1. Exercise 7.4-5 in CLRS Textbook.

Solution:

If we are only doing quick-sort until the problem size becomes ≤ k, then, we will have to take $\lg(\frac{n}{k})$ steps, since, as in the original analysis of randomized quick sort, we expect there to be lg(n) levels to the recursion tree. Since we then just call quicksort on the entire array, we know that each element is within k of its final position. This means that an insertion sort will take the shifting of at most k elements for every element that needed to change position. This gets us the running time described.

In theory, we should pick k to minimize this expression, that is, taking a derivative with respect to k, we want it to be evaluating to zero. So, $n - \frac{n}{k} = 0$, so $k \sim \frac{1}{n^2}$. The constant of proportionality will depend on the relative size of the constants in the nk term and in the $n \lg(\frac{n}{k})$ term. In practice, we would try it with a large number of input sizes for various values of k, because there are gritty properties of the machine not considered here such as cache line size.

2. Problem 7-2 in CLRS Textbook.

Solution:

a. Since all elements are the same, the initial random choice of index and swap change nothing. Thus, randomized quicksort's running time will be the same as that of quicksort. Since all elements are equal, PARTITION(A, P, r) will always return r − 1. This is worst-case partitioning, so the runtime is $\Theta(n^2)$.

```
PARTITION'(A,p,r)
x = A[r]
exchange A[r] with A[p]
i = p − 1
k = p
for j = p + 1 to r − 1 do
    if A[j] < x then
        i = i + 1
        k = i + 2
        exchange A[i] with A[j]
        exchange A[k] with A[j]
    end if
    if A[j] = x then
        k = k + 1
```

```
            exchange A[k] with A[j]
        end if
    end for
    exchange A[i + 1] with A[r]
    return i + 1 and k + 1


RANDOMIZED-PARTITION'
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION' (A,p,r)
```

c.

```
RANDOMIZED-PARTITION'
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION' (A,p,r)
```

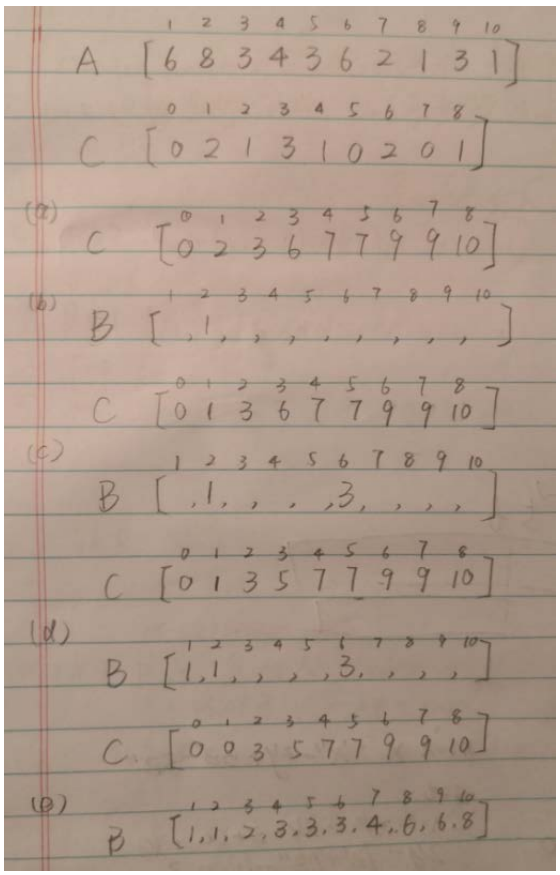
d.Let d be the number of distinct elements in A. The running time is dominated by the time spent in the PARTITION procedure, and there can be at most d calls to PARTITION. If X is the number of comparisons performed in line 4 of PARTITION over the entire execution of QUICKSORT', then the running time is O(d + X). It remains true that each pair of elements is compared at most once. If z_i is the i^{th} smallest element, we need to compute the probability that z_i is compared to z_j . This time, once a pivot x is chosen with $z_i \leqslant x \leqslant z_j$, we know that z_i and z_j cannot be compared at any subsequent time. This is where the analysis differs, because there could be many elements of the array equal to z_i or z_j , so the probability that z_i and z_j are compared decreases. However, the expected percentage of distinct elements in a random array tends to $1 - \frac{1}{e}$, so asymptotically the expected number of comparisons is the same.

```
QUICKSORT' (A,p,r)
if p < r then
    (q, t) = RANDOMIZED − P ART IT IONO
    QUICKSORT' (A,p,q-1)
    QUICKSORT' (A,t+1,r)
end if
```

3. Similar to Figure 8.2, illustrate the operation of COUNTING-SORT on

$$A = [6, 8, 3, 4, 3, 6, 2, 1, 3, 1]$$

Solution:



4. Exercise 8.2-3 in CLRS Textbook.

Solution:

The algorithm still works correctly. The order that elements are taken out of C and put into B doesn't affect the placement of elements with the same key. It will still fill the interval (C[k − 1], C[k]] with elements of key k. The question of whether it is stable or not is not well phrased. In order for stability to make sense, we would need to be sorting items which have information other than their key, and the sort as written is just for integers, which don't. We could think of extending this algorithm by placing the elements of A into a collection of elements for each cell in array C. Then, if we use a FIFO collection, the modification of line 10 will make it stable, if we use LILO, it will be anti-stable.

5. Exercise 8.3-3 in CLRS Textbook.

Solution:

After sorting on digit i, we will show that if we restrict to just the last i digits, the list is in sorted order. This is trivial for i = 1, because it is just claiming that the digits we just sorted were in sorted order. Now, suppose it' strue for i − 1, we show it for i. Suppose there are two elements, who, when restricted to the i last digits, are not in sorted order after the i' th step. Then, we must have that they have the same i' th digit because otherwise the sort of digit i would put them in the right order. Since they have the same first digit, their relative order is determined by their restrictions to their last i − 1

digits. However, these were placed in the correct order by the i − 1’ st step. Since the sort on the i’ th digit was stable, their relative order is unchanged from the previous step. This means that they are in the correct order still. We use stability to show that being in the correct order prior to doing the sort is preserved.

6. Problem 9-1 in CLRS Textbook.

Solution:

a. The running time of sorting the numbers is $O(n \lg n)$, and the running time of listing the i largest is $O(i)$. Therefore, the total running time is $O(n \lg n + i)$.

b. The running time of building a max-priority queue (using a heap) from the numbers is $O(n)$, and the running time of each call EXTRACT-MAX is $O(\lg n)$. Therefore, the total running time is $O(n + i \lg n)$.

c. The running time of finding and partitioning around the ith largest number is $O(n)$, and the running time of sorting the i largest numbers is $O(i \lg i)$. Therefore, the total running time is $O(n + i \lg i)$.

7. Exercise 11.2-1 in CLRS Textbook.

Solution:

Under the assumption of simple uniform hashing, we will use linearity of expectation to compute this. Suppose that all the keys are totally ordered $\{k_1, \ldots, k_n\}$. Let Xi be the number of $l > k_i$ so that $h(l) = h(k_i)$. Note, that this is the same thing as $\sum_{j>i} \Pr(h(k_j) = h(k_i)) = \sum_{j>i} \frac{1}{m} = \frac{(n-i)}{m}$. Then, by linearity of expectation, the number of collisions is the sum of the number of collisions for each possible smallest element in the collision. The expected number of collisions is $\sum_{i=1}^{n} \frac{n-i}{m} = \frac{n^2 - \frac{n(n+1)}{2}}{m} = \frac{n^2-n}{2m}$.