Sentence Reverse

You are given an array of characters **arr**, which consists of sequences of characters (words) separated by space characters.
How can you most efficiently reverse the order of words in the sentence?
Explain and code your solution and analyze the runtime and space complexity.

For example:
['p', 'e', 'r', 'f', 'e', 'c', 't', ' ', 'm', 'a', 'k', 'e', 's', ' ', 'p', 'r', 'a', 'c', 't', 'i', 'c', 'e']

would turn into:
['p', 'r', 'a', 'c', 't', 'i', 'c', 'e', ' ', 'm', 'a', 'k', 'e', 's', ' ', 'p', 'e', 'r', 'f', 'e', 'c', 't']

Hints & Tips

- Even if your language of choice enables that, using standard functions to join the characters into a string, split it and reverse the split words is *not* efficient. When you do that the compiler/interpreter is using **O(n)** space to create the string, **O(n)** space to hold the split words and additional **O(n)** space to hold the re-joined reversed string. Also, the idea of the question is to roll up your sleeves and do it yourself. Whenever a function like that is used ask your peer to explain the time and space complexity it takes and consider this on their complexity analysis.
- Watch for edge cases handling on your peer's solution: empty array, array with nothing but spaces, array with one word only, multiple spaces between words.
- To get maximum rating for problem solving, your peer must achieve linear time complexity and constant space complexity with no hints.
- If you have time left, ask your peer how could the mirrorReverse function be implemented with one index only. It's done with left to right linear iteration and swapping `arr[i]` and `arr[n-1-i]` as long as `i` < `n-1-i`.

Solution

A possible solutions is doing a linear iteration on **arr** while pushing a copy of the word to a stack and then pulling them in reverse order while copying the words back into arr. Another approach is initializing a new array at the same length, iterating **arr** from right to left and copying any sequence of characters to the new array from left to right. Both take linear **O(n)** runtime and at least **O(n)** space complexity.

A more elegant and efficient approach is to reverse all characters in **arr** and then reverse the characters on each words separately. The first reverse will give us he words on the reverse order we need, but will also reverse the characters of each word. The second reverse, done on each word separately, fix that.
Reversing items in an array is done by a 'mirror' function, that swaps the items of every 2 indices with the same distance from the middle.

```
function reverseWords(arr):
   # reverse all characters:
   n = length(arr)
   mirrorReverse(arr, 0, n-1)

   # reverse each word:
   wordStart = null
   for i from 0 to n-1:
      if (arr[i] == " "):
         if (wordStart != null):
            mirrorReverse(arr, wordStart, i-1)
            wordStart = null
      else if (i == n-1):
         if (wordStart != null):
            mirrorReverse(arr, wordStart, i)
      else:
         if (wordStart == null):
            wordStart = i

# helper function - reverses the order of items in arr
# please note that this is language dependent:
# if are arrays sent by value, reversing should be done in place
function mirrorReverse(arr, start, end):
   tmp = null
   while (start < end):
      tmp = arr[start]
      arr[start] = arr[end]
      arr[end] = tmp
      start++
      end--
```

Runtime Complexity: Iterating over the array twice with constant number of actions for each item, is linear **O(n)**.

Space Complexity: using iteration indices and one temp variable takes constant **O(1)** memory.