Space Complexity: Takeaways

Syntax

- Converting a list into a set:

```
my_set = set(my_list)
```

- No preprocessing algorithm for the sums of pairs problem:

```
def find_sums(values, target_sums):
sums = {}
for target in target_sums:
    sums[target] = False
    for i in range(len(values)):
        for j in range(i, len(values)):
            if values[i] + values[j] == target:
                sums[target] = True
return sums
```

- Full preprocessing algorithm:

```
def find_sums_precompute(values, target_sums):
    possible_sums = set()
    for i in range(len(values)):
        for j in range(i, len(values)):
            possible_sums.add(values[i] + values[j])
    sums = {}
    for target in target_sums:
        sums[target] = target in possible_sums
    return sums
```

- Balanced algorithm:

```
def find_sums_balanced(values, target_sums):
    value_set = set(values)
    sums = {}
    for target in target_sums:
        for value1 in values:
            value2 = target - value1
            sums[target] = value2 in value_set
    return sums
```

Concepts

- There are three basic types of values in Python: numbers, character, and references to variables.
- The space complexity of a more complex type is the total number of the basic values it holds. As with time complexity, we don't care about the exact number, but the order of growth of this quantity as a function of the input size.

- Time and space complexity can often be traded for one another. By preprocessing some of the data, we can usually improve the time complexity by increasing the space complexity.

Resources

- Space complexity
- 2 sum problem