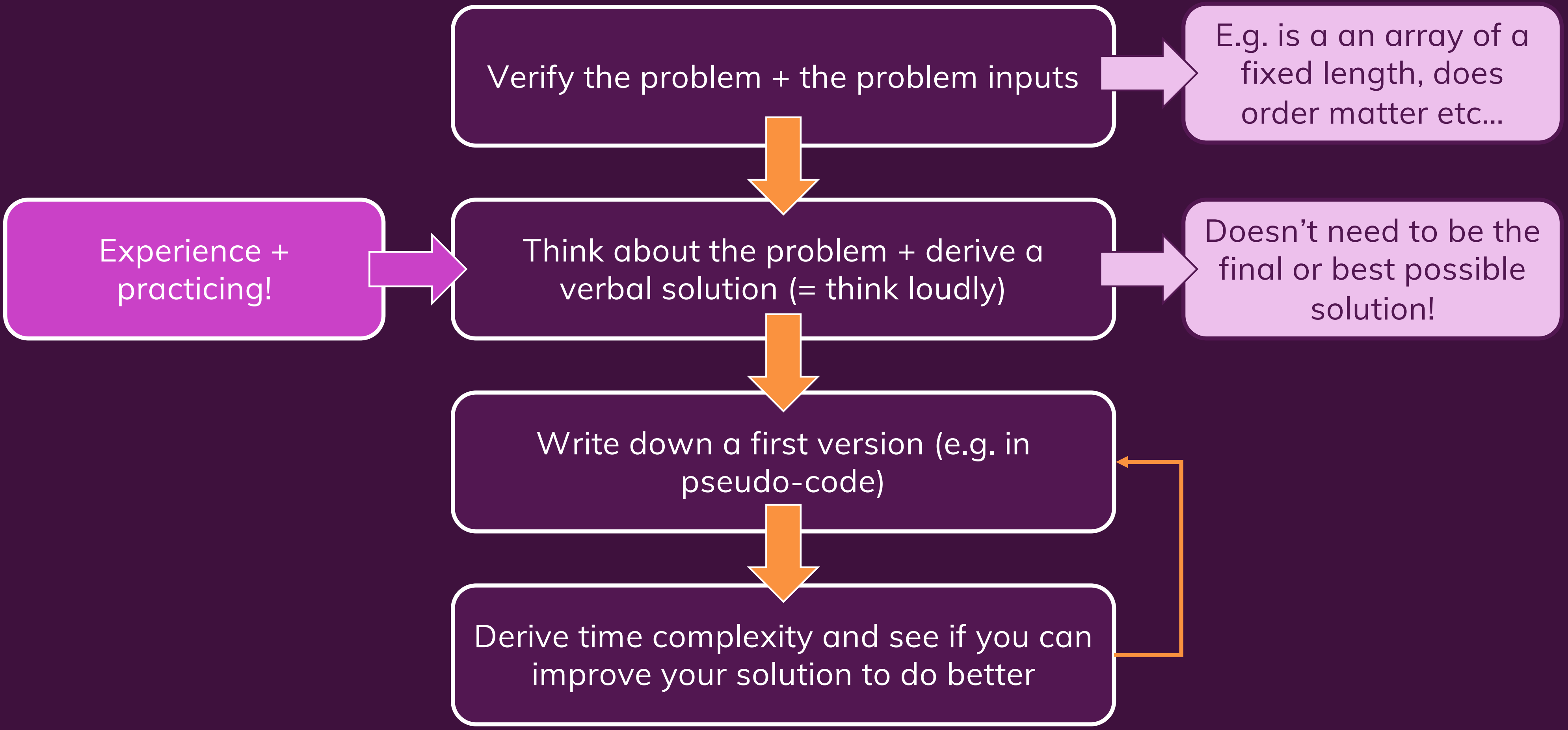
Complex Algorithms Are Complex…

Coming up with the best possible algorithm can be very hard for tricky problems

That’s why algorithms are popular in interviews: People want to see if you can solve problems

Good news: It’s NOT the best possible solution that counts, it’s your ability to come up with solutions

Solving Problems / Coming Up With Algorithms



Verify the problem + the problem inputs

E.g. is a an array of a fixed length, does order matter etc…

Experience + practicing!

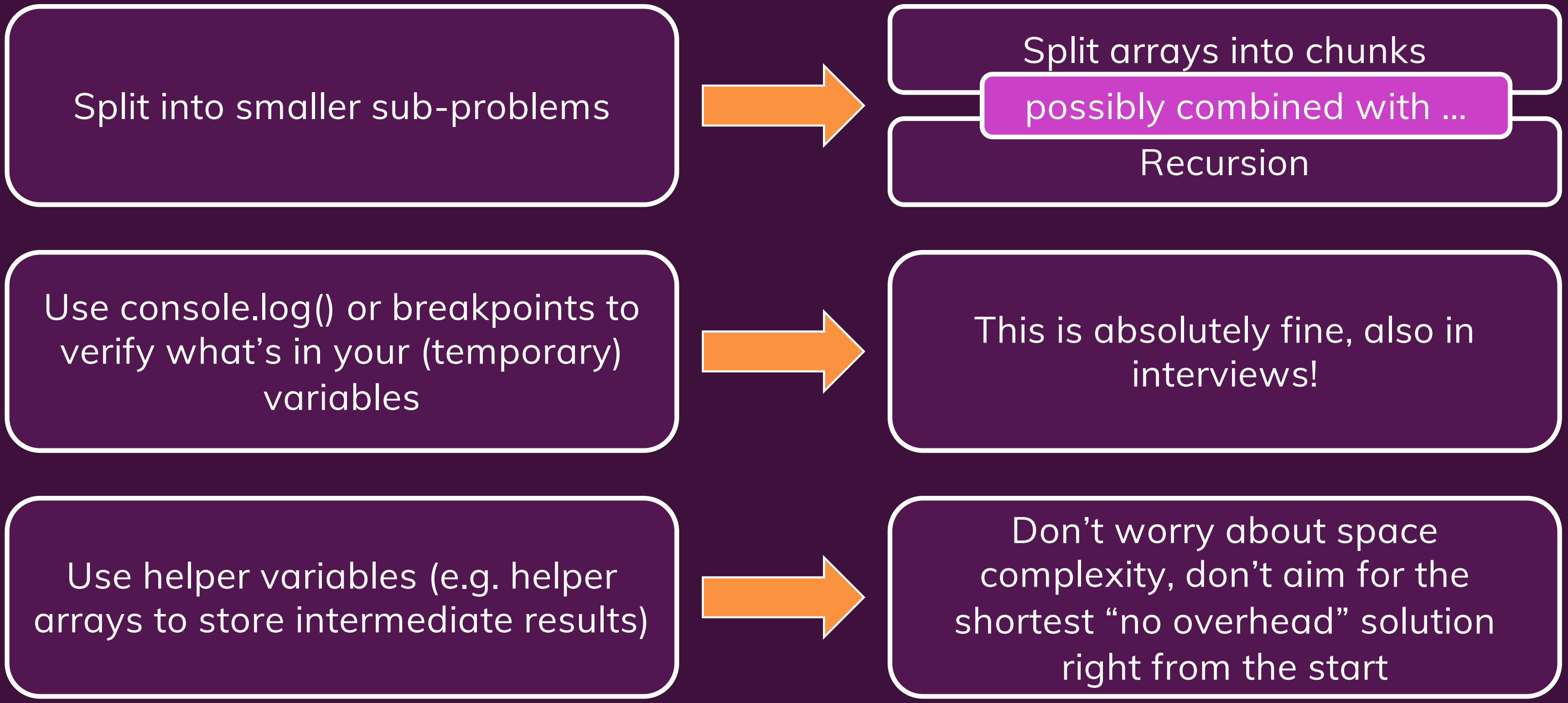
Think about the problem + derive a verbal solution (= think loudly)

Doesn’t need to be the final or best possible solution!

Write down a first version (e.g. in pseudo-code)

Derive time complexity and see if you can improve your solution to do better

Ways of Simplifying a Problem



Split into smaller sub-problems → Split arrays into chunks

possibly combined with …

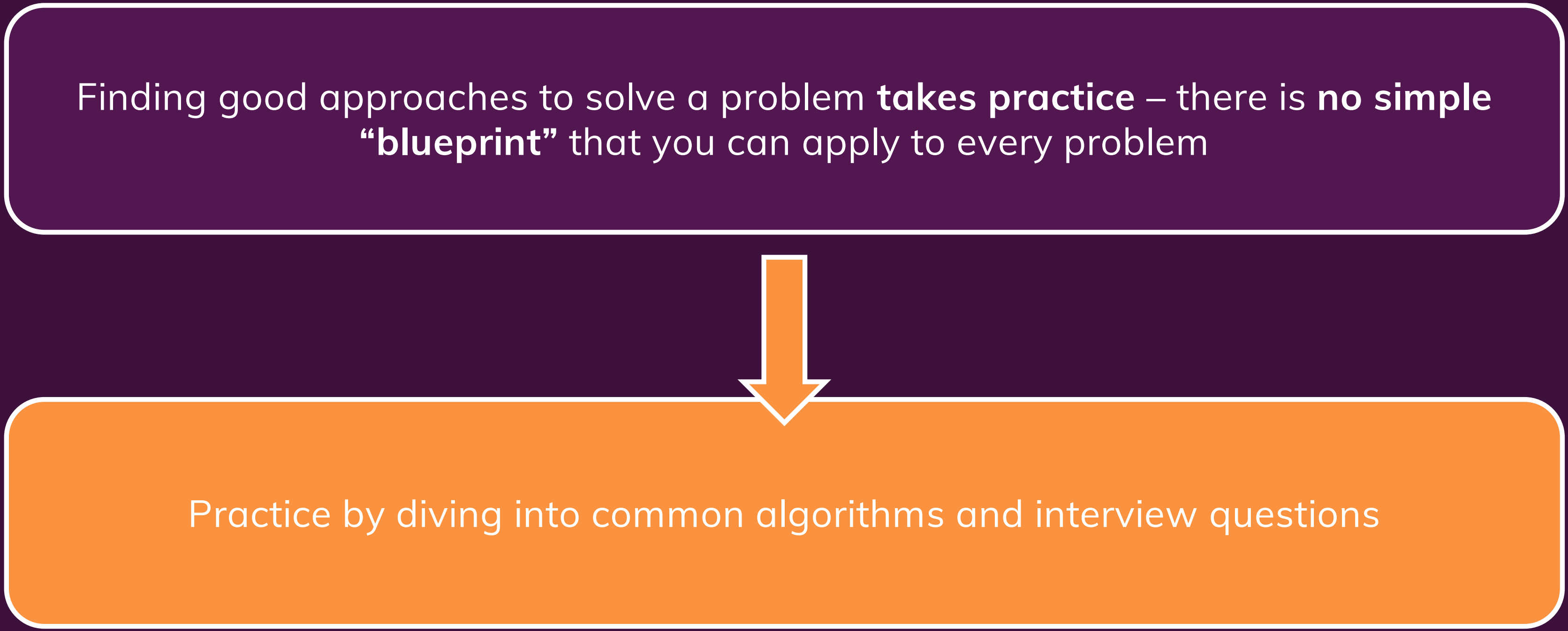
Recursion

Use console.log() or breakpoints to verify what's in your (temporary) variables → This is absolutely fine, also in interviews!

Use helper variables (e.g. helper arrays to store intermediate results) → Don't worry about space complexity, don't aim for the shortest "no overhead" solution right from the start

ACADE
MIND

Practice Makes Perfect!



Finding good approaches to solve a problem **takes practice** – there is **no simple “blueprint”** that you can apply to every problem

Practice by diving into common algorithms and interview questions

The Knapsack Problem

*You got a **list of items**, where every item has a **value and a weight**. You got a bag that holds a **maximum weight of X**.*

*Write a program that **maximizes the value** of the items you put into the bag whilst ensuring that you **don’t exceed the maximum weight**.*

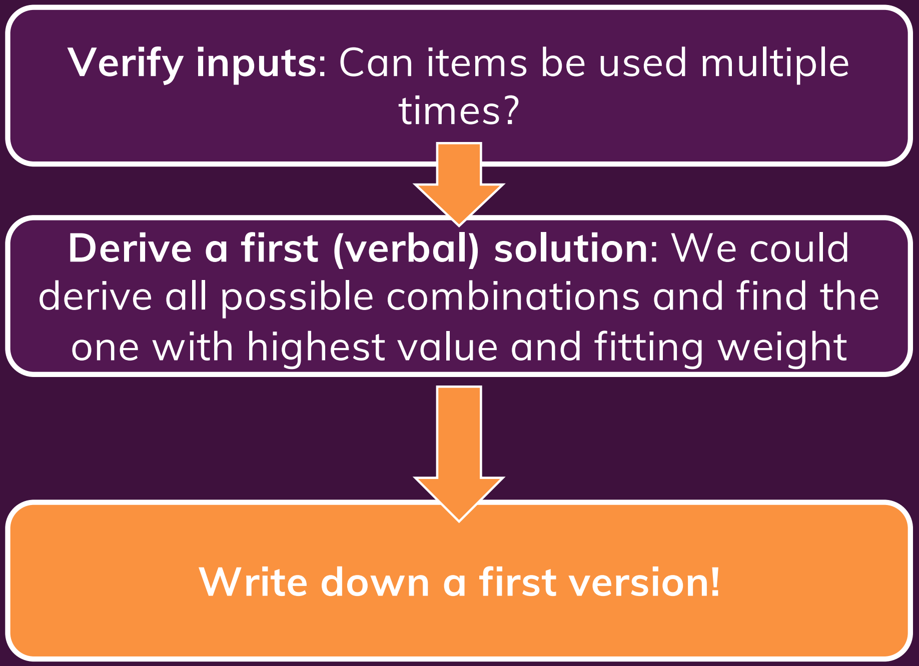
```
items = [
  {id: ‘a’, val: 3, w: 3},
  {id: ‘b’, val: 6, w: 8},
  {id: ‘c’, val: 10, w: 3}
]

maxWeight = 8

bag = [‘a’, ‘c’] // solution
```

ACADE MIND

Solving the Knapsack Problem

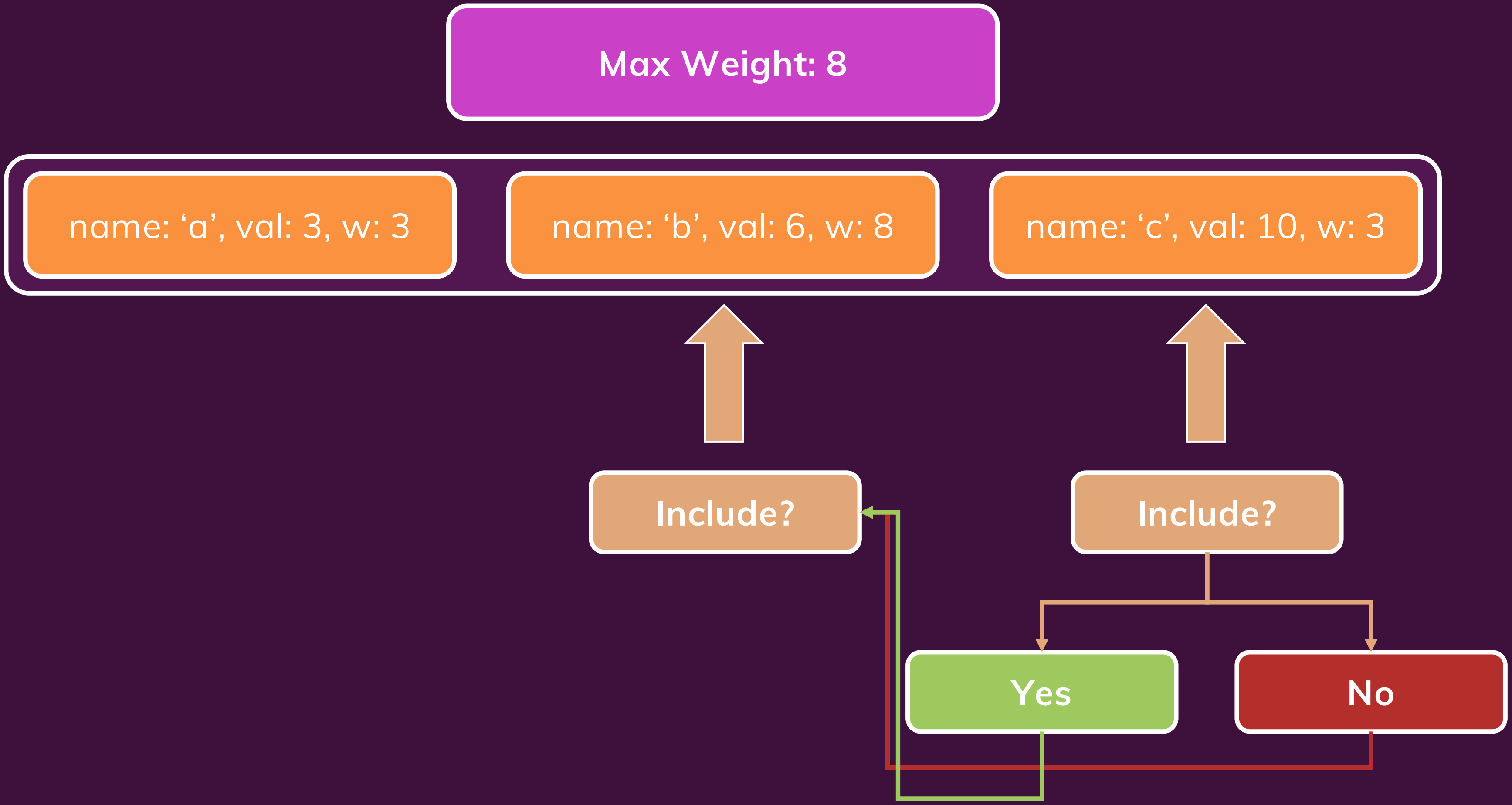


Verify inputs: Can items be used multiple times?

Derive a first (verbal) solution: We could derive all possible combinations and find the one with highest value and fitting weight

Write down a first version!

Let’s Rethink!



Max Weight: 8

name: ‘a’, val: 3, w: 3

name: ‘b’, val: 6, w: 8

name: ‘c’, val: 10, w: 3

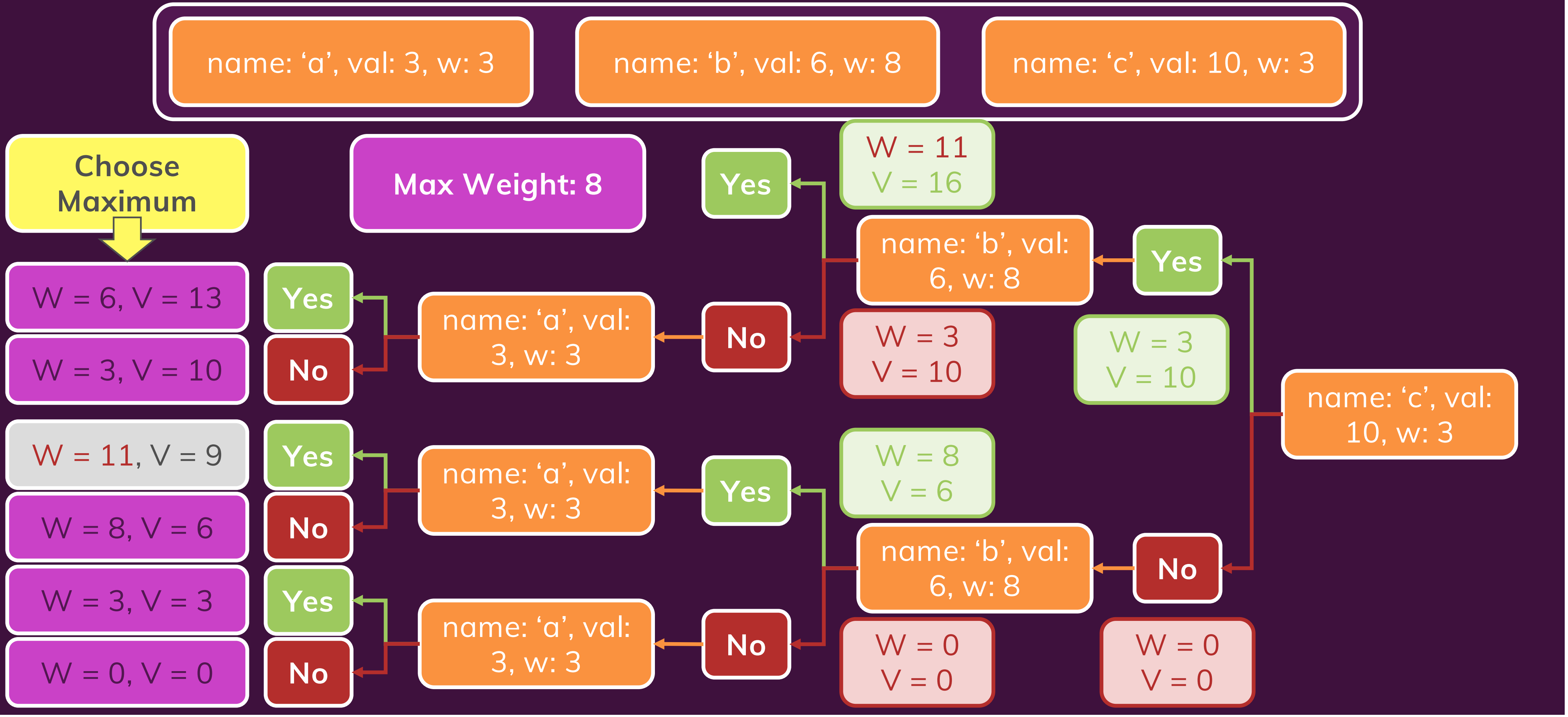
Include?

Include?

Yes

No

We Evaluate All Possible Cases / Combinations



name: ‘a’, val: 3, w: 3

name: ‘b’, val: 6, w: 8

name: ‘c’, val: 10, w: 3

Choose Maximum

Max Weight: 8

W = 6, V = 13

W = 3, V = 10

W = 11, V = 9

W = 8, V = 6

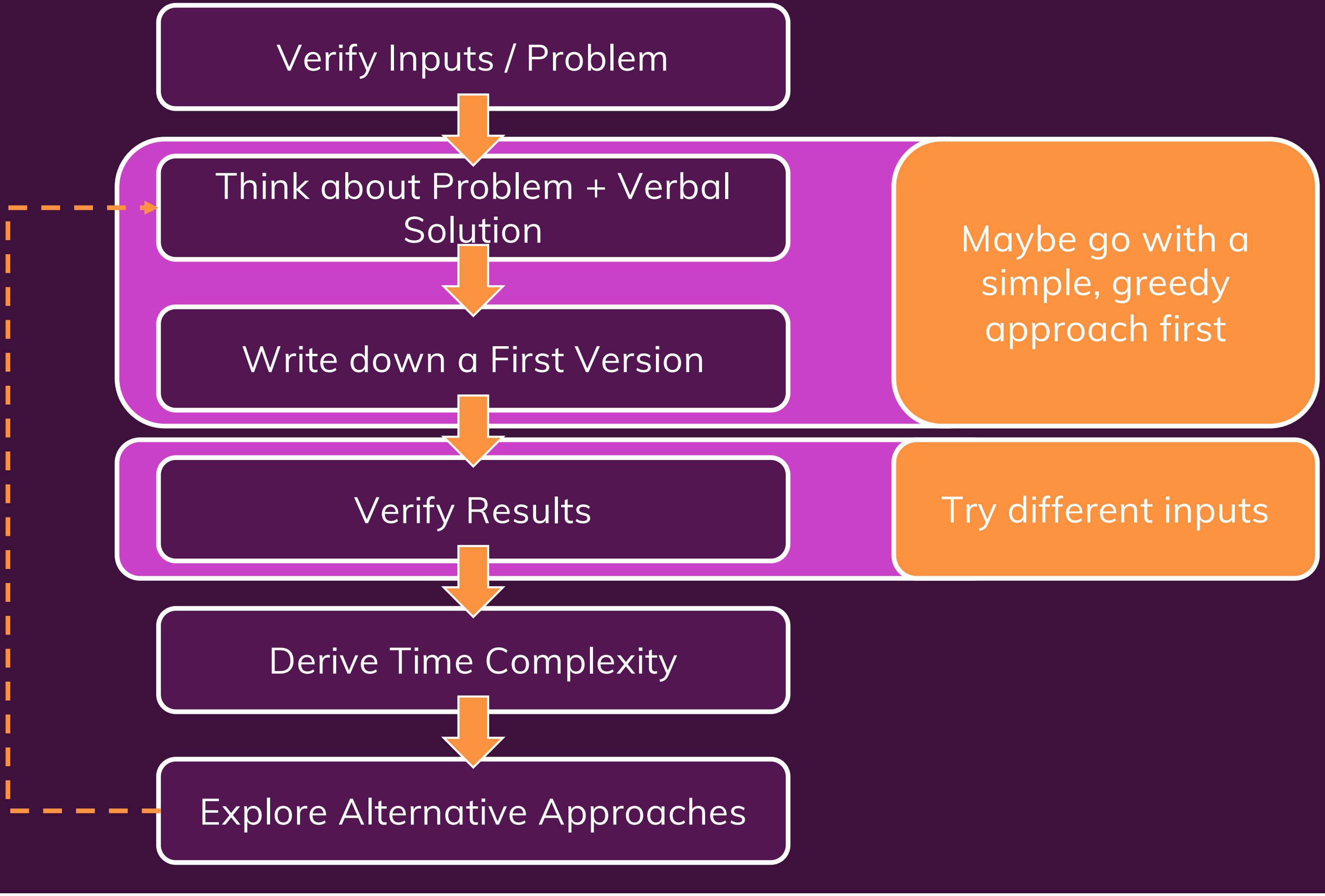
W = 3, V = 3

W = 0, V = 0

Greedy vs Dynamic Algorithms / Solutions

Greedy	Dynamic
Make best possible decision in every step and hope that it leads to the overall best solution	Evaluate all possible solutions and find overall best solution via comparison
Greedy algorithms often are faster to set up and come up with but they don't necessarily provide the best runtime and/ or result	"Divide and conquer" approach: Divide the problem into smaller, easy-to-solve subproblems

Our Final ”Problem-Solving” Plan



Verify Inputs / Problem

Think about Problem + Verbal Solution

Write down a First Version

Maybe go with a simple, greedy approach first

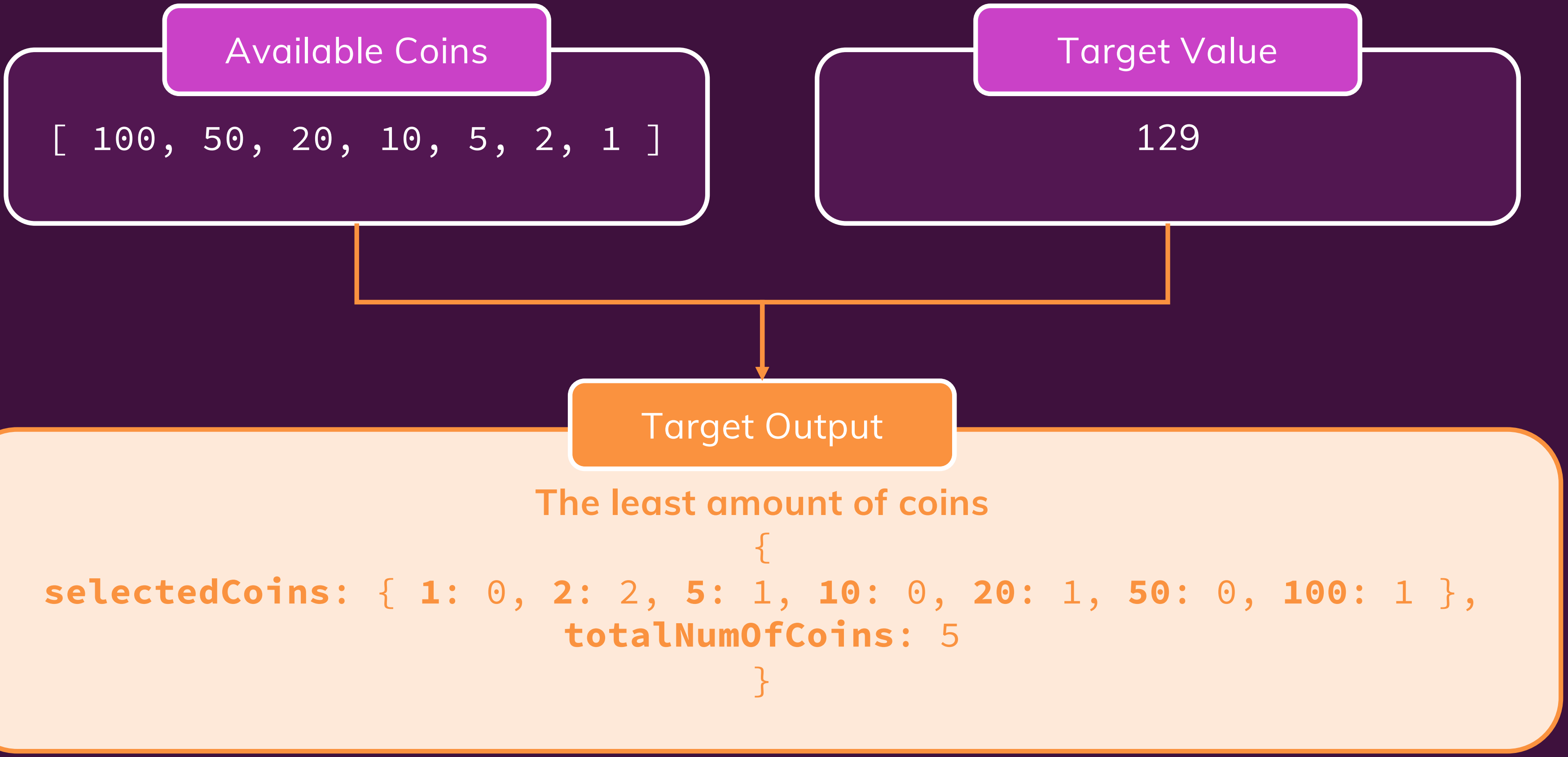
Verify Results

Try different inputs

Derive Time Complexity

Explore Alternative Approaches

The Change Making Problem



Available Coins

[100, 50, 20, 10, 5, 2, 1]

Target Value

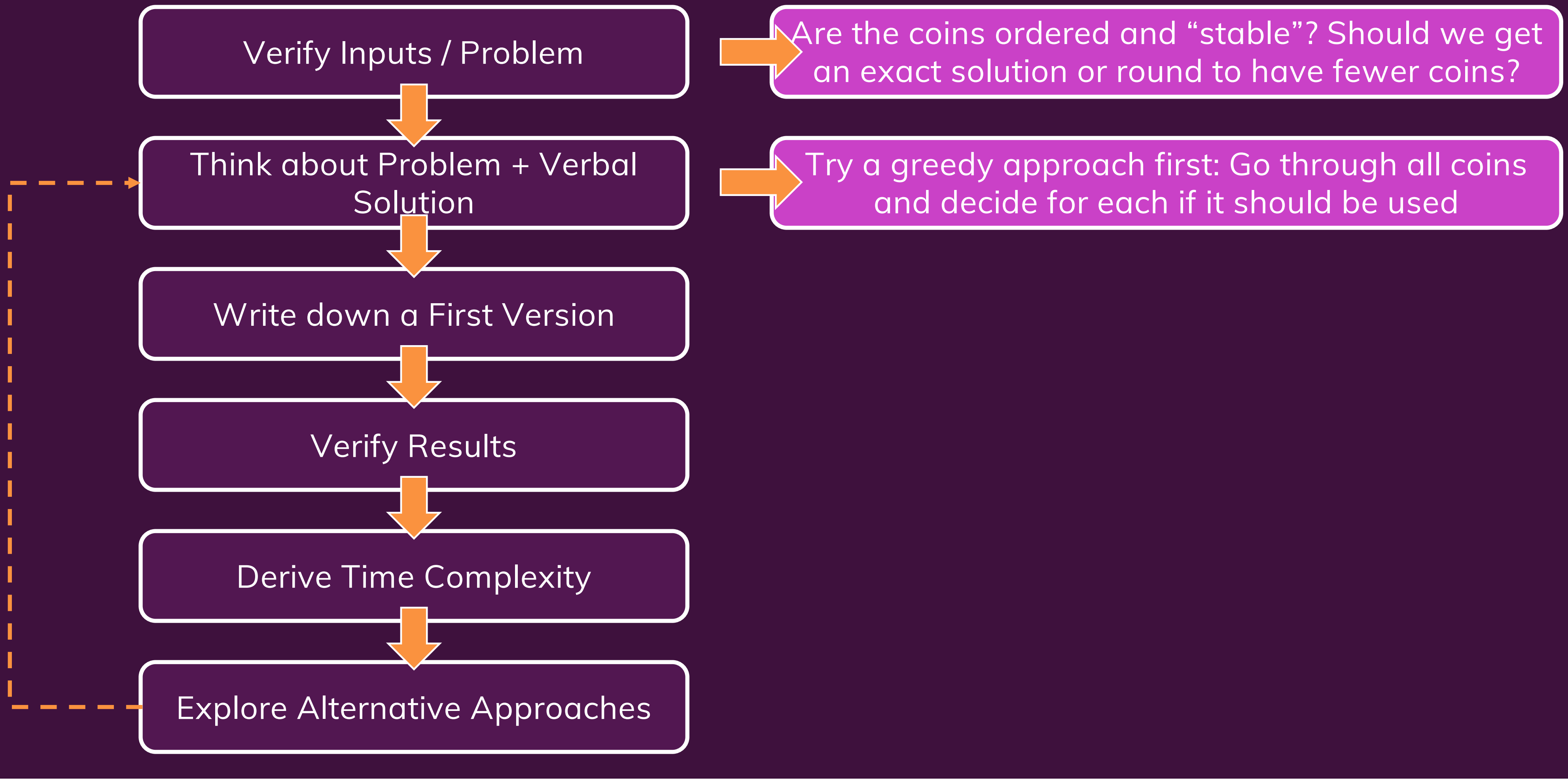
129

Target Output

The least amount of coins

```
{
selectedCoins: { 1: 0, 2: 2, 5: 1, 10: 0, 20: 1, 50: 0, 100: 1 },
totalNumOfCoins: 5
}
```

Change Making Problem: Our Plan



Verify Inputs / Problem

Are the coins ordered and “stable”? Should we get an exact solution or round to have fewer coins?

Think about Problem + Verbal Solution

Try a greedy approach first: Go through all coins and decide for each if it should be used

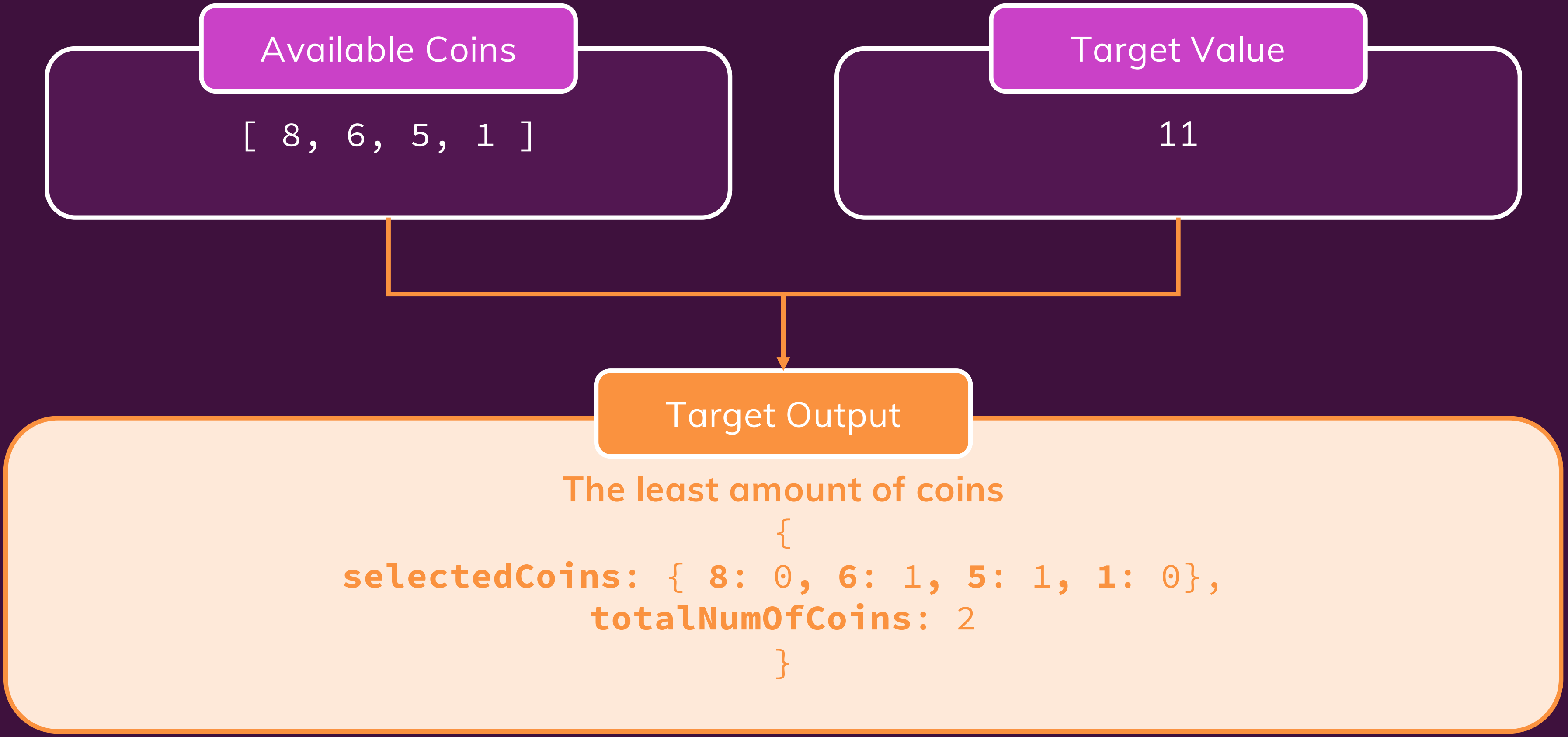
Write down a First Version

Verify Results

Derive Time Complexity

Explore Alternative Approaches

The More Difficult Change Making Problem



Available Coins

`[ 8, 6, 5, 1 ]`

Target Value

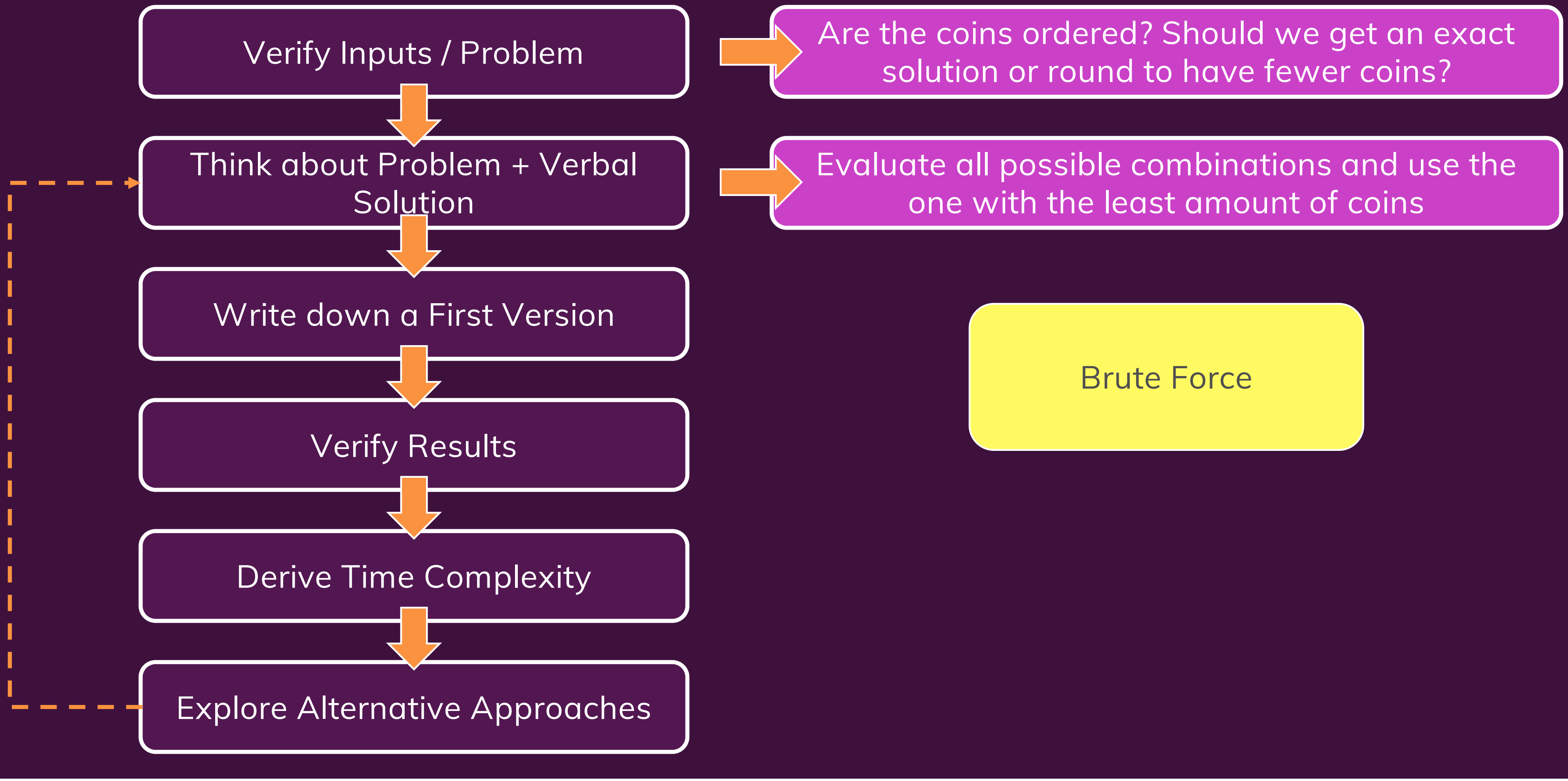
11

Target Output

The least amount of coins

```
{
selectedCoins: { 8: 0, 6: 1, 5: 1, 1: 0},
totalNumOfCoins: 2
}
```

Change Making Problem: Adjusted Plan



Verify Inputs / Problem

Are the coins ordered? Should we get an exact solution or round to have fewer coins?

Think about Problem + Verbal Solution

Evaluate all possible combinations and use the one with the least amount of coins

Write down a First Version

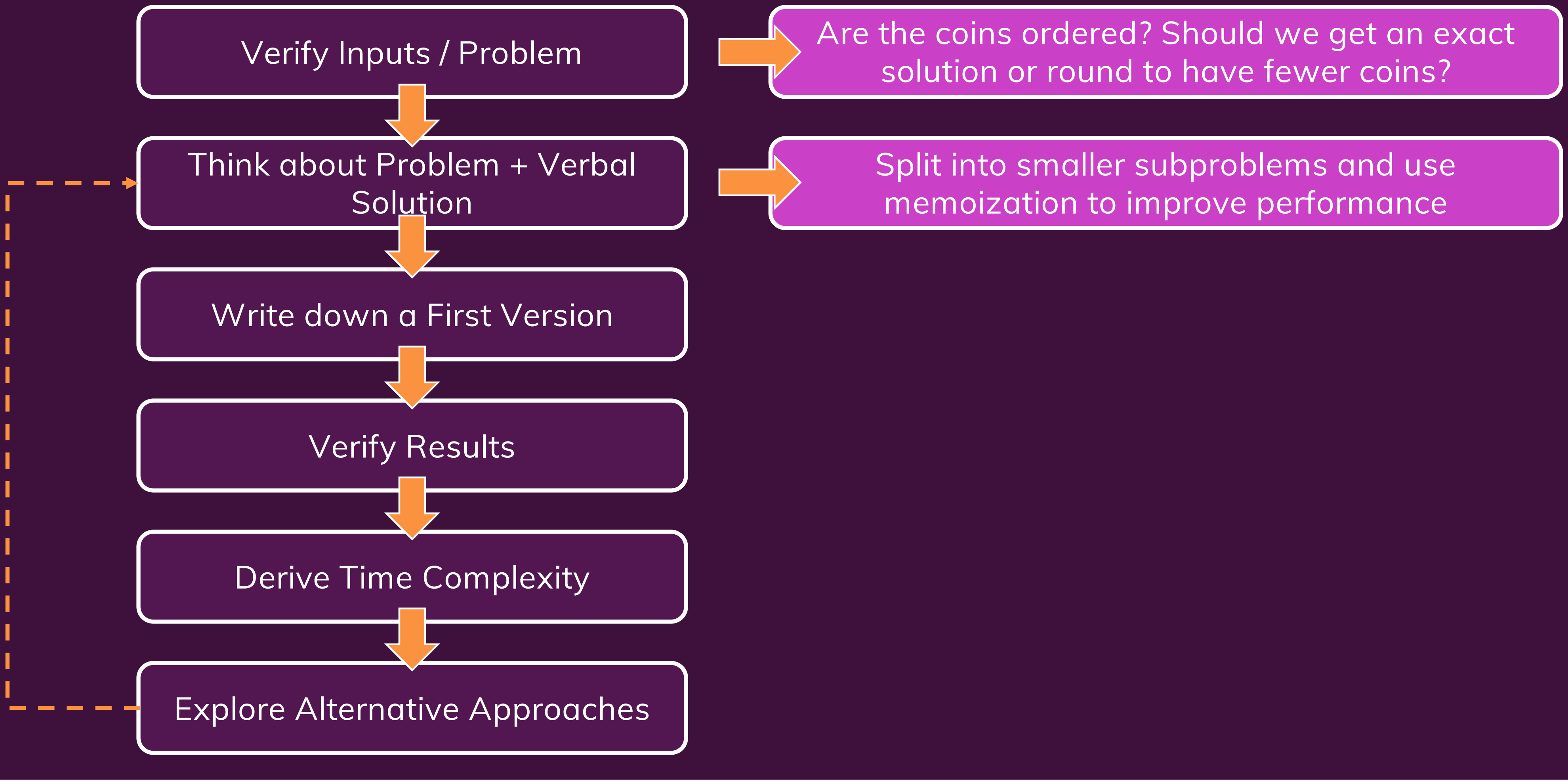
Brute Force

Verify Results

Derive Time Complexity

Explore Alternative Approaches

Change Making Problem: Final Plan



Verify Inputs / Problem

Are the coins ordered? Should we get an exact solution or round to have fewer coins?

Think about Problem + Verbal Solution

Split into smaller subproblems and use memoization to improve performance

Write down a First Version

Verify Results

Derive Time Complexity

Explore Alternative Approaches