CS 101, Assignment 3

Christopher Huynh, Oliver Rene

April 2016

1 Question 1

1.1

$T(n) \leq 7T(\frac{n}{3}) + n^2$, $T(3) \leq 1$
If we draw a recursion tree we find that there are 7^i nodes at depth i, and input size is $\frac{n}{3^i}$. Therefore the total cost of $T(n)$ at depth i is $c \cdot 7^i \cdot (\frac{n}{3^i})^2$ or $c \cdot (\frac{7}{9})^i \cdot n^2$.
For finding the total cost of $T(n)$ we solve

$$\sum_{i=0}^{maxdepth} c \cdot (\frac{7}{9})^i \cdot n^2 = cn^2 \sum_{i=0}^{maxdepth} (\frac{7}{9})^i$$

$$\leq cn^2 \sum_{i=0}^{\infty} (\frac{7}{9})^i = O(n^2)$$

1.2

$T(n) \leq 7T(\frac{n}{3}) + n$, $T(3) \leq 1$
If we apply the master theorem where a = 7, b = 3, c = 1, then $log_3 7 \approx 1.77 > 1$.
Therefore by the master theorem $T(n) = O(n^{log_3 7})$

1.3

$T(n) \leq 7T(\frac{n}{3}) + 1$ or $T(n) \leq 7T(\frac{n}{3}) + n^0$, $T(3) \leq 1$
If we apply the master theorem where a = 7, b = 3, c = 0, then $log_3 7 \approx 1.77 > 0$.
Therefore by the master theorem $T(n) = O(n^{log_3 7})$

1.4 Definition of Master Theorem

If $T(n) = aT(n/b) + n^c$ for constants $a > 0, b > 1, c \geq 0$, then
1. $T(n) = O(n^c)$ if $log_b a < c$
2. $T(n) = O(n^c logn)$ if $log_b a = c$
3. $T(n) = O(n^{log_b a})$ if $log_b a > c$

2 Question 2

$T(n) \leq 2T(\frac{n}{2}) + c\sqrt{n}$ with $T(1) = 1$
Prove $T(n) = O(n)$
We will therefore need to prove that

$$T(n) \leq 4(n - \sqrt{n})$$

for some $n \geq n_0$
Choose $n_0 = 2$

- **Base case:** $T(2) \leq 2T(\frac{2}{2}) + \sqrt{2}$
  $\leq 2 + \sqrt{2}$
  $4x(2 - \sqrt{2}) = 8 - \sqrt{2} \geq T(2)$
- **Induction:** $T(n) \leq 2T(\frac{n}{2}) + \sqrt{n}$
  $\leq 2(4(\frac{n}{2} - \sqrt{\frac{n}{2}})) + \sqrt{n}$
  $= 2(2n - 4\sqrt{\frac{n}{2}}) + \sqrt{n}$
  $= 2(2n - \sqrt{\frac{16n}{2}}) + \sqrt{n}$
  $= 2(2n - \sqrt{8n}) + \sqrt{n}$
  $= 2(2n - 2\sqrt{2n}) + \sqrt{n}$
  $= 4n - 4\sqrt{2n} + \sqrt{n}$
  $\leq 4(n - \sqrt{n})$

By proof by induction, $T(n) = O(n)$ $\blacksquare$

3 Question 3

3.1 Pseudocode

```
Count-Inversions(A)
n = A.length
if n == 1, return 0
count = 0
L = A[1..n/2]
R = A[(n/2 + 1)..n]
L_count = Count-Inversions(L)
R_count =Count-Inversions(R)
L[m + 1] = R[m + 1] = ∞
i = j = 1
while L[i] < ∞ or R[j[< ∞
    if L[i] > R[j]
        if L[i] != ∞, count = count + 1
        i = i + 1
    else j = j + 1
return count + L_count + R_count
```

3.2 Time-Complexity Analysis

In each recurrence of Count-Inversion, it runs itself 2 more time on sizes $\frac{n}{2}$. All additional run time (including the while loop) is in θ(n), so its at most cn. Therefore the recurrence relation for Count-Inversion is

$$T(n) \leq 2T(n/2) + cn$$

$$T(1) \leq 1$$

Lets guess $T(n) = O(nlog_2 n)$
We will therefore need to prove that

$$T(n) \leq c' nlog_2 n$$

for some constant $c' > 0$ and $n \geq n_0$.
Choose $c = c$ and $n_0 = 2$.

- **Base case:** $T(2) \leq 2T(1) + cn$
  $= 2 + c$
  $c \cdot 2 \cdot log_2 2 = 2c \geq T(2)$
- **Induction:** $T(n) \leq 2T(n/2) + cn$
  $\leq 2c(n/2)log_2(n/2) + cn$
  $= cnlog_2(n/2) + cn$
  $= cn(log_2 n - 1) + cn$
  $= cnlog_2 n$

Therefore by proof by induction, the time complexity of Count-Inversion, $T(n) = O(nlogn)$ ■

4 Question 4

4.1 Pseudocode

```
Kth-Smallest(A, k)
piv = A[1]
// Partition A around the pivot and return the position of pivot
pos = Partition(A)
if pos == k, return piv
else if pos > k, return Kth-Smallest(A[1..(pos − 1)], k)
else if pos < k, return Kth-Smallest(A[(pos + 1)..n], k − pos)
```

```
Partition(A)
n = A.length
piv = A[1]
i = 1
j = n + 1
while (True)
    do i = i + 1 while A[i] < piv
    do j = j − 1 while A[j] > piv
    if i ≥ j, break
    Swap A[i] and A[j]
Swap A[1] and A[j]
return j
```

4.2 Worst-Case Time Complexity

In the worst case that the position of the pivot after partitioning is always less than k, the recurrence of Kth-Smallest would run itself once on a size n-1. All additional run time (including the partitioning step) is in θ(n), so its at most cn. Therefor the recurrence relation for Kth-Smallest is

$$T(n) \leq T(n-1) + cn$$

This is the exact recurrence relation of Quick-Sort (which run time was solved in class) so we can conclude that Kth-Smallest, $T(n) = O(n^2)$

4.3 Randomized Pivot

Because we only care about the kth smallest element we are only running recurrences in the event that $pivot < k$ or $pivot > k$. Each element in array A has the possibility to be the pivot with the chance of $1/n$. Let s be the number of elements to the left of the pivot after partitioning. Therefore, the number of elements to the right of the pivot after partitioning is $n - s - 1$ The possibility that $pivot > k$ is $1/s$ and the possibility that $pivot < k$ is $1/(n - s - 1)$. From this we get the recurrence relation

$$T(n) \leq \frac{1}{n}[\sum_{s=0}^{n-1}(\frac{1}{s}T(s) + \frac{1}{n-s-1}T(n-s-1)] + cn$$

$$= \frac{1}{n}[\frac{1}{s}\sum_{s=1}^{n-1}T(s) + \frac{1}{n-s-1}\sum_{s=1}^{n-1}T(n-s-1)] + cn$$

$$= \frac{2}{n}[(\frac{1}{s} + \frac{1}{n-s-1})\sum_{s=1}^{n-1}T(s)] + cn$$

$$T(n) \leq \frac{2(n-1)}{ns(n-s-1)}[\sum_{s=1}^{n-1}T(s)] + cn$$

$$T(2) \leq c$$

Unfortunately, this recurrence is too complicated for us to solve, but we know from the algorithm's similarity to QuickSort, the runtime should be $T(n) = O(nlogn)$

5 Question 5

The elements must be distinct because partitions of equal size will be confused. In the even that there are many of the same elements, how do you decide which is k smallest? Would you consider each to be k or would you completely skip over all recurrences of the same element? A change that could be made to our original algorithm actually run a different form of recursion so that the entire array is sorted, then simply find the kth smallest element regardless of non-distinct elements. The time complexity should be similar if not identical to QuickSort.