NO.

DATE . .

DSAA (H) Lab 07.

Q.7.1:

	1	2	3	4	5	6	7
A	0	2	0	1	3	1	1

	0	1	2	3
⇒ C	2	3	1	1

	0	1	2	3
⇒ C	2	5	6	7

	1	2	3	4	5	6	7
⇒ B					1		

	0	1	2	3
C	2	4	6	7

	1	2	3	4	5	6	7
⇒ B				1	1		

	0	1	2	3
C	2	3	6	7

	1	2	3	4	5	6	7
⇒ B				1	1		3

	0	1	2	3
C	2	3	6	6

	1	2	3	4	5	6	7
⇒ B			1	1	1		3

	0	1	2	3
C	2	2	6	6

	1	2	3	4	5	6	7
⇒ B		0	1	1	1		3

	0	1	2	3
C	1	2	6	6

	1	2	3	4	5	6	7
⇒ B		0	1	1	1	2	3

	0	1	2	3
C	1	2	5	6

	1	2	3	4	5	6	7
⇒ B	0	0	1	1	1	2	3

	0	1	2	3
C	0	2	5	6

NO.

DATE

Q.7.2

Proof: Since the previous two for loops are correct,
we have: at the end of the two for loops,
the array C contains in each position C[i] the
number of elements equal to i.

Loop Invariant: At the end of the i-th iteration of the
for loop in lines 6-7, the subarray C[0, ..., i]
contains in each position C[j] the # of
elts. less than or equal to j.

Initialization: At the end of the 1-st iteration:
C[1] = C[0] + C[1]
= # of elts equal to 0
+ # of elts equal to 1
= # of elts ≤ 1. ✓

Btw: C[0] = # of elts equal to 0
= # of elts ≤ 0. ✓.

Maintenance: Assume that the loop invariant holds for
the past 1st, 2nd, ..., i-th iterations.
For the (i+1)-th iteration:
C[0], ..., C[i] holds the truth that they
are numbers of elts. less than or equal to j.
Only need to check C[i+1].
C[i+1] = C[i] + C[i+1]
= # of elts ≤ i
+ # of elts = (i+1)
= # of elts ≤ (i+1). ✓.

Termination: At the end of the last iteration:
the array C[0, ..., k] = C contains in each
position C[i] the number of elts less than
or equal to i. ✓. □

NO.

DATE

Q 7.3:

the algorithm : ③ be not stable but sort the numbers.

Proof. 1° not stable:

Consider 2 adjacent equal numbers in A, namely A[i]=A[i+1]

Suppose when it comes to assign A[i] to a place in B, C[A[i]] = x.

⇒ B[x] ← A[i]

Then, C[A[i]] ← C[A[i]) - 1.

⇒ B[x-1] ← A[i+1].

Now, you see that: originally, A[i+1] is to the right of A[i]. But in B, their position relationship is swapped.

⇒ Not stable ✓

2° sort the numbers.

Not considering stability, the reason why it sorts the numbers corresponds to the reason why the original CountSort sorts the numbers. just reversing the direction of assigning values to B. ✓ □

Q. 7.4:

Algorithm: CountingRange (A, k, a, b).

```
1. let C[0, ..., k] be a new array.                    } O(k).
2. for i = 0 to k do
3.     C[i] = 0
4. for j = 1 to A.length do                            } O(n)
5.     C[A[j]] = C[A[j]] + 1.
6. for i = 1 to k do                                   } O(k).
7.     C[i] = C[i] + C[i-1].
8. OUTPUT C[b] - C[a-1].                               } O(1)
```

Time complexity: $O(k) + O(n) + O(k) + O(1) = O(n+k)$. ✓.

NO.

DATE

Q.7.5:

COW	MOB	TAB	BAR
DOG	TAB	BAR	BIG
TUG	DOG	CAR	BOX
ROW	TUG	TAR	CAR
MOB	PIG	PIG	COW
BOX ⇒	BIG ⇒	BIG ⇒	DOG
TAB	BAR	MOB	MOB
BAR	CAR	DOG	PIG
CAR	TAR	COW	ROW
TAR	COW	ROW	TAB
PIG	ROW	WOW	TAR
BIG	WOW	BOX	TUG
WOW	BOX	TUG	WOW.

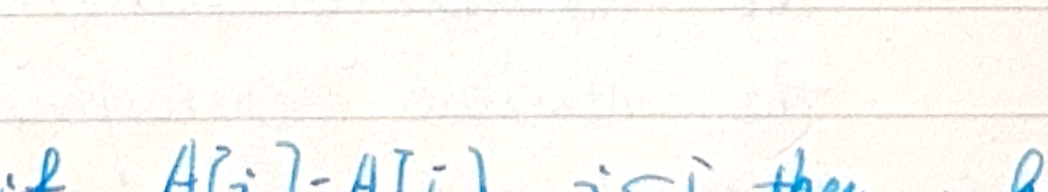
Q.7.6

1. Insertion Sort: Stable since if $A[i]=A[j]$, $i<j$, then when $A[j]$ is continuing its way left, it'll stop to the right of $A[i]$, which preserves stability.

2. Merge Sort: Stable. since the divide action preserves stability and the merge action keeps $L[i]$ left to $R[j]$ if $L[i]=R[j]$.

3. Heap Sort: NOT Stable.

Counter Ex: $[3_a, 3_b, 1_c]$:



resulted in: $[1_c, 3_b, 3_a]$ violating stability!

4. Quick Sort: NOT Stable.

Counter Ex: $[3_a, 3_b, 1_c]$: $[3_a, 3_b, 1_c] \Rightarrow [1_c, 3_b, 3_a]$.

not stable!