Exercise Sheet 7

Handout: Oct 21st — Deadline: Oct 28rd - 4pm

Question 7.1 (marks 0.25)

Illustrate the operation of `CountingSort(A, B, 3)` on the array

0	**2**	**0**	**1**	**3**	**1**	**1**

with input elements from the set {0, 1, 2, 3}

1. For each of the first three for loops, write down the contents of array C after the loop has ended.
2. For the last for loop, write down the contents of the arrays B and C at the end of each iteration of the loop.

Answer

1.

C after the first for loop

0	**0**	**0**	**0**

C after the second for loop

2	**3**	**1**	**1**

C after the third for loop

2	**5**	**6**	**7**

2.

after j=7,B and C

none	**none**	**none**	**none**	**1**	**none**	**none**

2	**4**	**6**	**7**

after j=6,B and C

none	**none**	**none**	**1**	**1**	**none**	**none**

2	**3**	**6**	**7**

after j=5,B and C

none	**none**	**none**	**1**	**1**	**none**	**3**

2	**3**	**6**	**6**

after j=4,B and C

none	**none**	**1**	**1**	**1**	**none**	**3**

2	**2**	**6**	**6**

after j=3,B and C

none	**0**	**1**	**1**	**1**	**none**	**3**

1	**2**	**6**	**6**

after j=2,B and C

none	**0**	**1**	**1**	**1**	**2**	**3**

1	**2**	**5**	**6**

after j=1,B and C

0	**0**	**1**	**1**	**1**	**2**	**3**

0	**2**	**5**	**6**

Question 7.2 (marks 0.25)

Prove that after the for loop in lines 6-7 (in the pseudo-code
provided at lecture) of CountingSort the array C contains in each position C[i] the number of elements less than or equal to i (You can assume the previous two for loops are correct).

Answer

After loop1,loop2, the array C is correctly constructed such that C[i] is the number of elements equals to i in the oringinal array A. We can use loop invariance to prove the correctness of the third loop.

- Loop invariance: Before the ith iteration of loop3, C[j] is the number of elements less than or equal to j in the original array A, for every j <= i.
- initialization: Before the first iteration, C[0] is the number of elements equals to 0 in the original array A, so the loop invariance holds.
- Maintenance: Before iteration j, C[j] is the number of elements less than or equal to j in the original array A, according to loop invariance. Then thourch the j+1th iteration, C[j+1] = C[j+1] + C[j] where C[j+1] was the number of elements equals to j+1 in the original array A, so the new C[j+1] is the number of elements less than or equals to j+1 in the original array A.C[1..j] is unchanged, so the loop invariance is restablished.
- Termination: The loop ends at i=k+1, which means for all elements in C ,C[i] is the number of elements less than or equal to i in the original array A

Question 7.3 (0.25 marks)

Suppose that we were to rewrite the last for loop header of CountingSort as
`for j = 1 to A.length`.
Then the algorithm:

1. Will not be stable and will not sort the numbers
2. Will be stable but will not sort the numbers
3. Will not be stable but will sort the numbers
4. Will be stable and will sort the numbers
   Justify your answer.

Answer

3 Will not be stable but will sort the numbers
After the third loop, C becomes a prefix-sum of number of elements in A, and each C[i] is essencially the last position of elements in the bucket, so iterate from 1 to A.length will lead to reversed order in each bucket(not stable), but the ability to sort the array is not affected.

Question 7.4 (0.5 marks)

Describe an algorithm `CountingRange(A, k, a, b)` that given n integers in the range 0 to k, preprocesses its input and and then answers any query about how many integers are present in the range [a : b] in O(1) time. The algorithm should use O(n + k) preprocessing time.

Answer

```
function CountingRange(A, k, a, b):
    for i=0 to k do
        C[i] = 0
    for j=1 to A.length do
        C[A[j]] = C[A[j]] + 1
    for i=1 to k do
        C[i] = C[i − 1] + C[i]

    if a = 0 then
        return C[b]
    else
        return C[b] − C[a − 1]
```

Question 7.5 (0.25 marks)

Illustrate the operation of RadixSort on the following list of English words:

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

Answer

We will sort the words from the last character to the first character using a stable sort (CountingSort). The number of possible characters is 26 (A-Z).

First, sort the list by the last character:

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

Then, sort the list by the second character:

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

Finally, sort the list by the first character:

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

Question 7.6 (0.25 marks)

State which of the following algorithms are stable and which are not:

1. InsertionSort
2. MergeSort
3. HeapSort
4. QuickSort
   For those that are stable argue why. For those that are not stable provide an example of an input that shows instability.

Answer

1. InsertionSort - Stable. When inserting an element into the sorted portion of the array, if it is equal to an existing element, it is placed after it, preserving the original order.
2. MergeSort - Stable. During the merge process, if two elements are equal, the element from the left subarray is chosen first, preserving the original order.
3. HeapSort - Not stable. During the process of MaxHeapifying, if the left child node and the right are equal, the right child is chosen first, which can change the original order of equal elements. Example: Consider the array $[5, 8_1, 8_2 ...]$. After MaxHeapifying, the result is $[8_2, 8_1, childof(8_2)...]$.
4. QuickSort - Not stable. During partition process, the order of equal elements in the larger part is changed every iteration. Example: Consider the array $[5, 8_1, 8_2, 8_3, 6, 7]$. After partitioning, the order of the result is $[5, 6, 7, 8_3, 8_1, 8_2]$. And calling QuickSort recursively on the larger part cannot guarantee the order of equal elements is preserved.