CMPS 101, Spring 2016: HW 3

Merrick Swaffar and Siobhan O'Shea

29 April 2016

Q1:

- a = 7, b = 3, c = 2, case 3
  $T(n) \leq 7T(n/3) + n^2$
  $log_3 7 = 2$
  $T(n) = O(n^{log_3 7})$

- a = 7, b = 3, c = 1, case 1
  $T(n) \leq 7T(n/3) + n$
  $log_3 7 > 1$
  $T(n) = O(n^{log_3 7})$

- a = 7, b = 3, c = 0, case 1
  $T(n) \leq 7T(n/3) + 1$
  $log_3 7 > 0$
  $T(n) = O(n^{log_3 7})$

- Master Theorem cases:
  $T(n) = aT(n/b) + f(n)$

  Case 1 :
  $f(n) \epsilon O(n^2), c < log_b a \Rightarrow T(n) \epsilon \theta(n^{log_b a})$

  Case 2 :
  $f(n) \epsilon \Theta(n^c log^k n), c = log_b a \Rightarrow T(n) \epsilon \theta(n^c log^{k+1} n)$

  Case 3:
  $f(n) \epsilon \Omega(n^c), c > log_b a \Rightarrow T(n) \epsilon \theta(f(n))$

Q2: $T(n) \leq 2T(n/2) + \sqrt{n}, T(1) = 1$
Prove T(n) ϵ O(n)

$T(n) \leq an + b\sqrt{n} \Rightarrow T(n) \epsilon O(n)$, where a and b are sufficiently large constants

base case:
$T(1) = 1 \leq a(1) + b\sqrt{1}$
for a = 2 and b = 1

inductive hypothesis:
$T(n/2) \leq a(n/2) + b\sqrt{n/2}$

induction:
$T(n) \leq 2T(n/2) + \sqrt{n}$
$\leq 2(a(n/2) + b\sqrt{n/2}) + \sqrt{n}$
$\leq a(n) + \frac{2}{\sqrt{2}}b\sqrt{n} + \sqrt{n}$
$\leq a(n) + (\frac{2}{\sqrt{2}}b + 1)\sqrt{n}$
$\leq a(n) + b\sqrt{n}$

Q3:

- inversions(A)
```
    n = A.length
    if n < 2
        return 0
    L = A[1, ... , n/2]
    R = A[n/2 + 1, ... , n]
    Count+ = inversions(L)
    Count+ = inversions(R)
    i = j = k = 1
    while i < L.length or j < R.length
        if L[i] ≤ C[j]
        i = i+1
    else
        j = j+1
        count + = L.length - i
    return count
```

The time complexity of this algorithm is $\Theta(nlogn)$. You are recursively splitting the array into two sub arrays, then you perform a linear time combine step. Therefore, it's time complexity is governed by the recurrence $T(n) = 2T(n/2) + cn$, just as with merge sort.

Q4:

1. k^{th}smallest(A,k)
```
    i = partition(A) //the partition algorithm discussed in class
    if (i = k)
        return A[k]
    n = A.length
```

L = A[1, ... , i - 1]
R = A[i + 1, ... , n]
if (i > k)
    return k^{th}smallest(L, k)
return k^{th}smallest(R, k - i)

2. In the worst case partition separates the array into two arrays of size 0 and $n-1$. This implies that there are $\sum_{i=1}^{n}(n-1)$ calls to partition, and partition runs in linear time, so the worst case time complexity is $O(n^2)$.

3. randomized-k^{th}smallest(A,k)
    n = A.length
    r = random(1 to n)
    swap(A[1] , A[r])
    i = partition(A) //the partition algorithm discussed in class
    if (i = k)
        return A[k]
    L = A[1, ... , i - 1]
    R = A[i + 1, ... , n]
    if (i > k)
        return k^{th}smallest(L, k)
    return k^{th}smallest(R, k - i)

$T(n) \leq \frac{2}{n}[\sum_{s=1}^{n-1} T(S)] + cn$
$T(n) \leq anlog_2 n - bn$

base case:
$n = 2$
$c = T(2) \leq a2log_2 2 - 2b$
$= 2a - 2b$
$c \leq 2(a - b)$

induction:
Assume $T(s) \leq aslog_2 s - bs \forall s < n$
$T(n) \leq [\sum_{s=1}^{n-1}(aslog_2 s - bs)] + cn$
$\leq \frac{2a}{n}[\sum_{s=1}^{n-1}(slog_2 s)] - \frac{2b}{n}[\sum_{s=1}^{n-1}(s)] + cn$
$\leq anlog_2 n - bn$