CS360 Assignment 4

7.2-2 What is the running time of QUICKSORT when all the elements of array A have the same value?

Since an array whose elements are all the same value is, by definition, *sorted in ascending order*, QUICKSORT will perform *worst case*. Note that line 4 of PARTITION will always be satisfied, thus all the elements will exchange with themselves leaving the pivot element at the end of the partition and only recursing on the first $n-1$ elements. This procedure will be similar for all other elements, producing $\Theta(n^2)$ behavior *even though no sorting is necessary.*

7.2-3 Show that the running time of QUICKSORT is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

Initially we note that the pivot element will be the *smallest* element in the array. Hence the first call to PARTITION() will cause lines 4-6 (the **if** block) to not execute since the pivot element is less than all the other element in the array. At the conclusion of PARTITION() the pivot will be swapped with the first element in the array, i.e. the *smallest* element will be swapped with the *largest* element. For example, after the first call to PARTITION() the following array will be

$$< 8, 7, 6, 5, 4, 3, 2, 1 > \Rightarrow < 1, 7, 6, 5, 4, 3, 2, 8 >$$

The first recursive call to QUICKSORT() will be empty (since the pivot moved to the first element) and the second recursive call to QUICKSORT() will contain the remaining $n-1$ elements (note they are **NOT** sorted at this point).

The next call to PARTITION() will select the pivot element which is the *largest* one in the array. Hence lines 4-6 (the **if** block) will execute for *every* iteration of the loop since the pivot element is greater than all the other element in the array. However the body of the **if** block will simply exchange each element with itself and then PARTITION() will complete by swapping the pivot element with itself. Hence after the second call to PARTITION() the array above will be

$$< 1, 7, 6, 5, 4, 3, 2, 8 > \Rightarrow < 1, 7, 6, 5, 4, 3, 2, 8 >$$

The first recursive call to QUICKSORT() will now contain elements $A[2..n-1]$ (note they **are** sorted in decreasing order at this point). The second recursive call, however, will be empty. PARTITION() will behave similar to the initial case resulting in the second smallest element being moved to the second location in the array as

$$< 1, 7, 6, 5, 4, 3, 2, 8 > \Rightarrow < 1, 2, 6, 5, 4, 3, 7, 8 >$$

The next pass will not change the array (since the next to last element is the second largest one). This pattern will continue causing one of the two recursive calls to QUICKSORT() to always be empty giving the recursive equation as

$$T(n) = T(n-1) + O(n)$$

which is identical to the worst case behavior when the initial array is sorted in increasing order giving (even though the operation of the algorithm is slightly different)

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

8.1-4 Given a sequence of n elements such that the sequence consists of n/k subsequences of length k such that all elements of the current subsequence are greater than those of the prior subsequence and less than those of the subsequent subsequence, show that $\Omega(n \lg k)$ is a lower bound on the number of comparisons needed to sort all n/k subsequences.

Since each subsequence has k elements, there are $k!$ permutations of each of the n/k subsequences. Hence the total number of permutations of the *entire* sequence is
$\underbrace{(k!)(k!)...(k!)}_{n/k} = (k!)^{n/k}$
These constitute the 2^h leaves of the decision tree $\Rightarrow 2^h \geq (k!)^{n/k} \Rightarrow h \geq \frac{n}{k} \lg(k!)$.

Since $k! = \prod_{i=1}^{k} i$

$$\begin{aligned} k! &= \left(\prod_{i=1}^{k/2} i\right)\left(\prod_{i=\frac{k}{2}+1}^{k} i\right) \\ &> \prod_{i=\frac{k}{2}+1}^{k} i \\ &> \prod_{i=\frac{k}{2}}^{k} \frac{k}{2} = (k/2)^{(k/2)} \end{aligned}$$

$\Rightarrow k! > (k/2)^{(k/2)}$

Therefore

$$\begin{aligned} h &\geq \frac{n}{k}\lg(k!) \\ &> \frac{n}{k}\lg((k/2)^{(k/2)}) \\ &= \frac{n}{k}\frac{k}{2}\lg(k/2) \\ &= \frac{n}{2}\lg(k/2) \end{aligned}$$

It is easy to show $\frac{n}{2}\lg(k/2) = \Omega(n \lg k)$ from the definition by choosing $k_0 = 4$ and $c = \frac{1}{4}$ giving $\frac{1}{4}n \lg k = \frac{n}{2}\lg k^{(1/2)} = \frac{n}{2}\lg\sqrt{k} \leq \frac{n}{2}\lg(k/2)$
$\Rightarrow h = \Omega(n \lg k)$

8.3-4 Show how to sort n integers in the range 0 to $n^3 - 1$ in $O(n)$ time.

First note that we *cannot* use standard counting sort since $k = n^3$ giving a run time of $O(n^3)$. Likewise we also cannot use standard radix sort since a decimal number has $d = \log_{10} n = c \lg n$ digits and thus would have a run time of $O(d(n + k)) = O((c \lg n)(n + 10)) = O(n \lg n)$. Thus we need to find a radix representation that keeps the number of digits *constant* yet has the range of *each digit* no worse than linear. Consider radix-n notation (each digit in the range 0 to n) such that we represent each value v in the form

$$v = d_0 n^0 + d_1 n^1 + d_2 n^2 ... + d_k n^k$$

Thus the number of digits for numbers in the range 0 to $n^3 - 1$ is $d = log_n n^3 = 3log_n n = 3$, i.e. is *constant*. Since each digit is in the range 0 to n, $k = n$. Then we can apply radix sort giving a run time of $O(d(n+k)) = O(3(n+n)) = O(6n) = O(n)$.