**CS 514**

**Quiz 1**

**Q1. Prove by contradiction that O( )**

**Solution -**

Assume the contrary that O( ).

Thus, there are constants c > 0 and >= 0 such that for all n >= ,

c

This is because should be the upper bound of

Therefore, for all n >= ,

c

Taking power of 3 on both sides, we get

* ( c

Dividing the inequality on both sides by n, we get



Now, cannot possibly hold true for arbitrarily large values of n since c is a constant.

Hence, here we have arrived at a contradiction.

This means that our original assumption that O( ) is wrong.

**Q2. Find the time complexity of the following foo2 function.**

foo2(n):

sum = 0

for i in range(n):

for j in range(n):

if( i == j):

for k in range(n\*n):

sum = i + j + k

**Solution -**

To find the time complexity of the function, we will follow the given steps :

**Step 1.** Identify the input parameter(s) that would impact the running time of the algorithm

Here, we see that the input parameter that would impact the running time of the algorithm is **n**.

**Step 2.** Identify the**basic** operation, that which would be executed a maximum number of times and impacts the execution time of the algorithm (this is usually located in the innermost loop).

To identify the basic operation, we consider each statement and calculate the number of times that particular statement is executed.

1. The assignment sum = 0 will only be executed once.
2. The first “for” loop is where i iterates from 0 to n, so this will be executed n times.
3. The second for loop is inside the first “for” loop so it will automatically be executed n times. As it also iterates from 0 to n, in total it will be executed n\*n = n^2 times.
4. The conditional “if” statement is inside the two “for” loops and so it will also be executed n\*n = n^2 times.
5. Now, the third “for” loop will be executed only when i == j. When we understand the logic of the program, we see that j iterates 0 to n times for every iteration of i. So, the case where i == j happens **only once** in every iteration of j. Hence, as this for loop is executed only when the condition i == j satisfies, it will be executed n times(even though two for loops precede it).
6. The assignment sum = i + j + k is inside the third “for” loop and this loop iterates from 0 to n^2, this statement is executed n^3 times.

Through this we identify that the basic operation which is executed most no. of times and impacts the execution time is the statement **sum = i + j + k.**

**Step 3.** Determine worst, average, and best cases for the input of size n, if the number of times the basic operation gets executed varies with specific instances of input.

The number of comparisons between i and j does not vary with the size of the array n. The comparison “if (i == j)” is inside a nested loop, which iterates n times for both i and j. Therefore, this comparison is performed `n \* n` times, regardless of the value of ‘n’.

Hence, we need not distinguish between worst case, best case and the average case for this algorithm.

**Step 4.** Set up a sum for the number of times the basic operation is executed and simplify it using standard summation formulas.

The basic operation sum = i + j + k is executed within the innermost loop, which is entered when the condition i == j is satisfied. The innermost loop iterates from 0 to n\*n-1. A sum for the number of times this operation is executed as follows:

Total Basic Operations T(n) = ∑(i=0 to n-1) ∑(j=0 to n-1) ∑(k=0 to n\*n-1) δ (i == j) 1

= n \* (n\*n)

=

Therefore, the time complexity of the code is

**Q3. Assume A is a list of integers. What is the correct loop invariant for the code provided below:**

for i in range(len(A)): # in pseudo-code for i=0,...,len(A)-1

answer += A[i]

return answer

**Solution -**

Assuming that answer was initialized to 0 before the start of the loop (refer Ed discussion- <https://edstem.org/us/courses/48189/discussion/3679617>), the correct loop invariant for the code provided is :

“After i iterations, the variable “answer” will be the sum of the first i elements of the list A”.

**Initialization:**

Before the loop starts, answer is 0, and as the execution has not entered the loop which means i is set to 0. So, the loop invariant asserts that answer should be the sum of the first i elements of list A and since we haven't included any elements from A yet, answer is indeed equal to its first I elements (0 in this case), satisfying the initialization condition.

**Maintenance**:

Assuming the loop invariant is true after i iterations, which means answer holds the sum of the first i elements of list A, we need to prove that it remains true after the (i+1)-th iteration. In the (i+1)-th iteration, we add A[i+1] to answer, so answer becomes the sum of its previous value (sum of the first i elements) and A[i+1]. This exactly matches the loop invariant, confirming the maintenance condition.

**Termination:**

The loop invariant holds after each iteration, ensuring that at the end of the loop, answer is the sum of the first len(A) elements of list A. The loop runs for i in the range of 0 to len(A) - 1, and it terminates when i reaches len(A), which is consistent with the exit condition of the loop. Consequently, after the final iteration, when i equals len(A), answer will be the sum of all elements in list A, effectively summing the entire list A. This unquestionably satisfies the termination condition and demonstrates that the loop has correctly computed the sum of all elements in list A, in alignment with the loop invariant.

Therefore, it is proved that the loop invariant is maintained throughout the loop, and it also holds at termination. This proves the correctness of the algorithm of finding the sum of the elements of the list while maintaining the specified loop invariant.

**Q4. Assume a is a list of integers. Consider the following pseudocode.**

def mystery (a):

temp = 5

for i in range(len(a)):

# Assertion 1: temp =

if a[i] > temp:

temp = a[i]

# Assertion 2: temp =

**(a) (6 points) Fill in the Assertion 1 which holds in the ith iteration and Assertion 2 which holds after the For loop exits.**

**(b) (10 points) Give an inductive proof that the Assertion 1 holds in all iterations.**

**Solution –**

**(a)** **Assertion 1:**

When we consider this assertion, we see that the value of temp after the start of the “for” loop and before the “if” statement depends on the previous iteration of the “for” loop. The value of temp after the end of previous iteration of the loop will be the value of temp at the start of the present iteration off the loop before it goes through the if statement. Considering this, we can state the first assertion as -

"In any ith iteration, Assertion 1 asserts that the variable 'temp' holds the maximum value encountered in the portion of the list “a” that has been processed up to index i – 1, if the maximum value is greater than 5; otherwise, it maintains the value of 5 assigned before the start of the loop.”

**Assertion 2**

“Assertion 2 asserts that the value of temp after the for loop exists will be the maximum value of the array if the maximum value is greater than 5; otherwise, it will be 5.”

**(b) Proof of Assertion 1 through mathematical induction**

**Base Case (i = 0):**

Before the start of first iteration, ‘temp’ is initialized to 5. In the first iteration (i = 0), the loop hasn't processed any elements yet so as the variable 'temp' is initially set to 5, and there are no elements in the portion of the list processed so far, so the maximum value is 5, which makes the assertion true. So, the base case holds.

**Inductive Hypothesis:**

Assume that Assertion 1 holds for some arbitrary value of i = k. That is, in the k-th iteration, the variable 'temp' holds the maximum value encountered in the portion of the list "a" processed up to index k - 1, if the maximum value is greater than 5. Otherwise the value is 5 (which is assigned before the start of the for loop).

**Inductive Step:**

We need to prove that Assertion 1 also holds for the iteration with index ‘k + 1’. As we know that the value of temp depends on the previous iteration, we need to prove the inductive step considering the entire cycle of execution of the loop for previous iteration ‘k’.

At the start of the iteration with index ‘k’, we need to consider two cases:

**Case 1: a[k] > temp (i.e., the element is greater than the ‘temp’ value in the previous iteration)**

In this case, ‘temp’ is updated to the new value a[k], and it becomes the maximum value in the subarray [0:k] since it's greater than the old value of ‘temp’. And as this value is not changed at the start of k+1 iteration, it satisfies Assertion 1.

**Case 2: a[k] <= temp (i.e., the element is not greater than the ‘temp’ in the previous iteration).**

In this case, ‘temp’ remains the same (it's already the maximum value in the subarray [0:k] if the value greater than 5, otherwise it is 5). It still holds the maximum value encountered up to index k. Therefore, the value of temp will be 5 or the maximum value encountered till index k. So Assertion 1 holds for i = k + 1

By the inductive hypothesis and the inductive step, we've shown that if Assertion 1 holds for i = k, it also holds for i = k + 1. This completes the inductive proof.

By the principle of mathematical induction, we have shown that Assertion 1 holds for all iterations from 0 to n-1.