HW 3

Daniel Sawyer

October 24, 2018

Q1.

a.) Ascending Order:
Base Case:
$p = 1, r = 2$ and array A is sorted in ascending order. $P(k) = 1, if A[k] \leq A[r]; 0, otherwise.$ Pivot $q = P(1) + P(2) = 1 + 1 = 2$, therefore q = r, which leaves two arrays, one from p to r and another empty as expected.
Proof by induction:
Assume array A is sorted in ascending order. So for any value k where $p \leq k \leq r$, $A[k] \leq A[r]$ for all k.
The pivot point $q = \sum_{k=p}^{r} P(k)$ where $P(k) = \{1$, if A[k] $\leq$ A[r]; 0, otherwise$\}$
Since A[k] will always be less than or equal to A[r], P(k) will always equal 1 for all k where $p \leq k \leq r$
Therefore pivot $q = \sum_{k=p}^{r} P(k) = P(k) + P(k+1) + ... + P(r) = r$ for all k where $p \leq k \leq r$ and whenever pivot q = r, you will have one sub array from p to r and another that is empty for any ascending sorted array A from p to r

b.) Descending Order:

Base Case:
$p = 1, r = 2$ and array A is sorted in descending order. $P(k) = 1, if A[k] \leq A[r]; 0, otherwise.$ Pivot $q = P(1) + P(2) = 0 + 0 = 0$, therefore q = p, which leaves two arrays, one from p to r and another empty as expected.
Proof by induction:
Assume array A is sorted in descending order and all values are unique. So for any value k where $p \leq k \leq r$, $A[k] \leq A[r]$ for all k.
The pivot point $q = \sum_{k=p}^{r} P(k)$ where $P(k) = \{1$, if A[k] $\leq$ A[r]; 0, otherwise$\}$
Since A[k] will always be greater than A[r], P(k) will always equal 0 for all k where $p \leq k \leq r$
Therefore pivot $q = \sum_{k=p}^{r} P(k) = P(k) + P(k+1) + ... + P(r) = p$ for all k where $p \leq k \leq r$ and whenever pivot q = p, you will have one sub array from p to r and another that is empty for any descending sorted array A from p to r

Q2.

No, the worst case running time is $O(n^2)$ and this happens if every element in the array is the same value.

Q3.

1.)
$q1 = i + 1$
$for j = q1 + 1$ to $r - 1$

$if A[j] \leq x$
$exchange A[i] \iff A[j]$
$exchange A[i+1] \iff A[r]$
$q2 = i + 1$

2.)
Yes, it is possible. If the array is already sorted in ascending order, then the final value of i after the first loop will equal r. So exchange A[i+1] $\iff$ A[r] are the same values so no actual exchange takes place, and the second loop will do the same thing so both pivots end up being the same.

3.)
Worst case would be an array in sorted ascending order. Since both pivots q1 and q2 will the the same value r, the last two calls to triple-quicksort will both have just one comparison. Every time the first triple-quicksort is called, there is just one array element eliminated per call. So the recurrence is T(n) = T(n-1) + T(1) + T(1) + n.

Proof by substitution:
Guess: $T(n) \leq cn^2$ for some value c.
T(n) = T(n-1) + T(1) + T(1) + n
$= T(n-1) + n$
$\leq c(n-1)^2 + n$
$\leq cn^2$ - $c2n$ + 1c + n
$= cn^2$ - cn + c
$\leq cn^2$ + c
$\leq cn^2$

Therefore, $T(n) = O(n^2)$

Q4.

Assuming that Counting Sort is stable (which it is) and that it is correct. Since radix sort just sorts by digit d, starting with LSD, every d value will be sorted correctly using stable counting sort
Base Case:
d = 1, so there is only one digit. Trivial sort that will leave array A sorted.
Proof by induction:
Assume d ¿ 1, so as each digit is read and sorted and a value k where $k \leq d$, for each k in d, it will sort each digit correctly and k+1 will also be sorted properly. Since Counting Sort is stable, the order will be preserved and radix sort will work for all k where $1 \leq k \leq d$ for all k. Therefore radix sort works and is stable.

Q5.

a.) r = 32
T(n) = $64/32 * (80000000 + 2^{32}) = 8749934592$

b.) r = 16
T(n) = $64/16 * (80000000 + 2^{32} = 320262144$

c.) r = 8
T(n) = $64/8 * (80000000 + 2^{32}) = 640002048$

d.) r = ceiling lg(80000000) = 27

$T(n) = 64/27 * (80000000 + 2^{27}) = 507775355$

e.) r = floor lg(80000000) = 26
$T(n) = 64/26 * (80000000 + 2^{26} = 362114126$

Q6.

$$\begin{aligned}
T(n) &= \frac{\lfloor lg((k+1)n^{100}-1)+1 \rfloor}{lg(n)} * (n + 2^{lg(n)}) \\
&= \frac{\lfloor lg((k+1)n^{100}-1)+1 \rfloor}{lg(n)} * (2n) \\
&\leq \frac{lg((k+1)n^{100}-1)+1}{lg(n)} * (2n) \\
&\leq \frac{lg((k+1)n^{100})+1}{lg(n)} * (2n) \\
&\leq \frac{lg(n^{101})+1}{lg(n)} * (2n) \\
&= \frac{101lg(n)+1}{lg(n)} * (2n) \\
&= 202n + \frac{2n}{lg(n)} \\
&\leq 202n + 2n \\
&\leq 204n \\
&= O(n)
\end{aligned}$$

Therefore, T(n) = O(n) with radix sort

Q7.

For this, you would run counting sort in order to create the count array which runs from 0 to k. Once you have the count array built, you just access the array. Algorithm:

```
def rangeCountPreProcess(A, C, k):
  if len(A) > 1:
    for i in range(0, k+1):
      C.append(0)
```

```
    for j in range(0, len(A)):
        C[A[j]] += 1

    for i in range(1, k+1):
        C[i] = C[i] + C[i-1]

def rangeCount(C, a, b):
    if a < 1:
        return C[b]
    else:
        return C[b] - C[a-1]
```

Preprocess:
rangeCountPreProcess(A, C, k):

Find number of elements from a to b:
count = rangeCount(C, a, b)
Finds count in O(1) time.

Q8.

T(n) = T(n-1) + n gives worst case running time:
{3,2,9,0,7,5,4,8,6,1}
{3,2,9,0,7,5,4,8,6}
{3,2,9,0,7,5,4,8}
{3,2,9,0,7,5,4}
{3,2,9,0,7,5}
{3,2,9,0,7}
{3,2,9,0}
{3,2,9}

{3,2}
{3}

Q9.

Split n into $n/5$ groups with at least 5 elements each, sort elements in groups and find the median of each group. Then find the median out of the 5 group's medians, x.
$x \geq 3(\frac{1}{2}\lceil\frac{n}{5}\rceil - 2)$
$\geq \frac{3n}{10} - 6$
Therefore there are at least $\frac{3n}{10} - 6$ elements that are smaller and larger than x.
Assume n = 140.
$\frac{3n}{10} - 6 = \frac{420}{10} - 6 = 36$
$\frac{n}{4} = \frac{140}{4} = 35$
Since $36 > 35$, any value of $n \geq 140$ will have at least $\lceil\frac{n}{4}\rceil$ values less and greater than x.

Q10.

```
Median-Closest(S, n, k):
    if n ≤ k:
        return S
    median = Select(S, n)
    if k % 2 == 0:
        l = median - k/2
        r = median + k/2
    else:
        l = median - (k-1)/2
        r = median + (k-1)/2
```

```
x = Randomized-Select(S, 1, n, l)
y = Randomized-Select(S, 1, n, r)
count = 0, A = []     for i from 1 to n:
   if S[i] > a and S[i] < b:
      A[count] = S[i]
      count += 1
return A
```