Why reinvent the wheel?
Reduce to another problem that you have already solved.

**2SUM PROBLEM**

- **Input:** Array $A = [A[1], ..., A[n]]$ of integers and a target $T$
- **Output:** true if there exist two values in $A$ (possibly the same value twice) whose sum equals $T$, else false

Additional definitions:
- A **yes-instance** is an input to a decision problem, for which the correct output is true
- A **no-instance** is an input to a decision problem, for which the correct output is false

Since the output is true/false, this is called a “decision problem”

**SIMPLE (BRUTE FORCE) SOLUTION**

```
2SUM BruteForce(A[1..n], T)
for i = 1 .. n
    for j = i .. n
            return true
return false
```

Runtime $O(n^2)$ by similar arguments to earlier...

Idea: let’s turn the innermost loop into something more efficient...

**AN IMPROVEMENT**

  we can rearrange to get $A[j] = T - A[i]$
- Instead of looping over $j$, 
  **search** the array for $T - A[i]$

How to do this efficiently?

**IMPROVED ALGORITHM**

```
2SUM Improved(A[1..n], T)
sort(A)
for i = 1 .. n
    j = binary search for $T - A[i]$
    in the subarray $A[i..n]$
    if search is successful return true
return false
```

VS. linear search, which takes $O(n)$ time

What is this algorithm’s time complexity?
**TIME COMPLEXITY**

- Loop: iterations \* work per iteration
  - \( \Theta(n) \) \* \( \Theta(\log n) \) = \( \Theta(n \log n) \)
  - Entire algorithm: \( \Theta(n \log n) + \Theta(n \log n) = \Theta(n \log n) \)

**3SUM PROBLEM**

- Input: Array \( A = [A[1], ..., A[n]] \) of integers and a target \( T \)
- Output: true if there exist three values in \( A \) (possibly taking the same value two or three times) whose sum equals \( T \), else false

This is quite similar to 2SUM... Can we reduce to 2SUM?

**3SUM TO 2SUM**

- How can we use Solve2SUM to solve 3SUM?
  - By changing the array \( A \) somehow?
  - By changing the target \( T \) somehow?

```
1. Reduce3SUMto2SUM(A[1..n], T)
2. for i = 1 to n
3. T2 = T - A[i]
4. if Solve2SUM(A, T2) return true
5. return false
```

**REDUCTIONS**

- Suppose we already have a solution to 2SUM called Solve2SUM
- Suppose we design an algorithm Reduce3SUMto2SUM that solves 3SUM, and this algorithm calls Solve2SUM as a subroutine

Solve2SUM is a black-box subroutine that we call an “oracle”

- Reduce3SUMto2SUM is called a reduction from 3SUM to 2SUM
- Could also process input / call Solve2SUM multiple times
- If 3SUM can be reduced to 2SUM, we denote this by 3SUM \( \leq \) 2SUM

Mnemonic: 2SUM goes into 3SUM as a subproblem

```
1. Reduce3SUMto2SUM(A[1..n], T)
2. for i = 1 to n
3. T2 = T - A[i]
4. if Solve2SUM(A, T2) return true
5. return false
```

**PREPROCESSING**

- The sort is an example of pre-processing
- If modifies the input to permit a more efficient algorithm (binary search as opposed to linear search)
- Note that a pre-processing step is only done once
**REDUCTION CORRECTNESS**

- **Must prove:** \( 3\text{SUM}(A, T) \Leftrightarrow \exists i : 2\text{SUM}(A, T - A(i)) \)
- **In other words,**
- Let \( A, T \) be any input to 3SUM
- There exist \( A(i), A(j), A(k) \) that sum to \( T \) if and only if
- There exists some \( A(m) \) such that \( 2\text{SUM}(A, T - A(m)) \) returns true

**REDUCTION CORRECTNESS 2**

- **WTP:** \( \exists A(i), A(j), A(k) \) that sum to \( T \) if and only if
- \( \exists A(m) \) such that \( 2\text{SUM}(A, T - A(m)) \) returns true

**REDUCTION RUNTIME**

- \( \Theta(n) \) loop iterations
- Each iteration does \( \Theta(1) + \text{Runtime}(2\text{SUM}) \) work
- Runtime depends on implementation of 2SUM!
- Brute force: \( \Theta(n^3) \)
- Binary search: \( \Theta(n) \times \Theta(n\log n) = \Theta(n^2\log n) \)

**FURTHER IMPROVEMENT**

- Recall our fastest Solve2SUM took \( O(n\log^2 n) \) time for sorting and \( O(n\log n) \) total time for searching
- Can actually improve 2SUM to \( O(n) \) searching time with a greedy approach
- Does not change complexity of 3SUM, but we will see we can still speed up our 3SUM reduction...

**FAST 2SUM**

\[
T = 23 \quad \begin{array}{ccccc}
2 & 3 & 5 & 11 & 12 & 20 & 22 \\
i & i & j & j & \\
\end{array}
\]

- **Correctness**
- Invariant: if there exists a solution \( i' < j' \) then \( i' \geq i \) and \( j' \leq j \)
- Exercise: fill in the proof details

**FAST 3SUM TO 2SUM REDUCTION**

- Although fast 2SUM is still \( \Theta(n\log n) \), we can sort only once in our reduction
- Since 2SUM is given a pre-sorted array, it takes \( \Theta(n) \) time!
- We get runtime \( \Theta(n\log n) + \Theta(n)\Theta(n) = \Theta(n^2) \)
**IS THERE A FASTER 3SUM ALGORITHM?**

- For many years, people thought this was likely optimal.
- However, faster algorithms appeared in 2014, 2017.
- Best known solution is $O(n^2 \log \log n)/(\log n)$.
- This is a polylog factor faster than $O(n^2)$.
- ...we suspect there is no solution faster than $O(n^{2.373})$.

**A TRIVIAL REDUCTION**

- Suppose we want to multiply two integers, $x$ and $y$.
- Consider the algebraic identity: $xy = \frac{(x+y)^2 - (x-y)^2}{4}$.
- This allows us to show that multiplication is squaring.
- **Oracle:** ComputeSquare
  - Oracle "gives" you a solution to the subproblem...
  - If you solve ComputeSquare, you've solved Multiply.

**A MEDIUM REDUCTION**

- **3SUMZero** problem:
- Suppose we have solved 3SUMZero and want to solve 3SUM.
- It is straightforward to modify any algorithm for 3SUMZero so it solves 3SUM.
- Another approach is to find a reduction $3SUM \leq 3SUMZero$. This would allow code re-use.

**3SUM $\leq$ 3SUMZERO**

- This suggests the following approach.

```
1 Reduce 3SUM to 3SUMZero (A, T)
2 for i = 1, ..., n
3 B[i] = 3A[i] - T
4 return Solve3SUMZero (B)
```

Given an oracle that solves 3SUMZero, let's solve 3SUM.

Can we find $\sum x_i = T$ by finding $\sum y_i = 0$?

**A HARD REDUCTION**

- **3array3SUMZero** problem:
  - Input: three arrays of $n$ integers: $A$, $B$, and $C$.
  - Output: true if there exist $A[i], B[j], C[k]$, whose sum equals 0, else false.
- Let's try to reduce this to 3SUMZero.

```
1 TRY_reduce_3array3SUMZero to 3SUMZero (A, B, C)
2 A' = concatenation of A, B, C
3 return Solve3SUMZero (A')
```

Given an oracle that solves 3SUMZero, might choose $\geq 2$ elements from the same array.
THE REDUCTION

\[ A' = \text{concatenation of } D, E, F \]

CORRECTNESS OF THE REDUCTION (1/3)

\[ \text{Reduce 3array3SUMZero to 3SUMZero}(A, B, C) \]

\[ \text{for } i = 1 \ldots n \]
\[ D(i) = 10A(i) + 1 \]
\[ E(i) = 10B(i) + 2 \]
\[ F(i) = 10C(i) - 3 \]

\[ A' = \text{concatenation of } D, E, F \]

CORRECTNESS OF THE REDUCTION (2/3)

- To show that this reduction is correct, we prove:
- \text{true is the correct output for } Solve3SUMZero(A') \text{ if and only if} \text{true is the correct output for } Solve3array(A, B, C)
- Case 1: Assume true is the correct output for \text{Solve3array}(A, B, C)
  - Want to show true is the correct output for \text{Solve3SUMZero}(A')
  - By our assumption, there exist \( A[i] + B[j] + C[k] = 0 \)
  - \text{So } 10A[i] + 1 + 10B[j] + 2 + 10C[k] - 3 = 0

\[ D(i) \]
\[ E(i) \]
\[ F(i) \]

- So \text{true is the correct output for } Solve3SUMZero(A')

CORRECTNESS OF THE REDUCTION (3/3)

- Case 2: Assume \text{true is the correct output for } Solve3SUMZero(A')
- Want to show \text{true is the correct output for } Solve3array(A, B, C)
  - Claim: this sum consists of one element from each of \( A, B \) and \( C \)

\[ \text{By cases... Example case:} \]
\[ \text{Suppose, for contradiction, that } A[i], A[j], A[k] \text{ are all in } B \]

Then the sum \( A[i] + A[j] + A[k] = 10B[\_] + 2 + 10B[\_] + 2 + 10B[\_] + 2, \text{ which is not zero} \text{ Contradiction!} \]

Consider the sum modulo 10... only way to get 0 is to pick one element from each of \( A, B, C \)

So, there is a sum 0, with one from each of \( A, B, C \)
So, true is the correct output for \text{Solve3array}(A, B, C)

So, \text{3array3SUMZero} \leq \text{3SUMZero}
MANY-ONE REDUCTIONS

- The previous three reductions had a very special structure
- We transformed (reduced) an instance of the first problem to an instance of the second problem
- We called the oracle once, on the transformed instance
- Reductions of this form, in the context of decision problems, are called **many-one reductions**
  - (also known as polynomial transformations or Karp reductions)
- We will many examples of these in the section on **intractability**