Algorithms -1
Tutorial 3
Solution Sketch

1. Show all steps in sorting the array A = (38 81 22 48 13 69 93 14 45 58 79 72) using Quicksort always choosing the pivot element to be the element in position (left+right)/2.

   Routine.

2. Suppose you have to sort the integers 1342, 233, 145, 8757, 23, 1888, 2545, 3245, 191 using radix sort. Show the sorted sequences after each digit is sorted.

   Routine.

3. How can you use the partitioning idea of quicksort to give an algorithm that finds the median element of an array of n integers? What is the time complexity of your algorithm?

   If n = 1 return the element as median.
   Partition the array around a pivot p. Let the position of the pivot be j after partitioning.
   If j = n/2, p is the median
   Else if j > n/2, recursive search for median in left subarray
   Else recursively search for median in right subarray

   Time complexity: $O(n^2)$ worst case, O(nlgn) average case

   (Adjust n/2 properly for odd-even n).

4. An array A of n integers consists of only the numbers 0, 1, and 2. Design an O(n) time algorithm to sort the array using (i) O(n) extra space, (ii) O(1) extra space.

   Use counting sort for the case with O(n) extra space
   For the case with O(1) extra space, choose an element with 1 as pivot, and partition, with all elements less than or equal to 1 put on the left. This puts all 2's together at the end of the array. Now use partition again to sort the left subarray with 0 and 1 only, choosing 1 as pivot again and putting all values less than pivot to the left. (Can be done in one pass only also).

5. Given an array A with n elements, two elements A[i] and A[j] form an inversion if A[i] > A[j] and i < j. Count the number of inversions in the array in O(nlogn) time.

   Mergesort the array, but with a modified merge to count inversions as you mege. Suppose you know the number of inversions in the left subarray (say IL) and in the right subarray (say IR). The base case values for IL and IR are 0 for n=1. Total number of

inversions will be IL + IR + IM, where IM is the number of inversions found during merge between the two subarrays. During merge, if you hit an A[i] > A[j], where i is the pointer in the first (left) subarray, and j is the pointer in the second (right) subarray, you have hit an inversion (as i is less than j because i is in the left subarray and j is in the right). But you have hit more than that. Think how many inversions you can add (note that all values in the left subarray from A[i+1] to A[mid] are higher than A[i], and i+1,i+2,...mid are all less than j). Think what should be the value of IM. Then IL+IR+IM becomes the IL (or IR depending on which subarray is being merged) for the next level merge.

6. Given n integers in the range 0 to k, design an algorithm that pre-processes its input in O(n + k) time, then answers a query about how many of the n integers fall into a range [a, b] in O(1) time. You can use O(k) additional space.

   Use counting sort. Create an array cnt of size k+1, with all values initialized to 0. For each integer x in the input, we increment cnt[*x*] by 1. After this, we update cnt[i] = cnt[i − 1] + cnt[i] for $1 \leq i \leq k$ (in order from 1 to k). So, now cnt[i] denotes yje number of integers in the input which are less than or equal to i. So, given a query [a, b] there are two cases:

   - a = 0, answer is cnt[b]
   - a > 0, answer is cnt[b] − cnt[a − 1]

7. You are given n intervals in the form of [l,r] where l and r are integers. An interval is said to be active at i if $l \leq i \leq r$. Design an O(nlogn) time algorithm to find the maximum number of intervals that are active at any integer.

The key idea is that it is sufficient to consider only the end points. So, if we take the maximum number of active intervals over all end points, that will also be equal to the maximum number over all other points. Let's say we have a list of pair of integers P. For each interval [l, r], we add (l, START) and (r, END) to P. START and END can be any distinct integers. We sort P using the comparison: $(x, y) < (x^i, y^i)$ iff $x < x^i$ or ($x = x^i$ and $y =$ START and $y^i =$ END). Assuming P is already sorted, here is the pseudo code:

```
answer = 0, activeCount = 0
for all (x, y) in P do
    if y = START then
        activeCount := activeCount + 1
        if answer < activeCount then
          answer = activeCount
    else
        activeCount := activeCount − 1
```