```
// int min_error_value = null
// arraylist successful_pair_list = null
// create file obj that all functions can see?
// create root
// populate root
// set current_node to first child
// run recursive function on current_node
// LABEL:
// if successful_pair_list is still null, print no possible solution
// else print successful pair list to file


  // recursive function:
    // inputs: node object
    // output: node object
    // run tests on node
    // if tests return -1, terminate node
    // else if getListSize(Node) = 8:
      // compare with min_error_value
      // if min_error_value = null, set value and set current pair list to successful_pair_list
      // else, if return value is less than min_error_value,
        //set mev to return value and set current pair list to successful_pair_list
      // terminate node
    // else if tests return other and successful_pair_list != null:
      // compare with min_error_value
      // if min_error_value = null, set value
      // else, if return value is less than min_error_value, set mev to return value
      // else terminate node
    // populate node
```

```
    // current_node = current_node.children[0] (the first element of the list)
        // note: the first element of the list will alwayse be untested
    // run recursive function on current node
```

//check_validity function:

```
    // input: node object
    // output: -1 if there is a hard constraint violation, in value of soft constraints otherwise
    // create var to store output: value
    // generate list of pairs
    // run no duplicates function (checks for duplicate tasks)
    // if failure, set value to -1
    // run three hard constraint functions
    // if failure, set value to -1
    // if var != -1:
            // run soft constraint functions
            // set value to sum of their output
    // return value
```

// terminate node function:

```
    // input: node to be terminated
    // output: ?
    // terminate node (method) (returns parent)
    // END CONDITION: if parent == null (this is the root)
        // break to LABEL
    // if parent has no children: run terminate node function
    //else:
        // set current node to first child
        // run recursive function
```