__/__/____

Strings

① Reverse a string

\# - keep two pointers at start & end of the array or string
- keep swapping the two and keep shrinking it till they ~~both~~ meet in the midpoint.

\# - using STL reverse

② Palindrome or not

\# - ~~compare~~ keep two pointers start & end
- compare the two if its not equal then return false
- or else if its equal & ~~soon~~ flow comes to mid point then return true.

③ largest odd substring

\# - iterate from back to find the last odd number & break
- iterate from first to find the ~~one~~ first non zero number
- then return the substring from first non zero to last odd.

__/__/____

(4) Longest common substring
- in multiple words find the longest common string
- first we sort the given arrays of string or the given words
- now we take first word & the last word & find the common in those two
Cuz all other words in b/w first & last will have the same prefixes like in a dictionary)
- just start comparing each letter of the two words & when its not equal break
- but until then keep adding the common letters to the string.
- & return the string.

(5) Isomorphic string
- basically in given 2 strings of same length S1 should be able to replace S2
ex: $e^{S1}gg \rightarrow add^{S2}$ e → a g → d
e hold a g holds d

egg → add a e → a g → d g → a
↓
cant because g already mapped to d

__/__/__

- So create two maps ~~for~~ for S1 & S2
- in map S1 S2 chars should be having same index if it mismatches then it's not isomorphic

ex: egg & add egg & ada

g-1	d-1
e-0	a-0

g-1
e-0

a
d-1
a-0

X g=1, a=0

⑥ Check if goal string is a ~~substring~~ rotation of given String

String: : npos → acts like .end()

- basically just add the given string twice
- now find all possible rotations if any of the rotation is equal to goal return true
- else false.

⑦ Anagram

Ex: cat can be made from act

#BF just sort both the string if both are equal then return true or else false.

OP - just store all elements/chars of first string in map with its occurences
- now iterate through the 2nd string decrementing the occurences of each letter basically cancels out to 0 but if it doesn't cancel out like if char doesn't exist or if chars value is already 0 then return false
- if not return true.

__/__/____

* ⑧ Sort characters by frequency

- basically all you have to do is store all characters in a map with its frequencies
- then find the higest frequency in the map
- now iterate from the higest frequency to lowest frequency
  - and an inner loop iterating through the map to find an element matching current frequency
- when you find the matching frequency you push that frequency element into vector as many times as its occurrence (freq.
- then just return the vector.

helper function