**Snippet 1:**

def find\_max(numbers):  
    max\_num = numbers[0]  
    for num in numbers:    
        if num > max\_num:  
            max\_num = num  
    return max\_num

Explanation: max\_num = numbers[0] initialize the max\_num with first element of array. The loop then iterates through all elements of the array, so the inside loop is only O(1) per iteration. So, the overall complexity is O(n)

Time Complexity: O(n)

Space Complexity: single array -> O(1)

Improvement: no need further improvement

**Snippet in page 2**

**Snippet 2:**

void printPairs(int[] array) {  
    for (int i = 0; i < array.length; i++) {  
        for (int j = i + 1; j < array.length; j++) {  
            System.out.println(array[i] + ", " + array[j]);

}}}

Explanation: There are 2 loops, first loop run n times, and second loop run n times.

Time Complexity: O(n2)

Space Complexity: As each iteration would run at constant time and does not store anything, the space complexity is O(1)

Improvement: Will depend on the purpose of the snippet. If it is finding the max then it is already optimal. However if it is to print any pair, then you can use parallel () which take advantage of multi-core processor to improve performance.

So a sample code would be:

void printPairs(int[] array) {

IntStream.range(0, array.length)

.parallel() // Parallelizes outer loop

.forEach(i -> {

IntStream.range(i + 1, array.length)

.forEach(j -> System.out.println(array[i] + ", " + array[j]));

});