Assignment 8

Answer the following questions.

1. Is the bucket-sort algorithm in-place? Why or why not?

   No. It's not an in-place algorithm. We are using another bucket array to sort the elements not in the same array. But it's stable sort algorithm.

2. Describe an O(n) algorithm that does the following:

   Given an input array of *n* integers lying in the range 0 .. 3n - 1, the algorithm outputs the first integer that occurs in the array only once. (You may assume that each input array contains at least one number that has no duplicates in the array.) Explain why your algorithm has an O(n) running time.

   Example: If the input array is [1, 2, 4, 9, 3, 2, 1, 4, 5], then the return value is 9 since 9 is the first integer that occurs in the array only once.

Solution: Let *A* denote the initial input array of integers. We use an auxiliary array T as a tracking array

```
T ← new array(3n)  //initialized with 0s
for i ← 0 to n − 1 do //scan A, increment T at A[i] each i
     x ← A[i]
     T[x] ← T[x] + 1
for i ← 0 to n-1 do
     x ← A[i]
     if T[x] = 1 then
          return x
```

Running Time: Two O(n) loops -> total running time is O(n)

3. Compare heaps vs array to implement Priority queue in terms of performance of PQ operations.

   Priority Queue can implement using sorted arrays or unsorted array.

   Unsorted Array Performance - LL

   - insertItem takes ***O***(1) time since we can insert the item at the end of the sequence
   - removeMin, minKey, and minValue take ***O***(***n***) time since we have to traverse the entire sequence to find the smallest key

Sorted Array Performance-LL

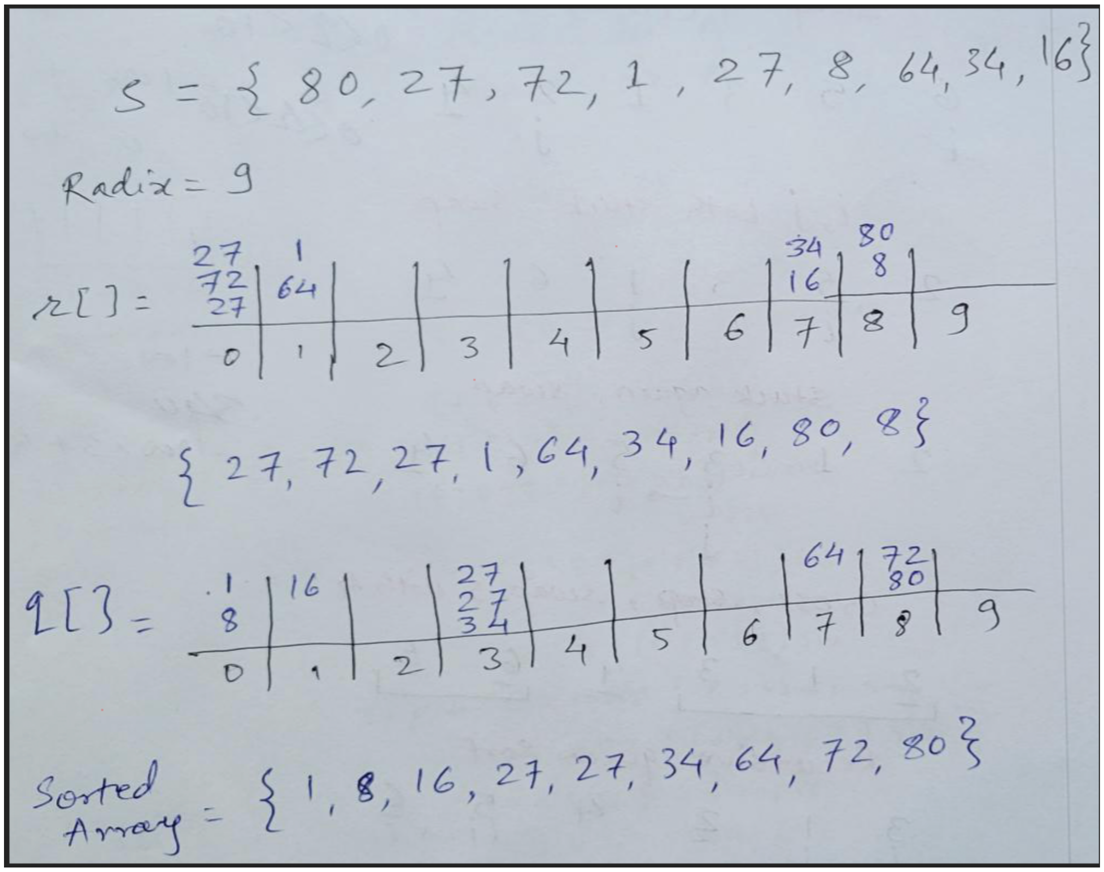
- insertItem takes $O(n)$ time since we have to find the place where to insert the item
- removeMin, minKey, and minValue take $O(1)$ time since the smallest key is at the beginning of the sequence

Heap Performance

- methods insertItem and removeMin take $O(\log n)$ time
- methods minKey, and minElement take $O(1)$ time

4. Carry out the steps of RadixSort to sort the following array:
   {80, 27, 72, 1, 27, 8, 64, 34, 16}.

   Use 9 as your radix.



S = {80, 27, 72, 1, 27, 8, 64, 34, 16}

Radix = 9

r[] =

0	1	2	3	4	5	6	7	8	9
27, 72, 27	1, 64						34, 16	80, 8	

{27, 72, 27, 1, 64, 34, 16, 80, 8}

q[] =

0	1	2	3	4	5	6	7	8	9
1, 8	16		27, 27, 34				64	72, 80	

Sorted Array = {1, 8, 16, 27, 27, 34, 64, 72, 80}

5. Suppose we are given two sequences A and B of *n* elements, possibly containing duplicates, on which a total order relation is defined (i.e., has a comparator). Using a Priority Queue design an efficient pseudo-code algorithm for determining if A and B contain the same set of elements (possibly in different orders and possibly containing duplicates). What is the running time of this method?

```
Algorithm arePQEuals(A, B, C)
Input: Sequence of A and B of n elements and the Comparator C
Output: returns boolean value to show the A and B are equal or not
        PQ1 ← new priority queue using C
        PQ2 ← new priority queue using C
        while A.size() > 0 do
            e ← A.remove (A.first())
            PQ1.insertItem(e)
        while B.size() > 0 do
            e ← B.remove (B.first())
            PQ2.insertItem(e)
  # Dequeue elements from both priority queues
  while not PQ1.isEmpty() and not PQ2.isEmpty() do
    elementA = PQ1.removeMin()
    elementB = PQ2. removeMin()

    if elementA != elementB then
      return false
  return PQ1.isEmpty() and PQ2.isEmpty()
```