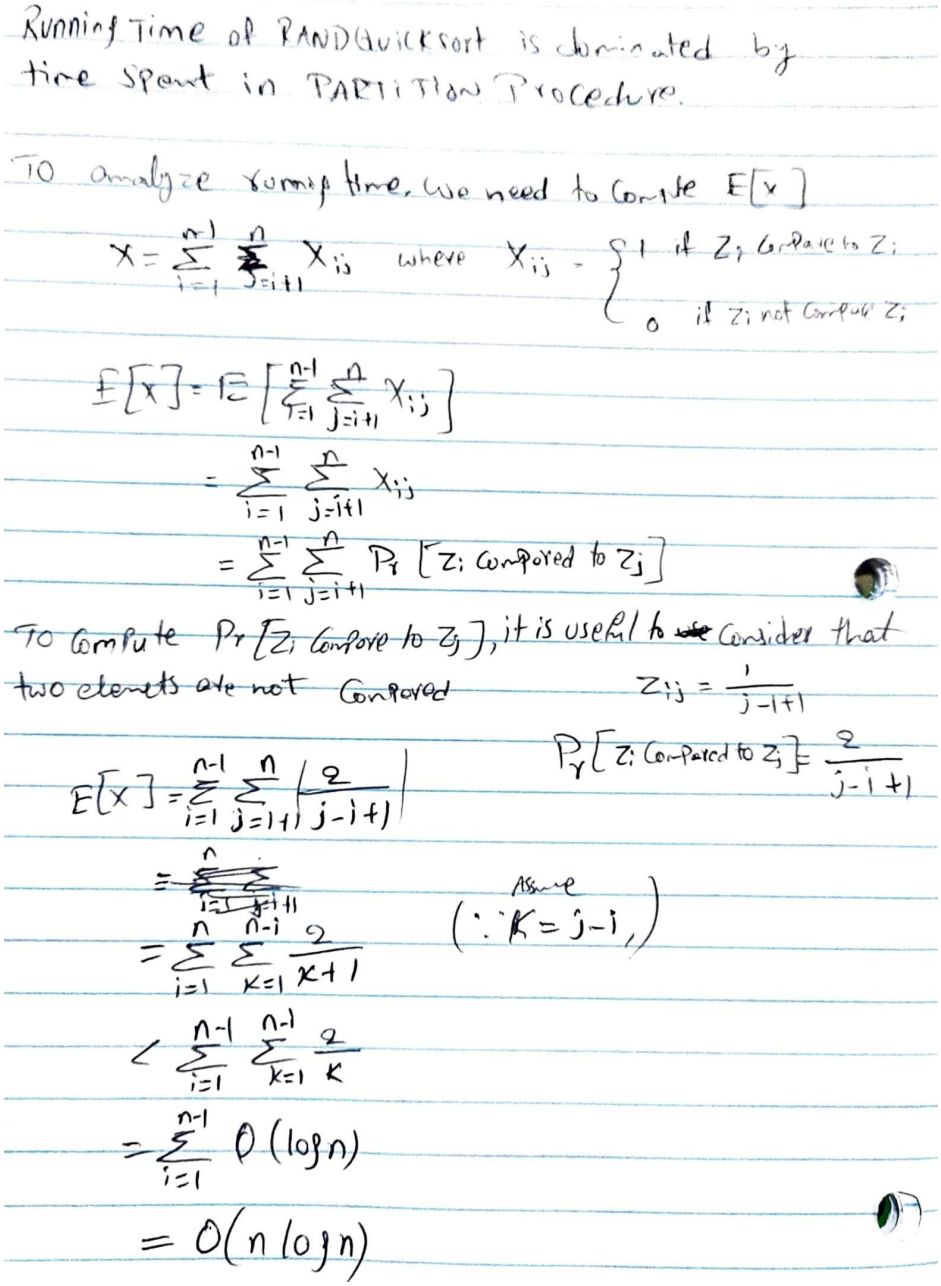
Problem 1 (7.2-3)

When array A has distinct elements and is sorted in decreasing order, this is when mostly it's pivot element is always the smallest element in each partition. The initial subarray has only one element (the smallest), and it is in the correct position. The remaining elements are included in the second subarray. The running time recurrence in this example will be $T(n) = T(n-1) + n$

$$T(n) = \Theta(n^2)$$

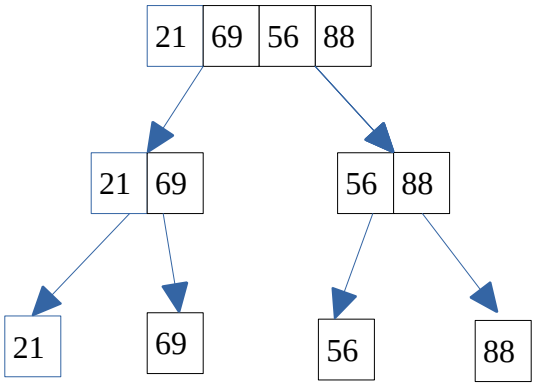
Therefore, The running time for the quick sort is $\Theta(n^2)$ when array A contains distinct elements and is sorted in decreasing order.

Problem 2 (7.4-4)



Problem 3

To draw partial decision tree for merge sort with four distinct integers, Let's assume the four distinct integers to be 21, 69, 56 and 88 in merge sort M.



m1=21; m2=69; m3=56; m4=88

Problem 4 (8.2-4)

By using counting sort algorithm as mentioned in textbook.

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

The method will begin by pre-processing in the same way as COUNTING-SORT does in lines 1 through 9 to pre-process the n integers in sorting order, so that C[i] holds the number of elements in the array that are less than or equal to i. When asked how many numbers fit into a range [a..b], just compute C[b] C[a 1]. This takes O(1) times and produces the required result.

Problem 5 (8.3-4)

By using radix sort in base n, we can sort n integers in the range 0 to n^3-1. Since the numbers are in base n, the range of digits is 1 to n => k = n. Each number has a variable amount of digits depending on the range of numbers. If we consider maximum number of digits is d, then it will make radix sort to run in O(d (n + n)) = O(n). Therefore, we can sort integers in O(n) time.

Problem 6 (8.4-2)

If all of the keys fall into the same bucket and are in reverse order, we must use insertion sort to sort a single bucket with n items in reverse order. This will take $\Theta(n^2)$.To reduce the worst-case running time, we can employ merge sort or heap sort. Insertion sort was chosen because it works well on linked lists, takes the least amount of time, and requires just a constant amount of extra space for small linked lists. If we use another sorting technique, we must convert each list to an array, which may slow the procedure down in practice.