6. (10 pts) Design a recursive algorithm that counts up all the ways that change can be made using an unlimited number of coins of given de-nomination. For example, using the standard Canadian coins (including the obsolete penny), $0.28 can be made in 13 distinct ways:

- one quarter and three pennies
- two dimes, one nickel, and three pennies
- two dimes and eight pennies
- one dime, three nickels, and three pennies
- one dime, two nickels, and eight pennies
- one dime, one nickel, and thirteen pennies
- one dime and eighteen pennies
- five nickels and three pennies
- four nickels and eight pennies
- three nickels and thirteen pennies
- two nickels and eighteen pennies

• one nickel and twenty-three pennies

• twenty-eight pennies

Show that all possibilities are counted – i.e. that your algorithm is correct. What is its time complexity?

Solution:

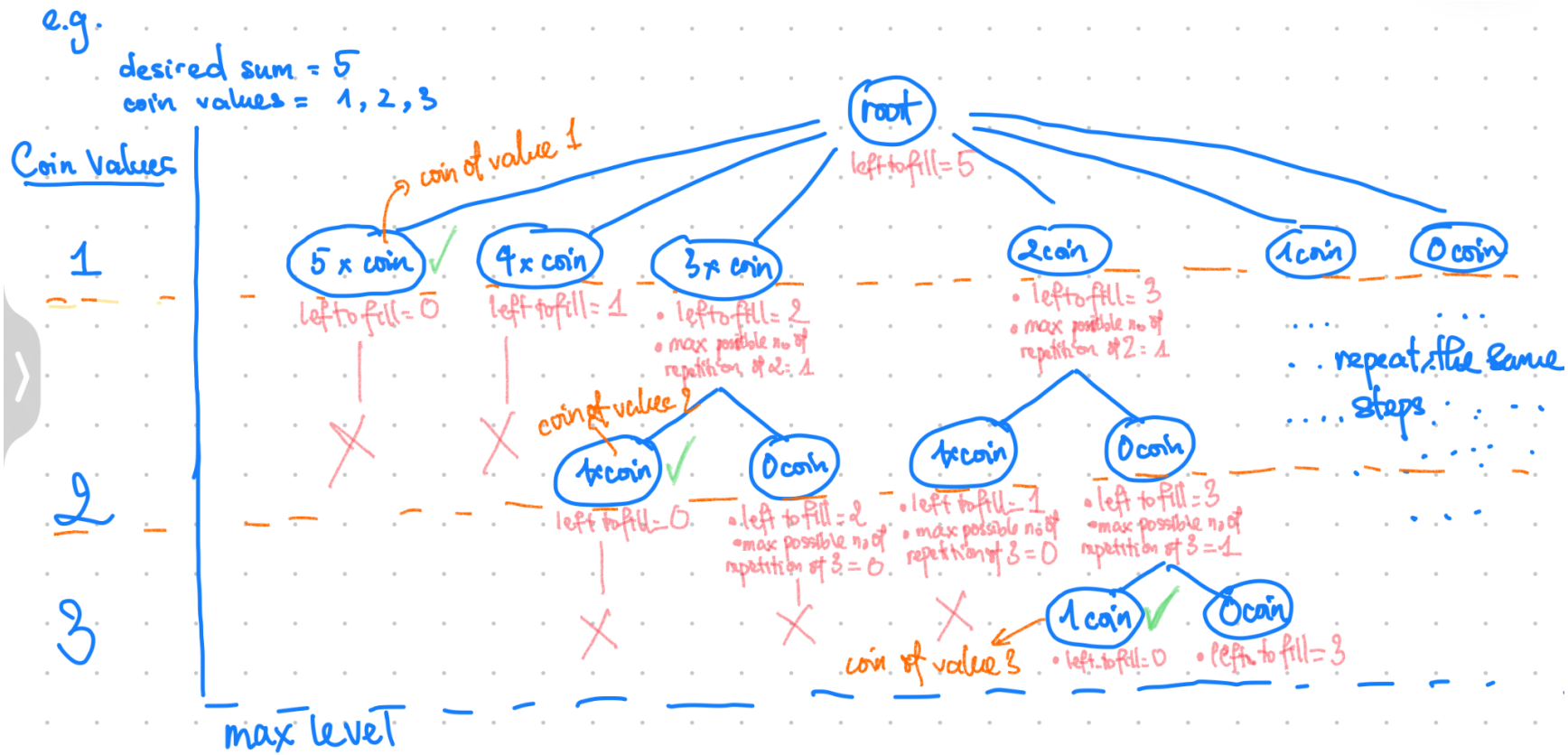
Max level = number of the coin values there are

Coin values: list of possible coins values, sorted in **ascending order**

Left to fill: the amount left to fill in at each node

Current search: the number of repetitions of each coin values

E.g. coinValues=[1, 2, 3], currentSearch=[5,2,1] means 5 coins of value 1, 2 coins of value 2 and 1 coin of value 1



Example of calling **search([], 0, 5)**

function search(int[] currentSearch, int currentLevel, int amtToFill):

```
    // If the amount to fill is 0, then we have found a solution
    if amtToFill == 0:
        possibilities.add(currentSearch)
        count += 1
        return;
```

```
// If we have gone pass the max level, then we have gone through all coin values
else if currentLevel > maxLevel:
    return;

int currentCoinValue = coinValues[currentLevel]
// If the current coin value is smaller than the amount to fill, then we can use it
if currentCoinValue <= amtToFill:
   int maxNumRepeat = Math.floorDiv(amtToFill, currentCoinValue);
   // i is a possible repetition of the current coin value
   for (int i = maxNumRepeat; i >= 0; i--)
      int currentNodeAmt = i * currentCoinValue;

      // if the amount of current node is already larger than the amount to fill, no need to search further down
      if (currentNodeAmt <= amtToFill)
         int[] newSearch = currentSearch.clone()
         newSearch[currentLevel] = i
         if (currentLevel <= maxLevel):
            search(newSearch, currentLevel + 1, amtToFill - currentNodeAmt)
```

Proof by induction

Preconditions:

- coinValues: a sorted list of coin values (length >= 1)
- currentSearch: a list of the same length as coin values
- currentLevel: >=1 and <= number of coin values
- maxLevel: the number of coin values
- amtToFill: the amount to fill

Let:

- coinValues = $[c_1, c_2, c_3, \ldots, c_k]$
- currentSearch = $[v_1, v_2, v_3, \ldots v_k]$
- currentSearch at currentLevel would be:

$$[v_1, v_2, v_3 \ldots v_{currentLevel}, 0, 0, \ldots 0] \quad (v_k)$$

Statement: search(currentSearch, currentLevel, amtToFill) will add all search results $[v_1, v_2, v_3 \ldots, v_k]$ such that

- the first currentLevel values of S & currentSearch are the same
- the rest of the values of S will satisfy

$$\frac{\begin{aligned} & c_{currentLevel} * v_{currentLevel+1} \\ + \; & c_{currentLevel+1} * v_{currentLevel+2} \\ & \ldots \\ + \; & c_k * v_k \end{aligned}}{amt\ to\ Fill} \quad (*)$$

* Base case:
  * amtToFill = 0

→ add currentSearch to possibilities list

↙

$[v_1, v_2, v_3 \dots v_{currentLevel}, 0, 0, \dots 0]$ ← v_k

satisfy (A)

* Hypothesis: Assume the above statement is true for all amtToFill = n ($n \in N$)

* Induction: Prove the above statement is true for amtToFill = n+1

- search(currentSearch, currentLevel, n+1)

calls these in the for-loop

(1) search(newSearch, currentLevel+1, n+1 − currentNodeAmt)

= no of repetition * coin value at current level

* For all no of repetitions > 0:
⇒ n+1 − currentNodeAmt ≤ n
⇒ (1) is correct by the hypothesis

* For no of repetitions = 0 ⇒ currentNodeAmt = 0
⇒ n − currentNodeAmt = n+1
⇒ (1) in this case:

Calls these in the for loop

(2) search(newSearch, currentLevel+2, n+1 − currentNodeAmt)

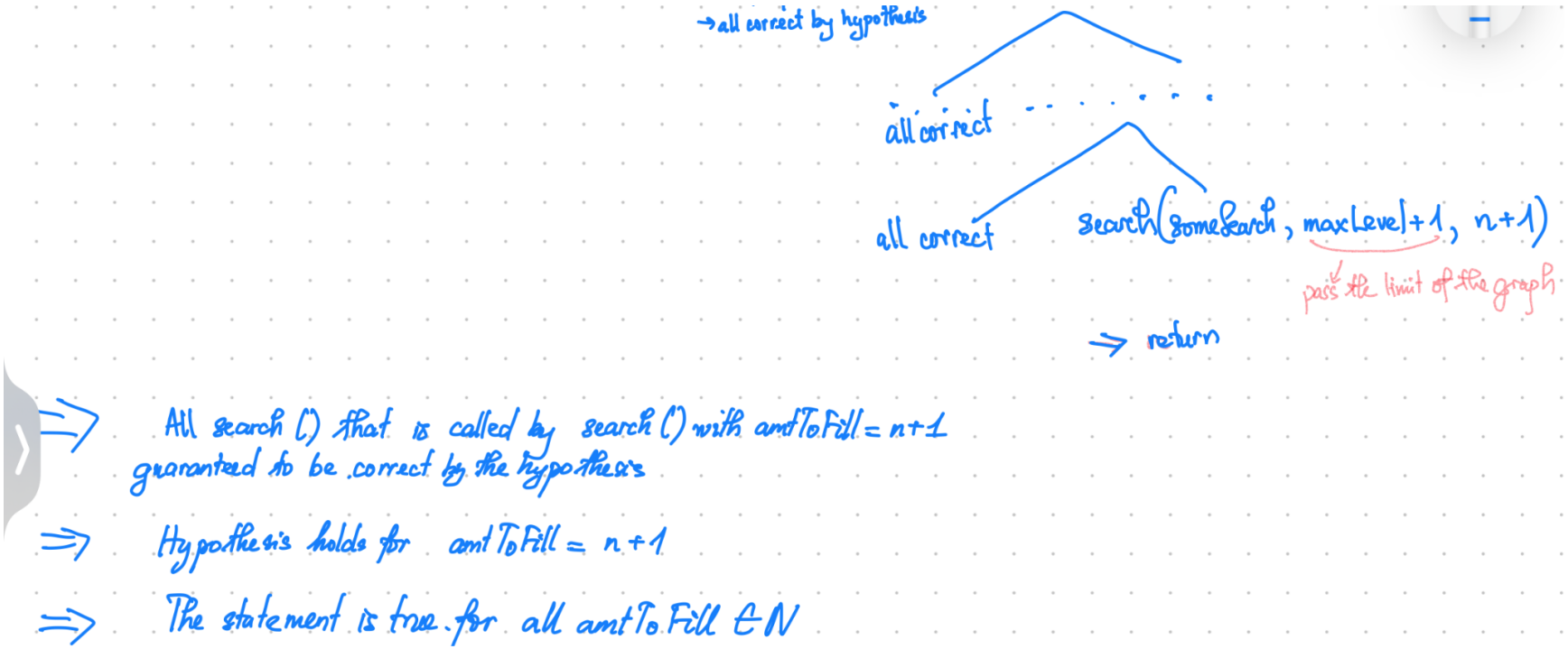
* For all no of repetitions > 0:
⇒ n+1 − currentNodeAmt ≤ n
⇒ (2) is correct by the hypothesis

* For no of repetitions = 0 ⇒ currentNodeAmt = 0
⇒ n+1 − currentNodeAmt = n+1
⇒ (2) in this case:

no of repetition > 0
→ all correct by hypothesis

no of repetion = 0

114%



- **The complexity of this algorithm:**

Branching factor: amtToFill/(coinValue at currentLevel)
Max depth: maxLevel
Time complexity: O((amtToFill/(coinValue at currentLevel))^maxLevel)