Prof. Pietro S. Oliveto

CS217 Data Structures and Algorithm Analysis
Fall term 2023

Solutions for Exercise Sheet 7

Handout: Oct 31st — Deadline: Nov 7th - 4pm

Question 7.1 (0.25 marks)

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.

Solution: in the following table the symbol ␣ indicates elements that have not been filled yet.

	Array C				Array B						
	0	**1**	**2**	**3**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
After first for loop	0	0	0	0							
After second for loop	2	3	1	1							
After third for loop	2	5	6	7							
After fourth loop iteration:											
$j = 7$	2	4	6	7	␣	␣	␣	␣	1	␣	␣
$j = 6$	2	3	6	7	␣	␣	␣	1	1	␣	␣
$j = 5$	2	3	6	6	␣	␣	␣	1	1	␣	3
$j = 4$	2	2	6	6	␣	␣	1	1	1	␣	3
$j = 3$	1	2	6	6	␣	0	1	1	1	␣	3
$j = 2$	1	2	5	6	␣	0	1	1	1	2	3
$j = 1$	0	2	5	6	0	0	1	1	1	2	3

Question 7.2 (0.25 marks) 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).

Solution: We use the loop invariant:

"At iteration i of the loop each element of the array C at position C[x] for $x < i$ contains the number of elements in A that are less or equal to x".

Initialisation: For i=1, the array contains in $A[0]$ the number of elements that are less or smaller than 0.

maintenance: At the start of iteration i the loop invariant tells me that in position $C[i-1]$ there are the elements that are less than or equal to $i-1$. Now by adding $C[i-1]$ to $C[i]$ which are the the number of elements equal to i in A (true by assumption that the previous loop is correct) the loop invariant is re-established: at the start of the next iteration of the loop the number of elements less than or equal to i will be in position $C[i]$ and positions up to $i-1$ are untouched.

termination: The loop terminates for $i = k+1$. Then the loop invariant states that each position $C[x], 0 \leq x \leq k$ contains the numbers of elements in A that are less than or equal to x, as desired.

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.

Solution: Item 3. The algorithm will insert the numbers in sorted order as it checks how many numbers are smaller or equal to it and slots it in the following position. However the algorithm is not stable as each equal number to itself will appear in the sorted array in inverse order as to where it appears in the original array.

Question 7.4 (marks:0.5)

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.

Solution: Just implement the first three **for** loops of COUNTINGSORT, and the return $C[b] - C[a-1]$ if $a > 0$ and $C[b]$ otherwise.

Question 7.5 (marks 0.25)

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

Solution:

unsorted list		sorted by last letter		sorted by middle letter		sorted by first letter
COW		MO**B**		T**A**B		**B**AR
DOG		TA**B**		B**A**R		**B**IG
TUG		DO**G**		C**A**R		**B**OX
ROW		TU**G**		T**A**R		**C**AR
MOB		PI**G**		P**I**G		**C**OW
BOX		BI**G**		B**I**G		**D**OG
TAB	$\rightarrow$	BA**R**	$\rightarrow$	M**O**B	$\rightarrow$	**M**OB
BAR		CA**R**		D**O**G		**P**IG
CAR		TA**R**		C**O**W		**R**OW
TAR		CO**W**		R**O**W		**T**AB
PIG		RO**W**		W**O**W		**T**AR
BIG		WO**W**		B**O**X		**T**UG
WOW		BO**X**		T**U**G		**W**OW

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.

Solution: InsertionSort is stable because elements are only shifted to the right if they are larger than the currently examined element. The considered element is placed in the array straight after an element that is either smaller or equal. Thus original order is preserved in case of ties.

MergeSort is stable because during the Merge procedure equal elements are always chosen first from the Left subarray that contains the elements appearing earlier in the original array, and the subarrays are searched from left to right.

HeapSort is not stable: for example on array $[2, 1, 2]$ the two 2s are returned in inverse order.

QuickSort is not stable: for example on array $[2, 2, 1]$ the two 2s are returned in inverse order.

Question 7.7 (0.25 marks) Implement CountingSort(A, B, k, n) that sorts n numbers between 0 and k and RadixSort(A, d, n) that sorts an English Dictionary with n words of d letters each.