CS-102 Midterm 1,

Arseniy Dolgin ([aad548@nyu.edu](mailto:aad548@nyu.edu))

**Question 1:**

1, 1.1: See Ex1.java – the program contains a total of 5 methods:

The first 3 are the different methods to find the minimal difference between any two numbers in a given array, methods 4 and 5 (sort and quicksort) are the methods that I have implemented in methods 2 and 3 to sort the given array in 2 different ways:

1. Method 1 has time complexity of O(n^2) since it uses two nested for-loops to arrive at solution.
2. Method 2 also has time complexity of O(n^2) because despite the fact that it runs on a sorted array, the array is sorted with Method 4 (sort), which is of O(n^2) complexity.
3. Method 3 has time complexity of O(n log n) because it runs on a sorted array once, and the array is sorted with Method 5 (quickSort), which is of O(n log n) complexity.

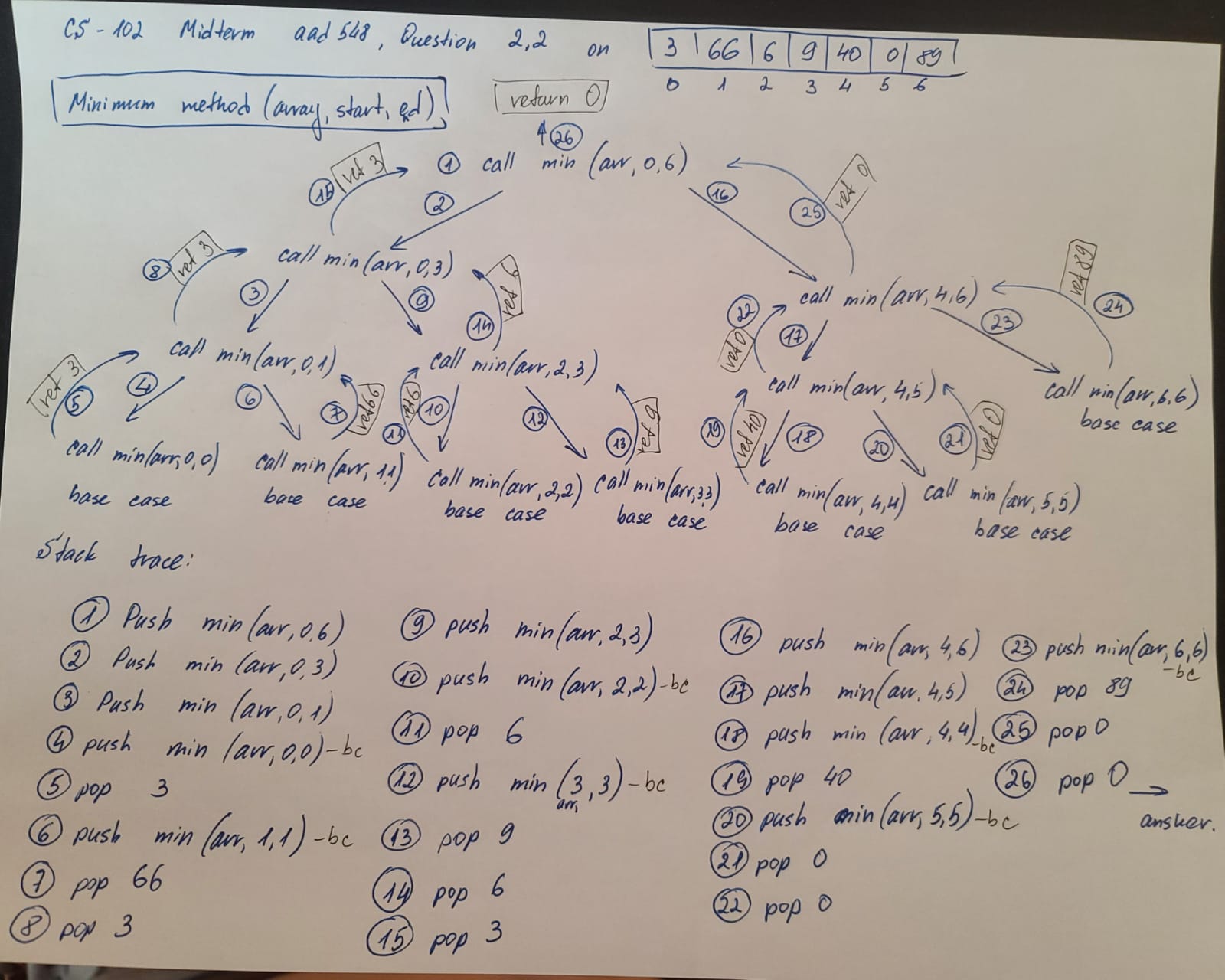
**Question 2:**

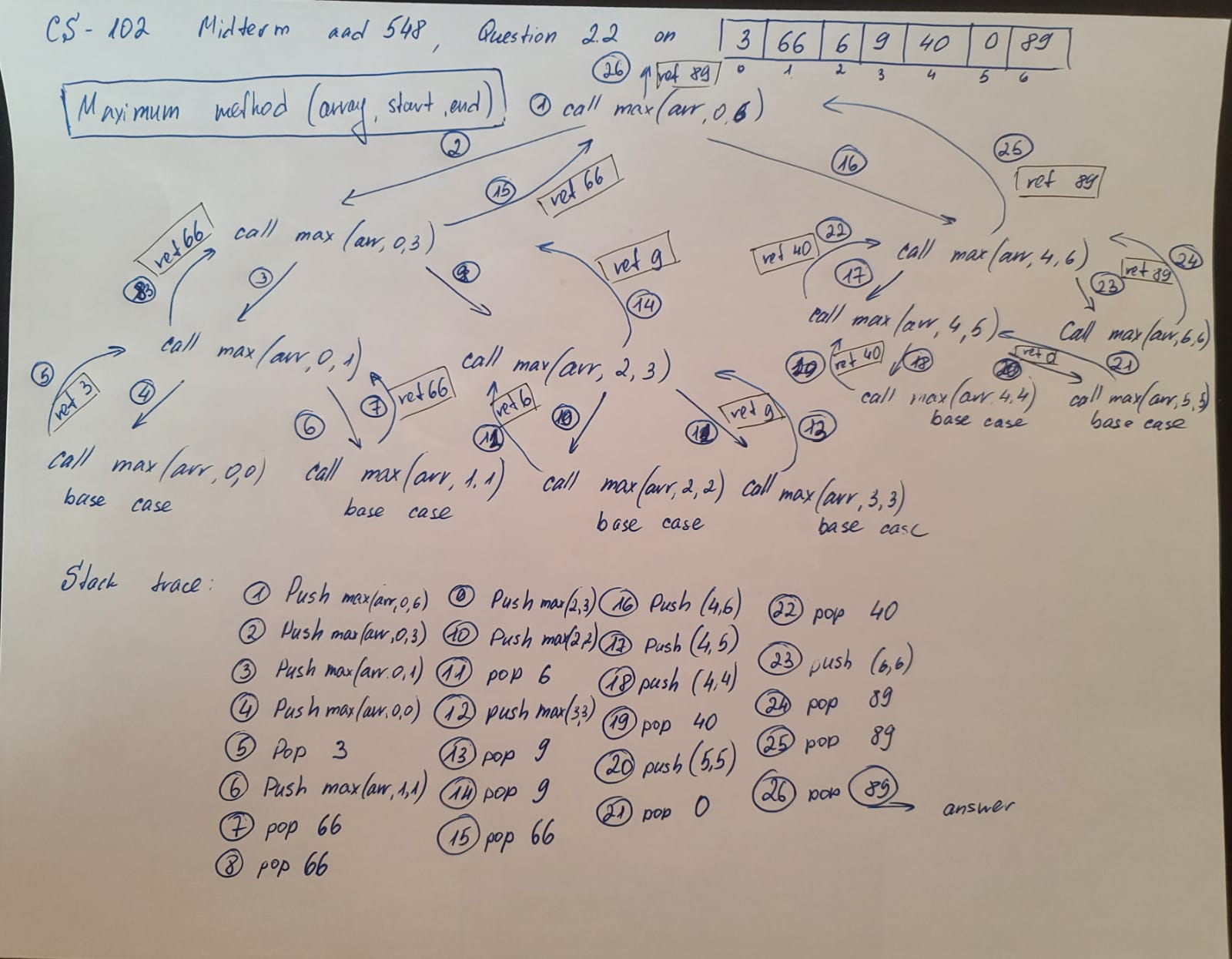
2.1: See Ex2.java – the program has 3 methods (maximum, minimum, and average), the max and min methods are binary recursive, whereas the average is simply recursive (I could not figure out how to make it binary).

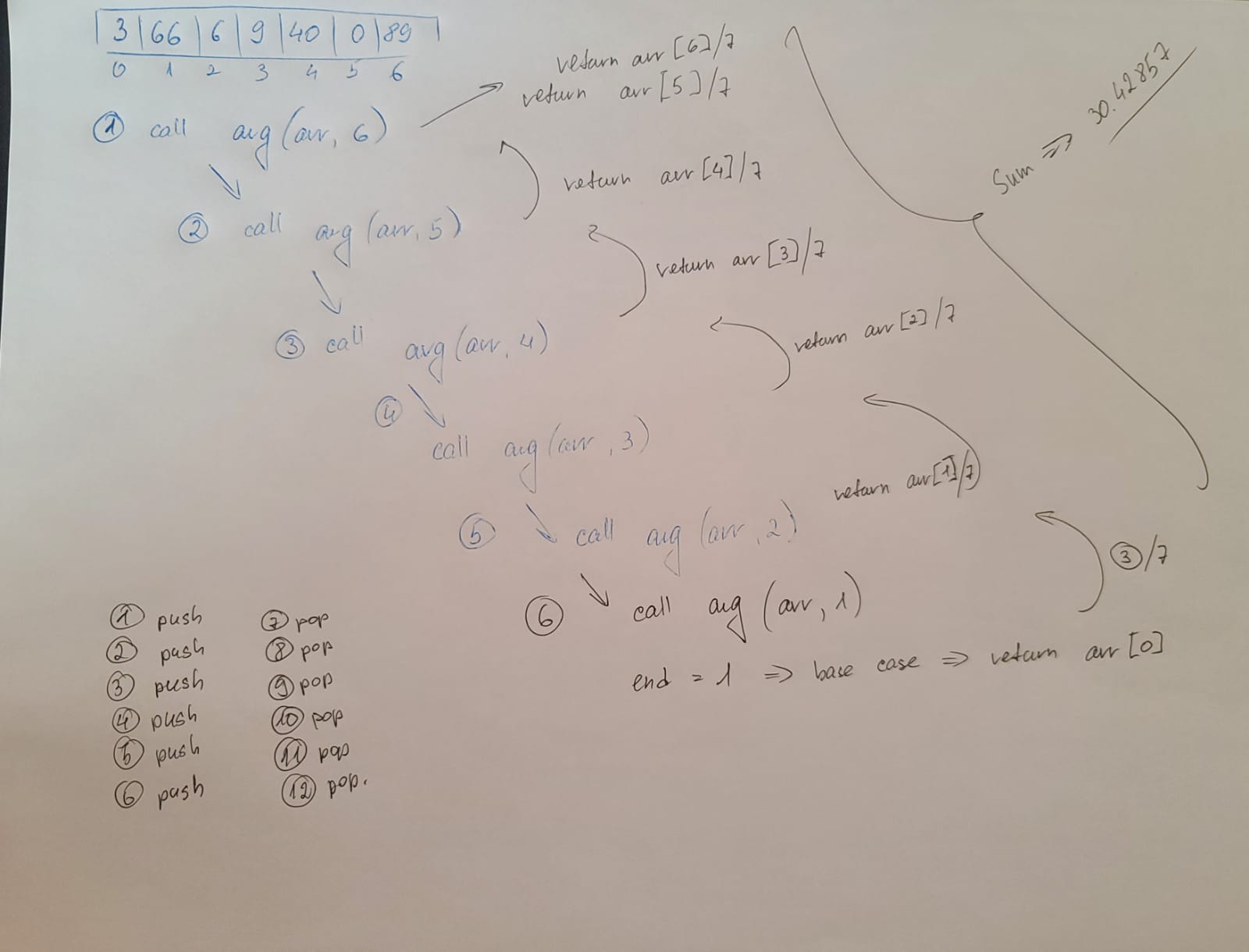
2.3:

1. Time complexity of method maximum is O(n).
2. Time complexity of method minimum is O(n).
3. Time complexity of method average is O(n).

2.2: See the drawings attached:







**Question 3:**

See Ex3.java.

Explanation of how my reverse method works:

1. First we initialize the stack with texts (Text 1 – Text 7).
2. We call the reverse() method, which:
   1. Checks whether the stack is not empty, and if it is not empty, it pops the String from the top and saves it into variable text.
   2. It keeps calling itself until the stack is empty.
   3. The final line in this method calls the method insert(), which checks whether the stack is empty, and if it is, it starts inserting the previously popped text.
   4. If the stack is not empty, the insert method pops the top of the stack and stores the String that it popped in variable a. The method then recursively calls itself to remove any Strings from the stack before it can start pushing them back in correct order.

The Second Method works by taking the original stack (if empty – base case), if not , pp the original stack and save in text, then push it to stack 2 and call yourself.

**Question 4:**

The running time of this program is (d) O(N\*N) because the first loop simply runs for 60 (or less) times, but it is a constant anyways. The number of times the second loop runs depends on N, and the number of times its nested for-loop runs also depends on N, so overall the time complexity of this program is O(N\*N).

**Question 5:**

The running time of this program is (e) O(1) because as long as N is not negative (actually, is not smaller than -3), j = i + N will always be greater than i/2, and the second for-loop will never run. If N < -3, then the second for loop starts, but the third for loop does not since k starts at 0, and runs until k<N. Since in the worst case the second for loop runs, but it does not contain any actions, the complexity of this algorithm is O(1).