Problem 1:

Step-by-step execution:

Using Example input [3,6,2,1,4] (length: 5)

1) Lines 1-12: When i = 0:
    Then, j = 1, a[i] = 3 < a[j] = 6, so a[i] and a[j] stay the same
    Then, j = 2, a[i] = 3 > a[j] = 2, so tmp = 3, a[i] = 2, and a[j] = 3
    (a = [2,6,3,1,4])
    Then, j = 3, a[i] = 2 > a[j] = 1, so tmp = 2, a[i] = 1, and a[j] = 2
    (a = [1,6,3,2,4])
    Then, j = 4, a[i] = 1 < a[j] = 4, so a[i] and a[j] stay the same
    Then j = 5, so the j-loop ends
2) Lines 1-12: When i = 1:
    (a = [1,6,3,2,4])
    Then, j = 2, a[i] = 6 > a[j] = 3, so tmp = 6, a[i] = 3, and a[j] = 6
    (a = [1,3,6,2,4])
    Then, j = 3, a[i] = 3 > 2, so tmp = 3, a[i] = 2, and a[j] = 3
    (a = [1,2,6,3,4])
    Then, j = 4, a[i] = 2 < a[j] = 4, so a[i] and a[j] stay the same
    Then, j = 5, so the j-loop ends
3) Lines 1-12: When i = 2:
    (a = [1,2,6,3,4])
    Then, j = 3, a[i] = 6 > a[j] = 3, so tmp = 6, a[i] = 3, and a[j] = 6
    (a = [1,2,3,6,4])
    Then, j = 4, a[i] = 3 < a[j] = 4, so a[i] and a[j] stay the same
    Then, j = 5, so the j-loop ends
4) Lines 1-12: When i = 3:
    (a = [1,2,3,6,4])
    Then, j = 4, a[i] = 6 > a[j] = 4, so tmp = 6, a[i] = 4, and a[j] = 6
    (a = [1,2,3,4,6])
    Then, j = 5, so the j-loop ends
5) Line 1: i = 4, so the i-loop ends (length-1 = 4, 4<4 is false)

Using Example input [1,2,3,7,8] (length: 5)

1) Lines 1-12: When i = 0:
    Then, j = 1, a[i] = 1 < a[j] = 2, so a[i] and a[j] stay the same
    Then, j = 2, a[i] = 1 < a[j] = 3, so a[i] and a[j] stay the same
    Then, j = 3, a[i] = 1 < a[j] = 7, so a[i] and a[j] stay the same
    Then, j = 4, a[i] = 1 < a[j] = 8, so a[i] and a[j] stay the same
    Then j = 5, so the j-loop ends
2) Lines 1-12: When i = 1:
    Then, j = 2, a[i] = 2 < a[j] = 3, so a[i] and a[j] stay the same
    Then, j = 3, a[i] = 2 < a[j] = 7, so a[i] and a[j] stay the same
    Then, j = 4, a[i] = 2 < a[j] = 8, so a[i] and a[j] stay the same

Then j = 5, so the j-loop ends
3) Lines 1-12: When i = 2:
Then, j = 3, a[i] = 3 < a[j] = 7, so a[i] and a[j] stay the same
Then, j = 4, a[i] = 3 < a[j] = 8, so a[i] and a[j] stay the same
Then j = 5, so the j-loop ends
4) Lines 1-12: When i = 3:
Then, j = 4, a[i] = 7 < a[j] = 8, so a[i] and a[j] stay the same
Then j = 5, so the j-loop ends
5) Line 1: i = 4, so the i-loop ends (length-1 = 4, 4<4 is false)

This function sorts the integers in an integer array in descending order.

Analysis
Let n denote the length of the array. The block inside the if-statement (lines 7-9) only gets and compares integers within the array, so it takes O(1) time. The if-statement (lines 5-10) only compares two values within the array, so it takes O(1) time. The j-for-loop (lines 3-11) takes linear time O(n), since it iterates n-(n-1) times the first time around, n-(n-2) the second time around, and n-(n-n) the nth time around, which just results in n. The i-for-loop (lines 1-12) would take linear time O(n-1), if not for the nested j-for-loop, since it iterates n-1 times (from i = 0 to i = n-2 [inclusive]). However, because there is an O(n) operation that takes place each time the i-for-loop runs, the overall run-time of the outer i-for-loop (lines 1-12) becomes O(n-1) * O(n) which is O(n^2). This loop is therefore the most expensive operation, and the function overall takes O(n^2) time.

--

Problem 2:
Step-by-step execution:
Using Example input str = "Chocolate", n = 4
1) Line 1: s = ""
2) Lines 2-5: i = 4, s = "Choc"
3) Lines 2-5: i = 3, s = "ChocCho"
4) Lines 2-5: i = 2, s = "ChocChoCh"
5) Lines 2-5, i = 1, s = "ChocChoChC"
6) Line 2: i = 0, so loop ends
7) Line 6: returns s ("ChocChoChC")

Using Example input str = "xyz", n = 3
1) Line 1: s = ""
2) Lines 2-5: i = 3, s = "xyz"
3) Lines 2-5: i = 2, s = "xyzxy"
4) Lines 2-5: i = 1, s = "xyzxyx"
5) Line 2: i = 0, so loop ends

7) Line 6: returns s ("Chocochocchoch")

This function appends a part of a given string to the end of another part of the string, with each appended part being smaller than the last, until there are no more characters in the string part.

Analysis
Let n denote the length of the string. The block inside the for-loop (line 4) uses the substring operation, which we are assuming takes O(n) time. The for-loop (lines 2-5) would take O(n) time without the substring operation within it, since it runs depending on the length of the string, n. However, since the substring operation is run each time the for loop is run, the run-time for the loop becomes O(n) * O(n), which is O(n^2). Line 1 is a simple assignment, which takes O(1) time. Line 6 is a simple return, so it takes O(1) time. Therefore, the overall complexity of the function is O(n^2).

--

Problem 3:
Step-by-step execution:
Using Example input [4,21,9,1], 5 (length: 4)
1) Line 1: idx = -1
2) Lines 2-8: i = 0, a[i] = 4 != 5, so idx stays the same
3) Lines 2-8: i = 1, a[i] = 21 != 5, so idx stays the same
4) Lines 2-8: i = 2, a[i] = 9 != 5, so idx stays the same
5) Lines 2-8: i = 3, a[i] = 1 != 5, so idx stays the same
6) Line 2: i = 4, so loop ends (length = 4, 4<4 is false)
7) Line 9: returns idx (-1)

Using Example input [10,2,8,24,0], 8 (length: 5)
1) Line 1: idx = -1
2) Lines 2-8: i = 0, a[i] = 10 != 8, so idx stays the same
3) Lines 2-8: i = 1, a[i] = 2 != 8, so idx stays the same
4) Lines 2-8: i = 2, a[i] = 8 == 8, so idx = i = 2
5) Lines 2-8: i = 3, a[i] = 24 != 8, so idx stays the same
6) Lines 2-8: i = 4, a[i] = 0 != 8, so idx stays the same
7) Line 2: i = 5, so loop ends (length = 5, 5<5 is false)
8) Line 9: returns idx (2)

This function finds the index (if there exists any) in the provided integer array that corresponds to the given integer value, otherwise returns -1.

Analysis
Let n denote the length of the array. The code block inside the for-loop (lines 4-7) takes O(1) time because it only performs comparisons and assignments of integers. Though the for-loop

(lines 2-8) *could* potentially end early if an index is found, the loop is still considered to take O(n) time because in a worst-case scenario, the index would not be found, and the loop would have to run for the length of the array, n. Line 1 is a simple assignment, which takes O(1) time. Line 6 is a simple return, so it takes O(1) time. Therefore, the overall complexity of the function is O(n).

--

Problem 4:

Step-by-step execution:

Using Example input [3,3] (length: 2)

1) Line 1: a = 0
2) Lines 2-12: When i = 0:
    Then, j = 1, nums[i] = 3 == nums[j] = 3, and j - i + 1 = 2 > a, so a = 2
    Then, j = 0, nums[i] = 3 == nums[j] = 3, and j - i + 1 = 1 < a, so a stays the same
    Then, j = -1, so the loop ends (j >= i, -1 >= 0 is false)
2) Lines 2-12: When i = 1:
    Then, j = 1, nums[i] = 3 == nums[j] = 3, and j - i + 1 = 1 < a, so a stays the same
    Then, j = 0, so the loop ends (j >= i, 0 >= 1 is false)
3) Line 2: i = 2, so the loop ends (length = 2, 2<2 is false)
4) Line 13: returns a (2)

Using Example input [1,2,3,1] (length: 4)

1) Line 1: a = 0
2) Lines 2-12: When i = 0:
    Then, j = 3, nums[i] = 1 == nums[j] = 1, and j - i + 1 = 4 > a, so a = 4
    Then, j = 2, nums[i] = 1 != nums[j] = 3, so a stays the same
    Then, j = 1, nums[i] = 1 != nums[j] = 2, so a stays the same
    Then, j = 0, nums[i] = 1 == nums[j] = 1, and j - i + 1 = 1 < a, so a stays the same
    Then, j = -1, so the loop ends (j >= i, -1 >= 0 is false)
2) Lines 2-12: When i = 1:
    Then, j = 3, nums[i] = 2 != nums[j] = 1, so a stays the same
    Then, j = 2, nums[i] = 2 != nums[j] = 3, so a stays the same
    Then, j = 1, nums[i] = 2 == nums[j] = 2, and j - i + 1 = 1 < a, so a stays the same
    Then, j = 0, so the loop ends (j >= i, 0 >= 1 is false)
3) Lines 2-12: When i = 2:
    Then, j = 3, nums[i] = 3 != nums[j] = 1, so a stays the same
    Then, j = 2, nums[i] = 3 == nums[j] = 3, and j - i + 1 = 1 < a, so a stays the same
    Then, j = 1, so the loop ends (j >= i, 1 >= 2 is false)
4) Lines 2-12: When i = 3:
    Then, j = 3, nums[i] = 1 == nums[j] = 1, and j - i + 1 = 1 < a, so a stays the same
    Then, j = 2, so the loop ends (j >= i, 2 >= 3 is false)
5) Line 2: i = 4, so the loop ends (length = 4, 4<4 is false)
6) Line 13: returns a (4)

Using Example input [1,4,2,1,4,3,4] (length: 7)

1) Line 1: a = 0

2) Lines 2-12: When i = 0:

Then, j = 6, nums[i] = 1 != nums[j] = 4, and j - i + 1 = 4 > a, so a = 4

Then, j = 5, nums[i] = 1 != nums[j] = 3, so a stays the same

Then, j = 4, nums[i] = 1 != nums[j] = 4, so a stays the same

Then, j = 3, nums[i] = 1 == nums[j] = 1, and j - i + 1 = 4 > a, so a = 4

Then, j = 2, nums[i] = 1 != nums[j] = 2, so a stays the same

Then, j = 1, nums[i] = 1 != nums[j] = 4, so a stays the same

Then, j = 0, nums[i] = 1 == nums[j] = 1, and j - i + 1 = 1 < a, so a stays the same

Then, j = -1, so the loop ends (j >= i, -1 >= 0 is false)

2) Lines 2-12: When i = 1:

Then, j = 6, nums[i] = 4 == nums[j] = 4, and j - i + 1 = 6 > a, so a = 6

Then, j = 5, nums[i] = 4 != nums[j] = 3, so a stays the same

Then, j = 4, nums[i] = 4 == nums[j] = 4, and j - i + 1 = 4 < a, so a stays the same

Then, j = 3, nums[i] = 4 != nums[j] = 1, so a stays the same

Then, j = 2, nums[i] = 4 != nums[j] = 2, so a stays the same

Then, j = 1, nums[i] = 4 == nums[j] = 4, and j - i + 1 = 1 < a, so a stays the same

Then, j = 0, so the loop ends (j >= i, 0 >= 1 is false)

3) Lines 2-12: When i = 2:

Then, j = 6, nums[i] = 2 != nums[j] = 4, so a stays the same

Then, j = 5, nums[i] = 2 != nums[j] = 3, so a stays the same

Then, j = 4, nums[i] = 2 != nums[j] = 4, so a stays the same

Then, j = 3, nums[i] = 2 != nums[j] = 1, so a stays the same

Then, j = 2, nums[i] = 2 == nums[j] = 2, and j - i + 1 = 1 < a, so a stays the same

Then, j = 1, so the loop ends (j >= i, 1 >= 2 is false)

4) Lines 2-12: When i = 3:

Then, j = 6, nums[i] = 1 != nums[j] = 4, so a stays the same

Then, j = 5, nums[i] = 1 != nums[j] = 3, so a stays the same

Then, j = 4, nums[i] = 1 != nums[j] = 4, so a stays the same

Then, j = 3, nums[i] = 1 == nums[j] = 1, and j - i + 1 = 1 < a, so a stays the same

Then, j = 2, so the loop ends (j >= i, 2 >= 3 is false)

5) Lines 2-12: When i = 4:

Then, j = 6, nums[i] = 4 == nums[j] = 4, and j - i + 1 = 3 < a, so a stays the same

Then, j = 5, nums[i] = 4 != nums[j] = 3, so a stays the same

Then, j = 4, nums[i] = 4 == nums[j] = 4, and j - i + 1 = 1 < a, so a stays the same

Then, j = 3, so the loop ends (j >= i, 3 >= 4 is false)

6) Lines 2-12: When i = 5:

Then, j = 6, nums[i] = 3 != nums[j] = 4, so a stays the same

Then, j = 5, nums[i] = 3 == nums[j] = 3, and j - i + 1 = 1 < a, so a stays the same

Then, j = 4, so the loop ends (j >= i, 4 >= 5 is false)

6) Lines 2-12: When i = 6:

Then, j = 6, nums[i] = 4 == nums[j] = 4, and j - i + 1 = 1 < a, so a stays the same
Then, j = 5, so the loop ends (j >= i, 5 >= 6 is false)
7) Line 2: i = 7, so the loop ends (length = 7, 7<7 is false)
6) Line 13: returns a (6)

This function appears to count starting from the first repeating number to its last appearing counterpart. That is, in an array containing [1,2,3,4,2], the result would be 4 because counting starting from the first appearing duplicate (2), until the last appearing duplicate, we get 4 (2,3,4,2 -> 1,2,3,4).

Analysis
Let n denote the length of the array. The code block inside the j-for-loop (lines 6-10) takes O(1) time because it only performs comparisons and assignments of integers. The j-for-loop itself (lines 4-11) runs n-1 times (j = nums.length-1 to j = 1 [inclusive]), which gives it a run-time of O(n-1). The i-for-loop (lines 2-12) runs n times (i = 0 to i = nums.length [end-point exclusive]), which without the j-for-loop would mean it would have a complexity of O(n). However, because there is an O(n-1) operation that takes place each time the i-for-loop runs, the overall run-time of the outer i-for-loop (lines 2-12) becomes O(n) * O(n-1) which is O(n^2). This loop is therefore the most expensive operation, and the function overall takes O(n^2) time.