CSCI2100C 2019-20: Solution 3

This assignment is due at 11:59:59pm, 12th April 2020.

- **Q1. [14 marks]** AVL-Tree.
  - **(i). [10 marks]** Given an empty AVL-Tree, show the AVL-Tree after inserting 4, 8, 9, 2, 3, 7, 6 in order and each insertion operation step by step. (Refer to CSCI2100C-Lecture13 Pages 20-21)

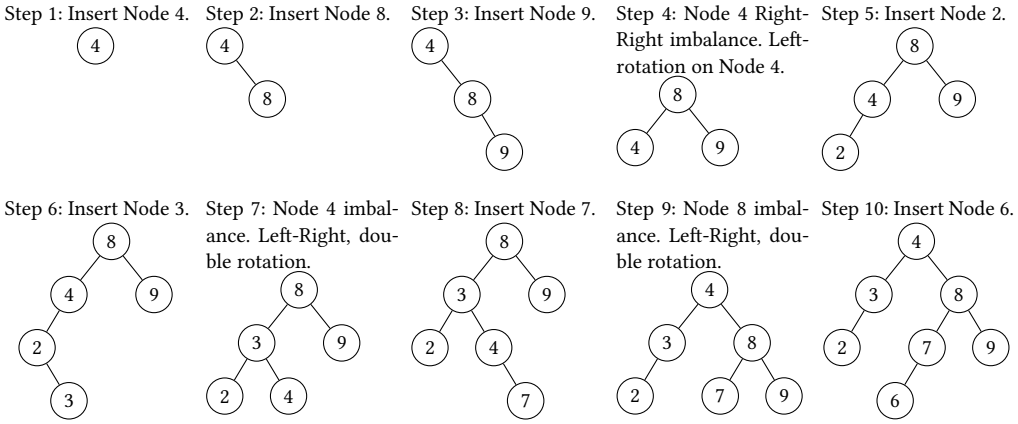

Figure 1. Solution of Q1(i)

  - **(ii). [4 marks]** Given an AVL-Tree as shown in the left most figure, show how to delete the node 2 in this AVL-Tree step by step. (Refer to CSCI2100C-Lecture13 Pages 24-27)

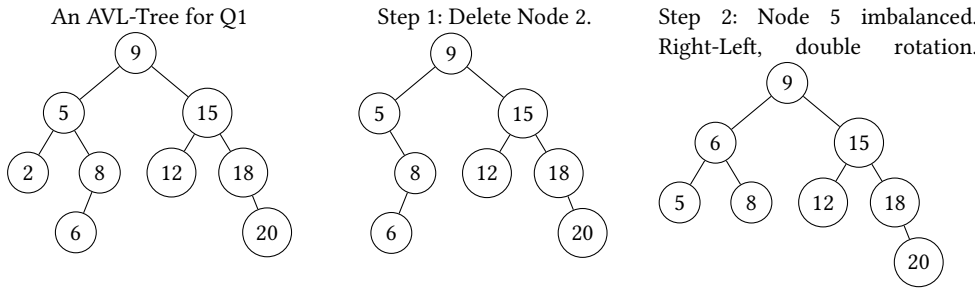

Figure 2. Original Figure and Solution of Q1(ii)

- **Q2. [14 marks]** Treap.

– **(i). [10 marks]** Draw the treap after inserting nodes 2, 3, 5, 7, 11 in order, assuming that their respective priorities are 6, 5, 7, 3, 8. Show each insertion operation step by step. (Refer to CSCI2100C-Lecture14 Pages 14-15)

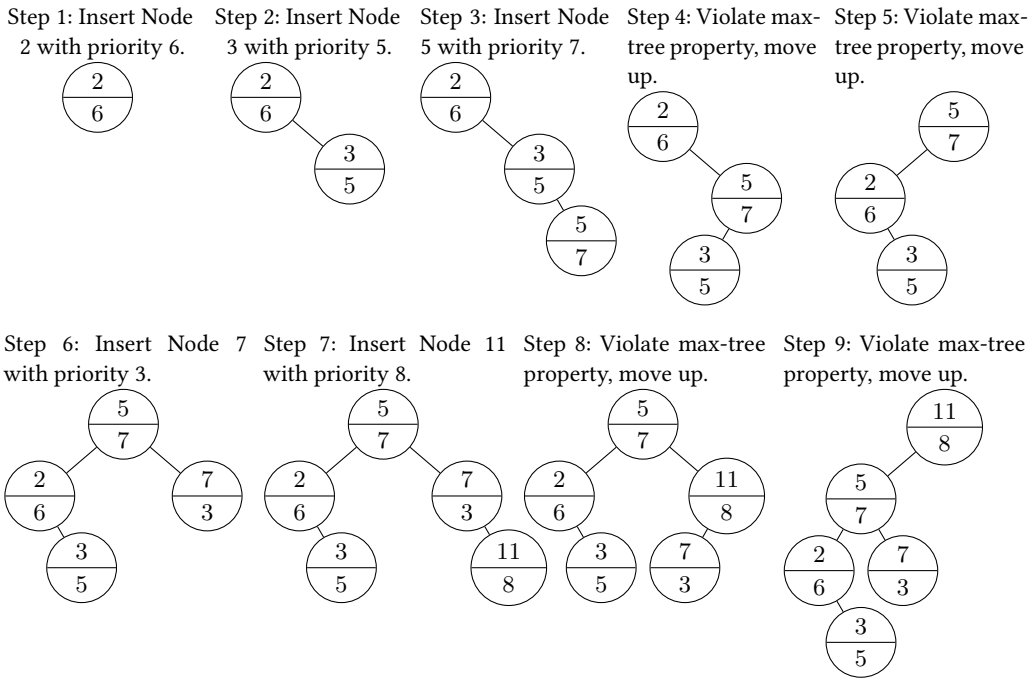

Figure 3. Solution of Q2(i)

– **(ii). [4 marks]** Given a treap as shown in Figure 4, where the value above the line is the key and the one below the line is the priority in a node, show how to delete node 7 (the node whose key is 7) in this treap step by step. (Refer to CSCI2100C-Lecture14 Pages 17-18)

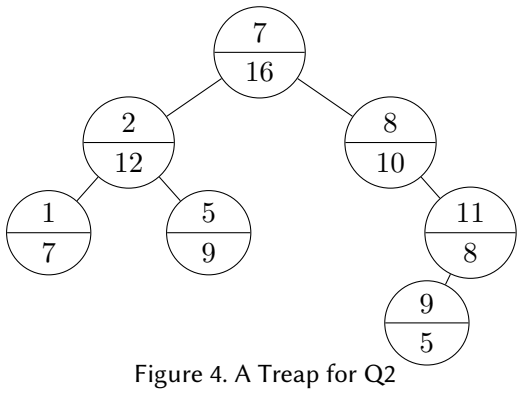

Figure 4. A Treap for Q2

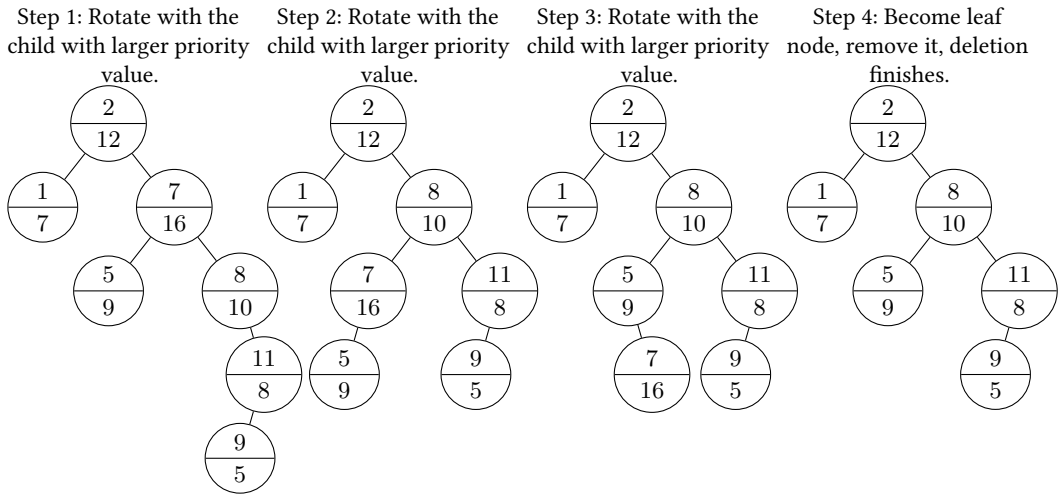

Figure 5. Solution of Q2(ii)

- **Q3. [12 marks]** Sort an array $A[1 \cdots 8] = [4, 6, 3, 8, 5, 2, 7, 9]$ in ascending order by heap sort.
  - **(i). [6 marks]** Show the contents of A in the heap adjust process to make it a max-heap step by step. (Refer to CSCI2100C-Lecture17 Pages 6-8)

    Step 1: Adjust node with id 4. | 4 | 6 | 3 | 9 | 5 | 2 | 7 | 8 |
    Step 2: Adjust node with id 3. | 4 | 6 | 7 | 9 | 5 | 2 | 3 | 8 |
    Step 3: Adjust node with id 2. | 4 | 9 | 7 | 8 | 5 | 2 | 3 | 6 |
    Step 4: Adjust node with id 1. | 9 | 8 | 7 | 6 | 5 | 2 | 3 | 4 |
  - **(ii). [6 marks]** Using the max-heap of part (i), show the contents of A in the sorting process of swaping elements in the array step by step. (Refer to CSCI2100C-Lecture17 Page 9)

    Step 1: Swap node id 8 and node id 1; reduce heap size by 1. | 4 | 8 | 7 | 6 | 5 | 2 | 3 | **9** |
    Step 2: Adjust. | 8 | 6 | 7 | 4 | 5 | 2 | 3 | **9** |
    Step 3: Swap node id 7 and node id 1; reduce heap size by 1. | 3 | 6 | 7 | 4 | 5 | 2 | **8** | **9** |
    Step 4: Adjust. | 7 | 6 | 3 | 4 | 5 | 2 | **8** | **9** |
    Step 5: Swap node id 6 and node id 1; reduce heap size by 1. | 2 | 6 | 3 | 4 | 5 | **7** | **8** | **9** |
    Step 6: Adjust. | 6 | 5 | 3 | 4 | 2 | **7** | **8** | **9** |
    Step 7: Swap node id 5 and node id 1; reduce heap size by 1. | 2 | 5 | 3 | 4 | **6** | **7** | **8** | **9** |
    Step 8: Adjust. | 5 | 4 | 3 | 2 | **6** | **7** | **8** | **9** |
    Step 9: Swap node id 4 and node id 1; reduce heap size by 1. | 2 | 4 | 3 | **5** | **6** | **7** | **8** | **9** |
    Step 10: Adjust. | 4 | 2 | 3 | **5** | **6** | **7** | **8** | **9** |
    Step 11: Swap node id 3 and node id 1; reduce heap size by 1. | 3 | 2 | **4** | **5** | **6** | **7** | **8** | **9** |

Step 12: Swap node id 2 and node id 1; reduce heap size by 1.

2	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Step 13: The heap size now is 1. Sort finishes.

2	**3**	**4**	**5**	**6**	**7**	**8**	**9**

- **Q4. [22 marks]** Answer the following questions about merge sort and quicksort.
  - **(i). [8 marks]** Given an unsorted sequence $[4, 1, 6, 3, 2, 9, 5, 7]$, please fill in the diagram of Figure 12 step by step to show how to sort the sequence with merge sort. (Some contents of the diagram have been given in Figure 12.) (Refer to CSCI2100C-Lecture15-16 Pages 16-25)

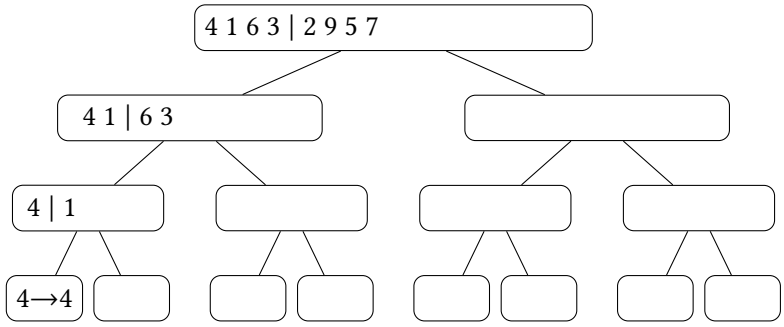

Figure 6. A Diagram of Mergesort for Q4

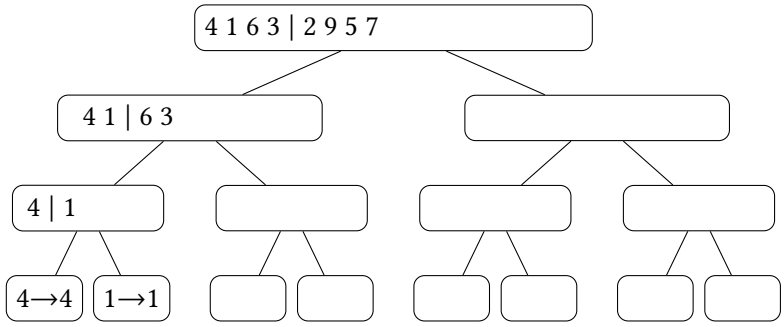

Figure 7. Diagram 1 of Solution of Q4(i)

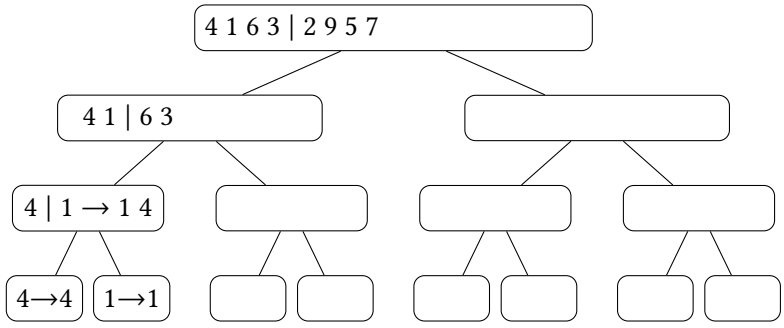

Figure 8. Diagram 2 of Solution of Q4(i)

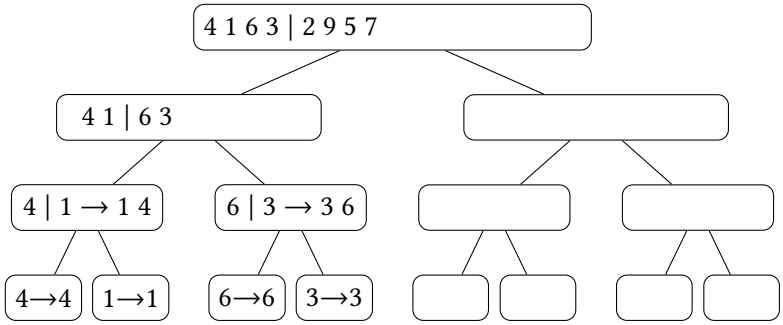

Figure 9. Diagram 3 of Solution of Q4(i)

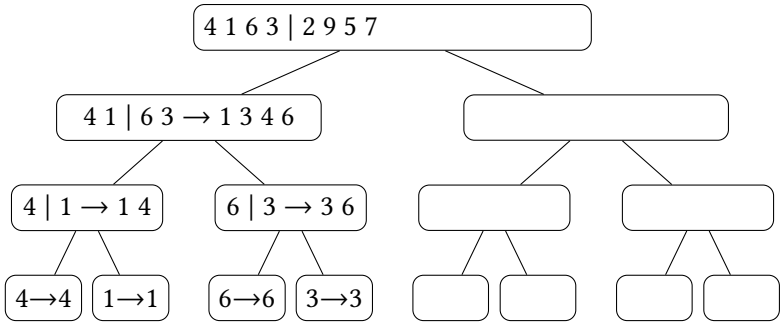

Figure 10. Diagram 4 of Solution of Q4(i)

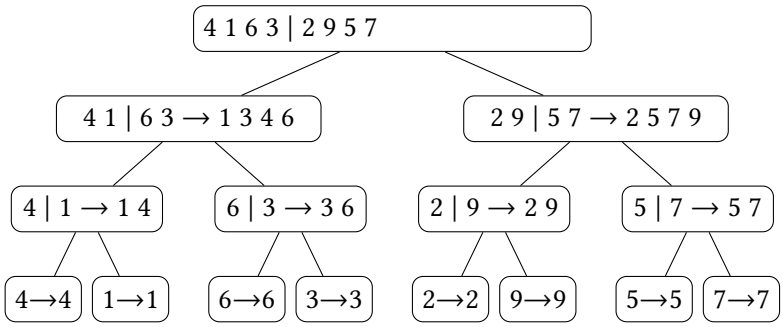

Figure 11. Diagram 5 of Solution of Q4(i)

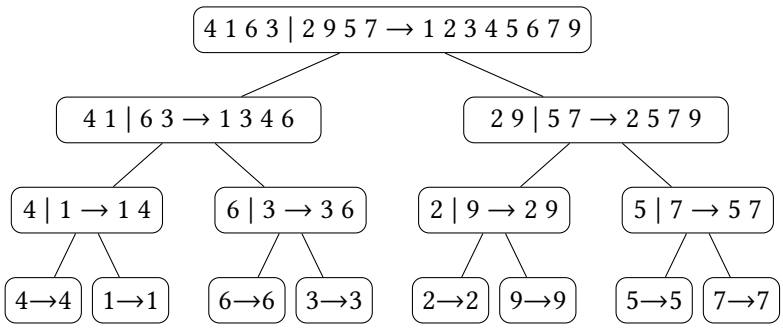

Figure 12. Diagram 6 of Solution of Q4(i)

– **(ii). [6 marks]** We use a **temparray** (Refer to CSCI2100C-Lecture15-16 Pages 26-28) to store the merge result. Given an array $A[0 \cdots 7] = [1, 3, 6, 8, 2, 5, 7, 9]$, during the invocation of **merge**(A, 0, 3, 7), show the contents of **temparray** and indicate the values of **left** and **secondleft** pointing the original array A step by step.

original array A:

1	3	6	8	2	5	7	9

temparray:

1								left = 1 and secondleft = 4.
1	**2**							left = 1 and secondleft = 5.
1	**2**	3						left = 2 and secondleft = 5.
1	**2**	3	**5**					left = 2 and secondleft = 6.
1	**2**	3	**5**	6				left = 3 and secondleft = 6.
1	**2**	3	**5**	6	**7**			left = 3 and secondleft = 7.
1	**2**	3	**5**	6	**7**	8		left = 4 and secondleft = 7.
1	**2**	3	**5**	6	**7**	8	**9**	left = 4 and secondleft = 8.

– **(iii). [8 marks]** Let $A[0 \cdots 7] = [6, 3, 5, 2, 7, 4, 1, 8]$. Assume that we call **quicksort**(A, 0, 7) (Refer to CSCI2100C-Lecture15-16 Pages 34-36) and the **pivot** (i.e., the position of the randomly chosen element in A) chosen randomly is 2. Show what happens in the contents of A and indicate the values of **nextsmallpos** during the invocation of **partition**(A, 0, 7, 2) step by step.

6	3	8	2	7	4	1	5	nextsmallpos = 0 (and j = 0).
6	**3**	8	2	7	4	1	5	nextsmallpos = 0 (and j = 1).
3	**6**	8	2	7	4	1	5	nextsmallpos = 1 (and j = 1).
3	6	**8**	2	7	4	1	5	nextsmallpos = 1 (and j = 2).
3	6	8	**2**	7	4	1	5	nextsmallpos = 1 (and j = 3).
3	2	8	**6**	7	4	1	5	nextsmallpos = 2 (and j = 3).
3	2	8	6	**7**	4	1	5	nextsmallpos = 2 (and j = 4).
3	2	8	6	7	**4**	1	5	nextsmallpos = 2 (and j = 5).
3	2	4	6	7	**8**	1	5	nextsmallpos = 3 (and j = 5).
3	2	4	6	7	8	**1**	5	nextsmallpos = 3 (and j = 6).
3	2	4	1	7	8	**6**	5	nextsmallpos = 4 (and j = 6).
3	2	4	1	5	8	6	7	Swap the nextsmallpos data and the last data.

Return nextsmallpos = 4, which is the position of the pivotVal.

■ **Q5. [10 marks]** Show how to use the counting sort to sort an array $A[0 \cdots 9] = [6, 1, 2, 1, 3, 4, 6, 1, 3, 2]$ with $U = 6$ (i.e., the keys are integers within the range $[1, 6]$) step by step. (Refer to CSCI2100C-Lecture17 Pages 27-32)

Step 1: Allocate an array *counter* of size $U + 1$. All values are initialized to be zero. Scan the whole array, from 0 to $n - 1$. Increment $counter[A[i]]$ for i from 0 to $n - 1$.

counter

3	2	2	1	0	2
1	2	3	4	5	6

Step 2: Use the *counter* array to get the last position of a key $x \in U$, $counter[i+1] = counter[i] + counter[i+1]$ for i from 1 to U.

counter

1	2	3	4	5	6
3	5	7	8	8	10

Step 3: Use another array B to store the sorted elements. Rescan array A from end to beginning and sort the elements using the information in step 2.

$i = 9$ Get $counter[A[i]] = 5$. Store $A[i]$ at $B[counter[A[i]]-1] = B[4]$. Decrement $counter[A[i]]$ by 1.

B

0	1	2	3	4	5	6	7	8	9
				2					

counter

1	2	3	4	5	6
3	**4**	7	8	8	10

$i = 8$ Get $counter[A[i]] = 7$. Store $A[i]$ at $B[counter[A[i]]-1] = B[6]$. Decrement $counter[A[i]]$ by 1.

B

0	1	2	3	4	5	6	7	8	9
				2		**3**			

counter

1	2	3	4	5	6
3	4	**6**	8	8	10

$i = 7$ Get $counter[A[i]] = 3$. Store $A[i]$ at $B[counter[A[i]]-1] = B[2]$. Decrement $counter[A[i]]$ by 1.

B

0	1	2	3	4	5	6	7	8	9
		1		2		3			

counter

1	2	3	4	5	6
2	4	6	8	8	10

$i = 6$ Get $counter[A[i]] = 10$. Store $A[i]$ at $B[counter[A[i]] - 1] = B[9]$. Decrement $counter[A[i]]$ by 1.

B

0	1	2	3	4	5	6	7	8	9
		1		2		3			**6**

counter

1	2	3	4	5	6
2	4	6	8	8	**9**

$i = 5$ Get $counter[A[i]] = 8$. Store $A[i]$ at $B[counter[A[i]]-1] = B[7]$. Decrement $counter[A[i]]$ by 1.

B

0	1	2	3	4	5	6	7	8	9
		1		2		3	**4**		6

counter

1	2	3	4	5	6
2	4	6	**7**	8	9

$i = 4$ Get $counter[A[i]] = 6$. Store $A[i]$ at $B[counter[A[i]]-1] = B[5]$. Decrement $counter[A[i]]$ by 1.

B

0	1	2	3	4	5	6	7	8	9
		1		2	**3**	3	4		6

counter

1	2	3	4	5	6
2	4	**5**	7	8	9

$i = 3$ Get $counter[A[i]] = 2$. Store $A[i]$ at $B[counter[A[i]]-1] = B[1]$. Decrement $counter[A[i]]$ by 1.

B

0	1	2	3	4	5	6	7	8	9
	1	1		2	3	3	4		6

counter

1	2	3	4	5	6
1	4	5	7	8	9

$i = 2$ Get $counter[A[i]] = 4$. Store $A[i]$ at $B[counter[A[i]]-1] = B[3]$. Decrement $counter[A[i]]$ by 1.

B

0	1	2	3	4	5	6	7	8	9
	1	1	**2**	2	3	3	4		6

counter

1	2	3	4	5	6
1	**3**	5	7	8	9

$i = 1$ Get $counter[A[i]] = 1$. Store $A[i]$ at $B[counter[A[i]]-1] = B[0]$. Decrement $counter[A[i]]$ by 1.

B

0	1	2	3	4	5	6	7	8	9
1	1	1	2	2	3	3	4		6

counter

1	2	3	4	5	6
0	3	5	7	8	9

$i = 0$ Get $counter[A[i]] = 9$. Store $A[i]$ at $B[counter[A[i]]-1] = B[8]$. Decrement $counter[A[i]]$ by 1.

B

1	1	1	2	2	3	3	4	**6**	6
0	1	2	3	4	5	6	7	8	9

$counter$

0	3	5	7	8	**8**
1	2	3	4	5	6

- **Q6. [6 marks]** Based on the idea of counting sort, describe an algorithm that, given n integers in the range $[1, k]$, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a, b]$ (a and b are integers in the range $[1, k]$) in $O(1)$ time. Your algorithm should use $O(n + k)$ preprocessing time. Justify your answer.

  The algorithm will begin by preprocessing exactly as counting sort does in Step 2 in Q5, so that $counter[i]$ contains the number of elements less than or equal to i in the array. When querying about how many integers fall into a range $[a, b]$, simply compute $counter[b] - counter[a-1]$ for $a > 1$. If $a = 1$, the result is $counter[b]$. This takes $O(1)$ times and yields the desired output.

- **Q7. [14 marks]** Radix Sort. (Refer to CSCI2100C-Lecture18 Page 5 and Page 8)
  - **(i). [8 marks]** Show the sorting results after sorting on each digit with radix sort for an array $A[0 \cdots 6] = [4650, 7391, 5479, 3152, 2465, 3918, 9830]$.

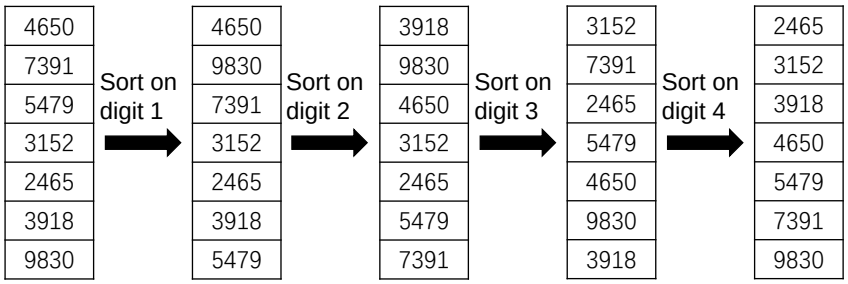

Figure 13. Solution of Q7(i)

  - **(ii). [6 marks]** Show how to sort n integers in the range $[1, n^3 - 1]$ in $O(n)$ time. Justify your answer.

    First run through the list of integers and convert each one to base n, then radix sort them. Each number will have at most $\log_n n^3 = 3 \log_n n = 3$ digits so there will only need to be 3 passes. For each pass, there are n possible values which can be taken on, so we can use counting sort to sort each digit in $O(n)$ time.

- **Q8. [8 marks]** You are given n distinct integers in an array **arr**. For finding the k-th smallest element in this array, there is a random algorithm shown as below. Based on this algorithm, answer the following questions.

```
Algorithm kthSmallest(arr, left, right, k)
1:  if left == right
2:      return arr[left]
```

```
3:  pivot = random(left, right)
4:  pivotnewpos = partition(arr, left, right, pivot)
5:  while pivotnewpos ∉ [left+1/3·(right-left+1), left + 2/3·(right-left+1)]
6:     pivot = random(left, right)
7:     pivotnewpos = partition(arr, left, right, pivot)
8:  if pivotnewpos-left == k-1
9:     return arr[pivotnewpos]
10: elseif pivotnewpos-left > k-1
11:    return kthSmallest(arr, left, pivotnewpos-1, k)
12: else
13:    return kthSmallest(arr, pivotnewpos+1, right, (k-1)-(pivotnewpos-left))
```

– **(i). [4 marks]** During an invocation of **kthSmallest(arr, left, right, k)**, we assume that the size of the subarray **arr[left...right]**, i.e., **right-left+1**, is an integer n that is multiple of 3. For this invocation, calculate the expected number of the basic operation **random(left, right)** executed from Lines 1-7. Besides, what is the expected running cost for Lines 1-7 in terms of Big-Oh? Justify your answer. (Refer to CSCI2100C-Lecture15-16 Pages 34-36 for functions of **random**, **partition** and definitions of the **pivot** and **pivotnewpos**.

There are $n/3 + 1$ positions in the range [left+$\frac{1}{3}$·(right-left+1), left + $\frac{2}{3}$·(right-left+1)]. We define the number of the basic operation **random(left, right)** executed from Lines 1-7 as a random variable X. Since with a probability $\frac{n/3+1}{n}$, **pivotnewpos** $\in$ [left+$\frac{1}{3}$·(right-left+1), left + $\frac{2}{3}$·(right-left+1)] and we jump out of the while loop, X satisfies the geometric distribution and the probability of success $p = \frac{n/3+1}{n}$. According to the property of the geometric distribution, $\mathbf{E}[X] = 1/p$ so we need to use the basic operation **random(left, right)** $\frac{n}{n/3+1}$ times in expectation. For an array of n integers, each **random** function takes $O(1)$ time and each **partition** function takes $n = O(n)$ time. Since $\frac{n}{n/3+1} = O(1)$, the expected running cost for Lines 1-7 in terms of Big-Oh is $n \cdot \frac{n}{n/3+1} = O(n)$ by using the product property.

– **(ii). [4 marks]** Let $T(n)$ be the expected number of comparisons in the above algorithm for finding the k-th smallest element in an array of size n. Derive the recurrence of $T(n)$ and analyze the asymptoic complexity of $T(n)$ in terms of Big-Oh. Justify your answer. (Hint. Consider the largest size of the subarray **arr[left, pivotnewpos-1]** (or **arr[pivotnewpos+1, right]**) when we do the recursion in Line 11 (or Line 13))

When we do the recursion in Line 11 (or Line 13), there are at most $\lceil 2n/3 \rceil$ elements left in the subarray. So we have the recurrence of $T(n)$:

$$T(1) \leq O(1)$$
$$T(n) \leq O(n) + T(\lceil 2n/3 \rceil).$$

Using the master theorem, we have $a = 1$, $b = 3/2$, $\lambda = 1$. Since $\log_b a < \lambda$, the asymptoic complexity of $T(n)$ in terms of Big-Oh is $O(n)$.