,4,4]) = [0,1,2,3,1,4,4]`

Problem 2: `cherrypick`

Implement `cherrypick`, which checks whether some selection of `n` elements of an array `A` add up to `required_sum`.

INPUTS:

- `A`, an array of integers.

- `n`, positive integer number of elements to pick from A.
- `required_sum` - integer number that the `n` elements of `A` add up to.

OUTPUT:

Boolean, `true` if there exists a selection of exactly `n` elements of `A` that sum to `required_sum`. If no such selection exists, return `false`.

EXAMPLES:

- `cherrypick([0,1,2,3], 2, 5) = true` because `2+3=5`
- `cherrypick([0,1,2,3], 2, 6) = false` because no two elements of `A` sum to `6`.
- `cherrypick([-2,-1,0,1,1], 4, 0) = true` because `-2+0+1+1=0`.

NOTE:

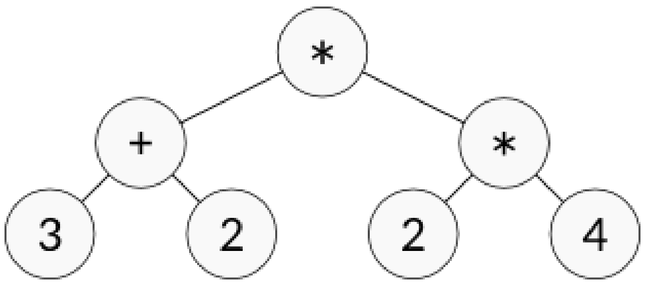
You are not allowed to import Python libraries that provide code for combinations and permutations (e.g., itertools). Solve the problem recursively!

Problem 3: `eval_ast`

Consider a data structure `AST`: a binary tree with children `"left"` and `"right"`, and a value given by `"node"`. The `node` is either an integer or an operation: add (`"+"`) or multiply (`"*"`). If the node encodes an operation, the operands are given by the children of this node: ``"left" ` and `"right" `. If` node ` is neither `"+" ` nor `"*"`, it must be an integer and the node has no children.

The `AST` encodes an algebraic expression. For example, `(3 + 2) * (2 * 4)` is encoded by the following `AST`:

```
AST_1 = { "node":   "*",
          "left":   { "node": "+",
                      "left": 3,
                      "right": 2 },
          "right":   { "node": "*",
                      "left": 2,
                      "right": 4 }
        }
```



INPUTS:

- `ast`, a dictionary, as defined above.

OUTPUT:

An integer, the result of evaluating the algebraic expression the AST encodes.

EXAMPLES:

- `eval_ast({"node": 1}) = 1`, trivially
- `eval_ast({"node": "+", "left": 1, "right": 4}) = 5` because `1 + 4 = 5`.
- `eval_ast(AST_1) = 40` because `(3 + 2)*(2 * 4) = 5 * 8 = 40`.