Week 2 Quiz

Question #1

TRUE

Easier way to solve this: O(n) is the **upper** bound of the asymptomatic behavior of a function, so f(n) = . Therefore TRUE.

Question #2

[7, 9, -11, 13, 15, -17, 19, 21] 🡪 56

Check codebase.py.

Question #3

Assume according to the quiz itself. Proof by induction:

Base case: , then it is true for .

Inductive step:

, substitute with the inequation.

Question #4

a = 8, b = 2

1.

The watershed function , runs polynomially faster than f(n) = . So case 1 applies, .

2.

grows at nearly the same rate as f(n) = , so case 2 applies, , when k = 0, .

3.

grows polynomially slower than as f(n) = , and , aka for some exists, so case 3 applies. Thus, .

Week 3 Quiz

Question #1

10

Question #2

C = [5, 7, 7, 10, 12]

Step 1: Determine the range of values (k).

The minimum value is 1 and the maximum is 5, so k = 5.

Step 2: Initialize the counting array C.

C = [0, 0, 0, 0, 0] (indices 0 to 4, representing values 1 to 5)

Step 3: Count occurrences of each element.

For each element in A, increment the corresponding count in C. After this step: C = [5, 2, 0, 3, 2] (5 ones, 2 twos, 0 threes, 3 fours, 2 fives)

Step 4: Compute running sums in C.

Use index i to represent the actual value, so we start from i = 1 to 5:

C[i] = C[i] + C[i-1] for i from 1 to 5

C[1] = C[1] + C[0] = 5 + 0 = 5

C[2] = C[2] + C[1] = 2 + 5 = 7

C[3] = C[3] + C[2] = 0 + 7 = 7

C[4] = C[4] + C[3] = 3 + 7 = 10

C[5] = C[5] + C[4] = 2 + 10 = 12

After this step: C = [5, 7, 7, 10, 12]

Step 5: Build the output array B.

Initialize B with the same length as A: B = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].

Now, iterate through A from right to left:

A[12] = 4: B[C[4]] = B[10] = 4, then C[4] = 9

A[11] = 4: B[C[4]] = B[9] = 4, then C[4] = 8

A[10] = 5: B[C[5]] = B[12] = 5, then C[5] = 11

A[9] = 1: B[C[1]] = B[5] = 1, then C[1] = 4

A[8] = 2: B[C[2]] = B[7] = 2, then C[2] = 6

A[7] = 1: B[C[1]] = B[4] = 1, then C[1] = 3

A[6] = 4: B[C[4]] = B[8] = 4, then C[4] = 7

A[5] = 1: B[C[1]] = B[3] = 1, then C[1] = 2

A[4] = 5: B[C[5]] = B[11] = 5, then C[5] = 10

A[3] = 1: B[C[1]] = B[2] = 1, then C[1] = 1

A[2] = 2: B[C[2]] = B[6] = 2, then C[2] = 5

A[1] = 4: B[C[4]] = B[7] = 4, then C[4] = 6

Final sorted array: B = [1, 1, 1, 1, 1, 2, 2, 4, 4, 4, 5, 5]

Question #3

True. As referred from CLRS p. 207:

Assume a decision tree of height with reachable leaves corresponding to a comparison sort on elements.

Each of the permutations appear as one or more leaves,

.

A binary tree of height has no more than leaves,

,

,

. As for the upper bound and lower bound of ,

,

,

The upper bound of is . Similarly,

,

,

, the lower bound of is .

, according to Theorem 3.1 (CLRS p. 56)

Question #4

True, when it is the best and average case for quick select. Pseudocodes

are as following:

QUICK-SELECT(arr, k):

# Randomly choose a pivot from the array.

pivot = random.choice(arr)

# Split the array into three parts, which are lists of numbers lower than the pivot,

# equal to the pivot, and higher than the pivot.

low\_nums = [element for element in arr if element < pivot]

pivots = [element for element in arr if element = pivot]

high\_nums = [element for element in arr if element > pivot]

if k < len(low\_numbers):

return QUICK-SELECT(low\_nums, k)

elif k < len(low\_numbers) + len(pivots):

# When k is larger than the length of smaller numbers but smaller than the sum of

# length of lower and equal numbers, then obviously the k-th number is in the pivot

# list.

return pivot[0]

else:

return QUICK-SELECT(high\_nums, k – len(low\_nums) – len(pivots))

And the time complexity of above quick select algorithm is , since on average we split the array into two halves, and each time we would recurse one list, based on the length of those lists. Therefore, we know that:

Thus, the upper bound of above algorithm is .