

# Week 3 Exercises

Christopher Rodgers

March 30, 2024

Please complete all exercises below. You may use any library that we have covered in class UP TO THIS POINT.

1) Two Sum - Write a function named `two_sum()`

Given a vector of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Example 1:

Input: `nums = [2,7,11,15]`, `target = 9` Output: `[0,1]` Explanation: Because `nums[0] + nums[1] == 9`, we return `[0, 1]`. Example 2:

Input: `nums = [3,2,4]`, `target = 6` Output: `[1,2]` Example 3:

Input: `nums = [3,3]`, `target = 6` Output: `[0,1]`

Constraints:

`2 <= nums.length <= 104` `-109 <= nums[i] <= 109` `-109 <= target <= 109` Only one valid answer exists.

*Note: For the first problem I want you to use a brute force approach (loop inside a loop)*

*The brute force approach is simple. Loop through each element  $x$  and find if there is another value that equals to  $target - x$*

*Use the function `seq_along` to iterate*

```
two_sum <- function(nums_vector,target){
  #your code here
  result <- list()
  n <- length(nums_vector)
  for (i in seq_along(nums_vector)){
    for (j in seq_along(nums_vector)) {
      if(i!= j && nums_vector[i] + nums_vector[j] == target) {
        result[[length(result) + 1]] <-c(i,j)
        print(c(i,j))
      }
    }
  }
}
```

```
# Test code
nums_vector <- c(5,7,12,34,6,10,8,9)
target <- 13
```

```
two_sum(nums_vector,target)
```

```
## [1] 1 7
## [1] 2 5
## [1] 5 2
## [1] 7 1
```

```
#expected answers
#[1] 1 7
#[1] 2 5
#[1] 5 2
```

- 2) Now write the same function using hash tables. Loop the array once to make a hash map of the value to its index. Then loop again to find if the value of target-current value is in the map.

*The keys of your hash table should be each of the numbers in the `nums_vector` minus the target.*

*A simple implementation uses two iterations. In the first iteration, we add each element's value as a key and its index as a value to the hash table. Then, in the second iteration, we check if each element's complement ( $\text{target} - \text{nums\_vector}[i]$ ) exists in the hash table. If it does exist, we return current element's index and its complement's index. Beware that the complement must not be `nums_vector[i]` itself!*

```
library(hash)
```

```
## Warning: package 'hash' was built under R version 4.3.3
```

```
## hash-2.2.6.3 provided by Decision Patterns
```

```
h <- hash()
two_sum <- function(nums_vector,target){
  #your code here
  for (i in seq_along(nums_vector)){
    h[nums_vector[i]] <- i
  }
  for (i in seq_along(nums_vector)){
    complement <- target - nums_vector[i]
    if(complement %in% names.hash(h) && complement !=i)
      print(c(nums_vector[i], complement))
  }
}

# Test code
nums_vector <- c(5,7,12,34,6,10,8,9)
target <- 15

two_sum(nums_vector,target)
```

```
## [1] 5 10
## [1] 7 8
## [1] 6 9
## [1] 10 5
## [1] 9 6
```

```
#expected answers
#[1] 10 5
#[1] 8 7
#[1] 9 6
#[1] 5 10
#[1] 7 8
#[1] 6 9
```