Week 13 — Space Constraints

Anh Huynh

1.

```
function wordBuilder(array) {
        let collection = [];
        for(let i = 0; i < array.length; i++) {
                for(let j = 0; j < array.length; j++) {
                        if (i !== j) {
                                collection.push(array[i] + array[j]);
                        }
                }
        }
        return collection;
}
```

The wordBuilder function uses nested loops (one loop inside another), so it combines every element with every other element. This produces about n x n strings so the space complexity is $O(n^2)$

2.

```
function reverse(array) {
        let newArray = [];
        for (let i = array.length - 1; i >= 0; i--) {
                newArray.push(array[i]);
        }
        return newArray;
}
```

This reverse array function is O(n) because the function creates a new array that stores all the n elements of the original array. It loops through the array once and each element gets copied into newArray. This means that the amount of extra space grows linearly with the size of the input.

3.Create a new function to reverse an array that takes up just O(1) extra space.

```
#include <iostream>
#include <vector>
using namespace std;

void reverseArray(vector<int>& arr){
    int left = 0;   // left ptr to first element
    int right = arr.size() - 1; // right ptr to last element

```

```
    while (left < right){   // while left is less than right
        int temp = arr[left];   // set first element to temp var
        arr[left] = arr[right]; // swap left and right elements
        arr[right] = temp;  // set right to temp var value

        // move window inward from both sides
        left++; // increment left side
        right--;    // decrement right side
    }
}

void printArr(vector<int>& arr){
    for (int x : arr){  // print every value in arr
        cout << x << " ";   // print format
    }
    cout << endl;   // newline
}

int main(){
    vector<int> list = {1, 2, 3, 4, 5, 6, 7, 8};    // create an arr

    printArr(list); // print pre-reversal

    reverseArray(list); // reverse arr

    printArr(list); // print post-reversal
}
```

Output:
1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1

4.

```
function doubleArray1(array) {
    let newArray = [];

    for(let i = 0; i < array.length; i++) {
        newArray.push(array[i] * 2);
    }
    return newArray;
}


function doubleArray2(array) {
    for(let i = 0; i < array.length; i++) {
    array[i] *= 2;
  }
    return array;
```

```
}


function doubleArray3(array, index=0) {
    if (index >= array.length) { return; }
  array[index] *= 2;
  doubleArray3(array, index + 1);
    return array;
}
```

Version	Time complexity	Space complexity
Version #1	O(n) — loops through n items	O(n) — create new array; size n
Version #2	O(n) — loops through n items	O(1) — modifies original array so no new array created
Version #3	O(n) — recursively checks n items	O(n) — recursion so adds n stack frames

Video Link: https://youtu.be/zRE-wBw7oEM