



[1] Strings, Arrays, Pointers, and Hashing





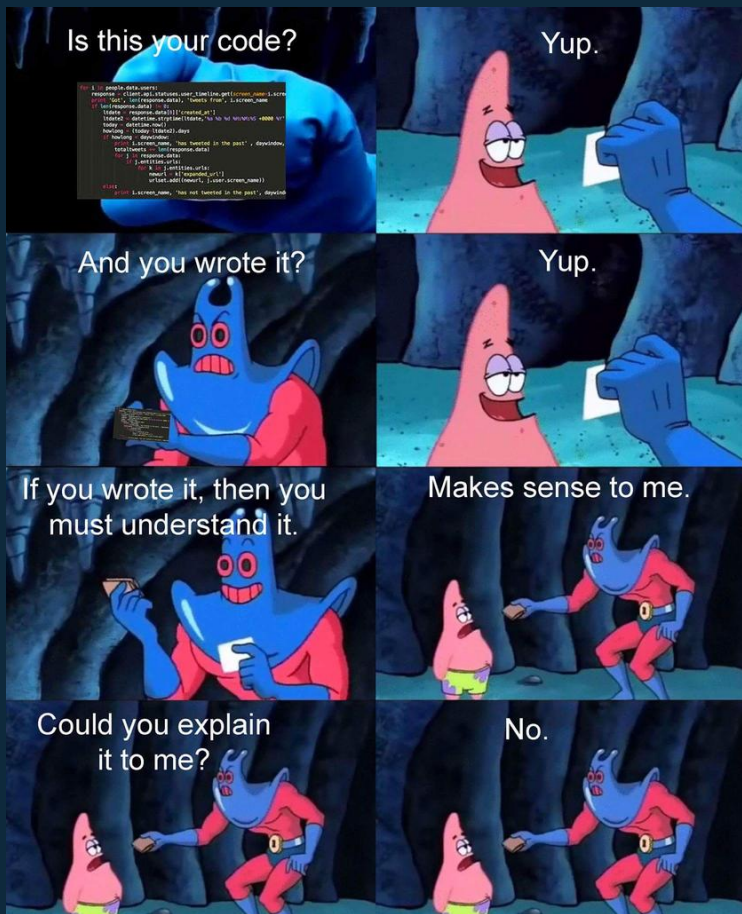
Quick Tips

Try solving these problems on your own (without the thought of trying to program them). Then, once you find a pattern, code it

Thoroughly test your algorithm and handle edge cases

Best conceivable runtime helpful for a goal to reach

Think about how fast you should be able to solve the problem, and shoot for that as a goal





Strings (Object)

Immutable, so use
StringBuilder.

Bad ----->

Use .equals for
comparison

```
String joinWords(String[] words) {  
    String sentence = "";  
    for (String w : words) {  
        sentence = sentence + w;  
    }  
    return sentence;  
}
```



Palindromes!

Same phrase that reads backwards and forwards.
Racecar, madam, nurses run, taco cat, naan.

How would you code this?



Possible Considerations

Recursive? Check first and last characters recursively call string without first and last. $O(n)$ time, $O(n^2)$ space.

Create a reversed string and return `isEqual`s of the two strings? $O(n)$ time to create reversed string. $O(n)$ space.

Start from the center/outside of the string, expand outward/inward comparing adjacent characters for equality. Also $O(n)$ time, but better space than the last solution as you don't have two strings.



Trusty Solution

Show code



Longest Palindrome in a String

“bananas” has “anana”

“abracadabra”, there is “aca” and “ada”

“cat” would not have any

Try “afdswoaaotwadddtfkfdatadf”



afdswoaaotwaddtfdadaf

How did you do it?

Can you code this process?

Difference between odd and evenly sized palindromes.

Let's traverse the string and "expand" outward from each character to find if this character is the center of a palindrome.

What about words with an even number of characters? "naanbbnaan" if you expand at the first or second b individually, you miss the palindrome... Just start your expansion without considering the second b.



Time to code!!



Space and Time Complexity

Worst Case:

Aaaaaaaaaa

Best Case:

abdcefg hij

Gives $O(n^2)$. Why?

Space?

$O(n)$ with input, $O(1)$ extra space needed aka Auxiliary space.



Arrays and ArrayLists (Java)

Arrays:

- Holds all objects and primitives
- Constant Size

.length


[index]

ArrayLists:

- No primitives
- Variable length with amortized insertion $O(1)$

.size()

.get() and add()



Merge Two Sorted Array

Important helper method in Merge Sort

{ 1, 3, 4, 5} and {2, 4, 6, 8}

Output : {1, 2, 3, 4, 5, 6, 7, 8}

{ 5, 8, 9} and {4, 7, 8}

Output : {4, 5, 7, 8, 8, 9}



Try keeping two pointers throughout the arrays as you try to traverse through the inputs

Time to code!

Hashing and HashMaps

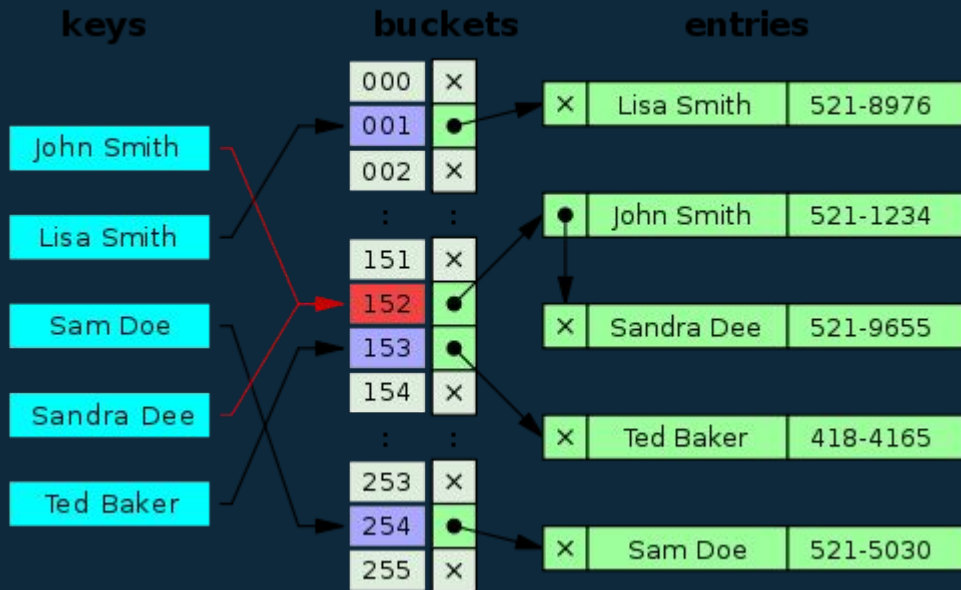
HashMaps: Unique keys
lead to values


HashFunction() returns
int (HashValue).

HashValues lead to
buckets. Linked list of
key/value pairs.

$O(1)$ get() and put() in
best case.

$O(n)$ worst case (when
everything is in the
same bucket)





Why HashMaps are awesome!

$O(1)$ lookup time! Turn $O(n)$ methods in arrays to constant time in HashMaps!!

Ex:

```
public int numberOf(int n, int[] input) {  
    int count = 0;  
    for(int num: input) {  
        if(num == n) {  
            count++;  
        }  
    }  
    return count;  
}
```

Could replace with Map where key (int in the array) leads to number of times the key has appeared.

Maps are useful to increase your speed, but keep in mind the extra space requirement



Isomorphic Strings

"abca" and "zbxz" are isomorphic - map 'a' to 'z', 'b' to 'b' and 'c' to 'x'

"foo" and "bar", return false.

"paper" and "title", return true.



Time to Code!



2sum

Given an array of integers, and a target number, find the number of pairs of integers in the array whose sum is equal to the target.

$\{1, 5, 7, -1\}$, target = 6

Return 2

$\{10, 12, 10, 15, -1, 7, 6, 5, 4, 2, 1, 1, 1\}$, target = 11

Return 9

Interesting to think about: What about 2sum on sorted array? How does time and space complexity compare?



Try finding the complement of a given number where the complement is the target minus the current number.

{1, 5, 7, -1}, target = 6. Start at 1, loop through the rest for a 5. This is $O(n^2)$.

Do you think you could do better than that? What is the bottleneck in this case? Finding the other number.

Hashmaps are good for increasing the speed. Try using them.



Time to code!!



3sum

Given an array, return true if 3 numbers sum to 0, else return false.

$S = [-1, 0, 1, 2, -1, -4],$
 $[-1, 0, 1],$
 $[-1, -1, 2]$

What is the simplest way to solve this? Brute force? What about reframing it to a 2sum problem?

How fast do you think you should be able to do this? $O(n)$, $O(n^2)$, $n \log n$?

What are the bottlenecks, can you reframe this to a problem you already know?

Does sorting the array help?



Try reframing into 2sum problem:

Iterate through the array, find a pair of numbers that equals the negative of the current number using the rest of the array.

Ex. [1, 2, -3, 5, -2, 9], iterate at 1, use array (excluding the 1) to get -1.

Repeat for all numbers. Space and time?

Sorting method:

Very similar process to previous above solution, but slightly more efficient yet still $O(n^2)$. Why is it more efficient?

You don't have to iterate through the entire array as you can easily bound searches on sorted arrays.



Time to code!!



Time complexity?

$O(n^2)$

Space complexity?

Depends on sorting, but most likely $O(1)$.

$O(n)$ for non sorting



Shortest Substring with characters of Original String

Given a set T of characters and a string S , find the substring in S which will contain all the characters

2 input strings, return string

S - teststring

T - ig

Returns ing

S - ADOBECODEBANC

T - ABC

returns BANC



Iterate through the string and create a “window” with all the characters required.

Remove unnecessary characters from the beginning of the window until you no longer have all the necessary characters.

Extend the end until you have all the characters again.



Time to code!!



Next Largest Permutation

Given a number n , find the **smallest number** that has **same set of digits** as n and is **greater than n** .

If x is the greatest possible number with its set of digits, then print “not possible”.

Input: $n = "1234"$ -> $"1243"$

Input: $n = "4321"$ -> Not Possible

Input: $n = "218765"$ -> $"251678"$



Notice any patterns?

If the numbers are in descending order, you cannot find a bigger number.
Does this make sense?

What about ascending order?

Can you find something in the middle? Here's another example, try it out.

Input: n = "534976" -> "536479"



Coding Time