1. push(8) → [8]
   push(2) → [8, 2]
   pop() → [8]
   push(pop()*2) → [16]
   push(10) → [16, 10]
   push(pop()/2) → [16, 5]
   Final stack: [16, 5]

2. push(4) → [4]
   push(pop()+4) → [8]
   push(8) → [8, 8]
   push(pop()/2) → [8, 4.0]
   pop() → [8]
   pop() → [] - empty
   Final stack: [] - empty

3. ```
   public class DequeSearch {
       public static int findElement(Deque<Integer> n, int x) {
           int left = 0;
           int right = n.size() - 1;
           // Convert deque to array for indexed access
           Integer[] arr = n.toArray(new Integer[0]);
           while (left <= right) {
               if (arr[left] == x) {
                   return left; // Found from the left
               }
               if (arr[right] == x) {
                   return right; // Found from the right
               }
               left++;
               right--;
           }
           return -1; // Element not found
       }
   ```

7. Question 4:
   - For time complexity, it is O(n) where n is the length of the string s. I say that because the for loop processes through each character in the string once.
   - For space complexity, it is O(n) where n is the string length. I say that because in the worst case, when all the characters are opening brackets, the stack will store all c characters.

Question 5:

- For time complexity, it is O(n+m) where n is the length of the input string and m is the length of the final decoded string. The outer loop is O(n) because it iterates over each
   ```

character in the input string s, which processes each character exactly once. Whereas the inner loop is O(m) because it creates new strings during the decoding process when characters inside brackets are repeated. Therefore, the total number of characters added to the final string is a function of the number of repetitions and the length of the strings being repeated, which is why it is O(m) and not O(n^2).

- For space complexity, it is also O(m+n) where O(n) is for the space used by the input string s and O(m) is for the space used to store the final decoded string. I say that because in the worst case, you might store up to n items in the 2 stacks, where n is the length of the input string. However, currentString stores the partially decoded string at each stage of the loop, where when the string is fully decoded, its size will be m, the length of the decoded output. Therefore the space taken by currentString and the result after it is decoded makes the time complexity O(m). So the overall space complexity is O(n+m) where n is the length of the input string and m is the length of the final decoded string.

Question 6:

- For time complexity, the loop iterates through each character of the string exactly once, and in the worst case, performs popping and other stack operators at most once for each character, which takes O(n) in the worst case, where n is the length of the input string s.
- For space complexity, every character in the string could be an operator or a parentheses & the stack could hold up to n elements, so the space complexity due to the stack is O(n). The StringBuilder also holds the same number of characters as the input string since each stack operator is appended to the result or popped form the stack so it requires O(n) space.