# Assignment 01
# EECS 3101


## Anuj Kumar
## 214297998

## Problem 1

(a) Find the proper loop invariant for the while-loop

**Inv(p1, p2, p3)**:
At the start of each iteration of the **while** loop of lines 7–18, the following conjuncts are true:

1. If p3>0, then the sub-array of lst from 0 to p3-1 (including both indices) consists of the elements originally in lst from 0 to p3-1, but in sorted order.
2. p1 <= p2 <= p3
3. if p1>0, then the sub-array of lst from 0 to p1-1 contains only 'A'
4. if p2>0, then the sub-array of lst from p1 to p2-1 contains only 'B'
5. if p3>0, then the sub-array of lst from p2 to p3-1 contains only 'C'
6. p3 <= lst.length

(b) Prove the invariant that you find in (a), using induction.

Let's consider the base case, which is before the start of the while loop. p1=p2=p3=0. It is easy to see that all the above 6 conditions are true. Hence the invariant Inv(p1,p2,p3) is true for the base case.

Now let's assume that the invariant is true at an arbitrary iteration of the while loop. Then we have to show that the invariant will be true for the next iteration as well.

So at iteration n, we know the sub-array of lst from 0 to p3-1 consists of the elements originally in lst from 0 to p3-1, but in sorted order. Then for iteration n+1, we have to consider several cases;

Case 1: lst[p3] == 'A': if the character at position p3 is 'A', then we swap lst[p3] with the lst[p1]. There are three sub cases. lst[p1] can be either 'A', 'B' or 'C'

> Case 1.1: lst[p1] == 'A': This can happen only if p1=p2=p3 (by condition 3, 'A' appears only up to p1-1, so if 'A' appears again at p1, that would mean there are no 'B's and 'C's in the lst from 0 to p3). So swapping changes nothing, p1 is incremented by 1, p2 is also incremented by 1 and p3 is also incremented by 1. So p1<=p2<=p3 also holds true. All conditions hold true & the inv(p1, p2, p3) holds true.
>
> Case 1.2: lst[p1] == 'B': After swapping lst[p1] with lst[p3], lst[p3] becomes 'B' and lst[p1] becomes 'A'. p1 is incremented. The condition that the sub-array of lst from 0 to p1-1 contains only 'A' holds true again. Earlier we had lst[p1:p2-1] to be 'B', but after p1 is incremented, p2 is set as max(p1,p2) which means that p2 is at least p1. So again, lst[p1:p2-1] are all 'B' holds true again.
>
> But we have a 'B' at lst[p3] and there could be 'C' before that (from lst[p2:p3-1]). This breaks the sort order (condition 1), but it is taken care of by the second if condition inside the while loop. lst[p3] & lst[p2] are swapped. That is, lst[p2] is set as 'B' and p2

is incremented. Hence the condition that sub-array of lst from p1 to p2-1 contains only 'B' holds true once again. If p2 was equal to p3, then the swapping is b/w the same position, p2 incremented so condition 4 holds true. Otherwise (if p3 was greater than p2), then 'C' was at p2 (by condition 5), and it goes to p3. P3 is incremented by 1. The condition 5 holds true again. p1<=p2<=p3 also holds true. All the 6 conditions will hold true and hence the invariant is true.

Case 1.3: lst[p1] == 'C': This would mean that there are no 'B' from 0 to p3-1 & p1 is equal to p2. (by conditions 4 & 5). Then swapping p1 & p3, means 'A' comes to p1 (& p1 is incremented means condition 3 holds true), and 'C' comes to p3. Since there are no 'B's, condition 4 is trivially true, and condition 5 also holds true because p3 will be incremented by 1. The inv(p1, p2, p3) holds true.

Case 2: lst[p3] == 'B': if the character at position p3 is 'B', then it is swapped with character at positon p2. p2 can be equal to p3, in which case p2 is simply incremented (so condition 5 holds true). P3 is also incremented so condition 5 is also true trivially.

On the other hand, if p2 < p3, then the character at lst[p2] is 'C' (by condition 5). Hence C goes to p3 and B goes to p2. P2 is incremented, so condition 4 holds true again. Also p3 is incremented, so condition 5 also holds true. Condition 3 will also hold true, because we haven't changed any element from 0 to p1-1.

Case 3: lst[p3] == 'C': Both the if conditions are false. The p3 is incremented by 1. So the condition 5 that 'C' appears from p2 to p3-1 holds true again. The other conditions also hold true, since we haven't touched p1 or p2. The inv(p1, p2, p3) holds true.

Hence by the principle of induction, the inv(p1, p2, p3) is true for any arbitrary iteration n.

(c) If the loop terminates, then p3 = lst.length. Hence by condition 1, the lst contains all original elements in lst but in sorted order. Hence SortABC is proven to work correctly (partially correct) if it terminates.

(d) It is easy to see that the loop terminate after finite number of iterations. In fact the loop terminates after exactly n number of iterations (where n = lst.length). This is because the variable p3 is incremented by 1 in each iteration and as soon as p3 becomes equal to n, the loop terminates.

## Problem 2

(a) Find the proper loop invariant Inv(x, y)

The loop invariant contains the following conjuncts;

Inv(x, y):
Before the start of the while loop from lines 6-13, the following conditions should hold true;

1. x <= LCM (x, y)
2. y <= LCM(x, y)
3. x > 0
4. y > 0

(b) Prove the invariant by induction.

Base Case: Initially x=a, y=b, if the condition inside the while loop becomes false, i.e. x==y, then a=b => LCM(a,b) = a. Hence (return x) will return a, which is the correct solution. So all the above conditions of the invariant are true. (by preconditions, a>=1 & b>=1, so x > 0 & y>0 hold true. Also x<=LCM(x,y) and y<=LCM(x,y) is also true. So the invariant is true). Otherwise, if x!=y, then also the invariant is true trivially.

Induction: Assume the inv(x, y) is true at step n, then we have to show that the invariant is true for step n+1 as well.

Case 1: y = LCM(x, y): In this case x < y ( otherwise the if x=LCM(x,y) then the loop would have exited in the previous iteration itself, or if x > LCM(x,y), then the condition 1 will be violated, which we assumed to be true). Then we increment x by a. x cannot exceed LCM(x,y) because we increment x by a every step, and by the definition of LCM, LCM(x,y) is an integer multiple of a. So x has to be <= LCM(x,y) after update. Hence the condition holds true.

Case 2: x = LCM(x, y): (same argument with x & y interchanged) y < x. y is an integer multiple of b and LCM(x, y) is also an integer multiple of b. So y <=LCM(x, y)

Case 3: Both x & y < LCM(x, y): None of x or y can exceed LCM(x, y) since LCM(x, y) is an integer multiple of a & b. x is an integer multiple of a and y is an integer multiple of b.

Hence the invariant inv(x, y) holds true at iteration n+1 as well.

(c ) If the loop terminates, the x = y. But we know by the definition of LCM(x, y), it is the smallest number where a integer multiple of a becomes equal to an integer multiple of b. Coupled with the invariants that x<=LCM(x,y) & y<=LCM(x,y), x = y = LCM(x, y).

Hence the proof.