# Hey There!

Welcome to Actimize X-Sight Cloud Software Engineer home assignment.

Given the following function:

*/\*\*  
 \* Get a random integer in range  
 \** ***@param min*** *- the min number in the range  
 \** ***@param max*** *- the max number in the range  
 \** ***@return*** *- a random integer number between the min number and max number inclusive.  
 \*/  
public static int* randomInRange(*int* min, *int* max) {  
 *return* ThreadLocalRandom.*current*().nextInt(min, max + 1);  
}

Please write a function that will get a range (int min, int max) and by using the provided randomInRange function print all the numbers between min and max (inclusive) in a random order.

**Some guidelines:**

1. You project should be a Maven, Java project.
2. After providing a solution, do you think about more options that can solve this problem in another way?

If you have additional option, please provide all options.

1. Please specify the time and space complexity of your solutions.

**Hello to you to !☺ these are my answers:**

1.At first, I programmed in a more naïve manner and used 2 different data structures – two-sided linked list and 2 arrays – (thought about time complexity and figured out later that the same can be done using 2 arrays.

Note that this can be done using only one array in -place mixing- using only the first array and each iteration instead of putting the shuffled number at the beginning of the array – he will be putted at the end of the array (as we at every iteration "cut" the end of the array. ) it will save space of a size of an array but overall time and space complexity wouldn’t change and it will remain O(n).

**I provided the two options in code.**

3.Time complexity and space-

For the previous version- I had an array that held pointers to the relevant nodes in the two- sided linked list – Each data structure takes O(n) Space – **Overall O(n) Space complexity.**

In the While loop:

For each iteration- the call for the -Random In Range function is O(1)

Each access to node in a specific index is O(1) also because each cell holds a pointer to a specific node (in at first I started programming in a more naïve

– I designed the nodes such that each iteration, instead of changing the data inside the nodes, I changed the pointers – for no reason… but if the node's data held something more complex, such as pointers to pointers or something less primitive perhaps it would have been efficient

Each access to the output array is O(1) also. Overall constant amount of O(1) operations = > O(1)

We have n-1 iterations therefore = > **O(n) run time.**

**Improved Solution:**

Time complexity and space-

I had an array that held all the numbers between the ranges given and an array for output– Each data structure takes O(n) Space – **Overall O(n) Space complexity.**

Similar to the previous solution each access to a cell in an array is O(1) Operation. There Is a constant amount of operations in each iteration=> there are O(n) operations and therefore **O(n) run time overall.**