Assignment #2

Due on Thursday 1/31/2019

1. The following pseudo-code shows a variation of the merge sort algorithm where in each iteration, the list is divided into three sublists.

```
function MERGE-SORT(A, p, r)
    Return If p ≥ r
    q1 ← ⌊(2 * p + r)/3⌋
    q2 ← ⌊(p + 2 * r)/3⌋
    MERGE-SORT(A, p, q1)
    MERGE-SORT(A, q1 + 1, q2)
    MERGE-SORT(A, q2 + 1, r)
    MERGE(A, p, q1, q2, r)
end function
```

   (a) Write down the recurrence relation that represents the running time of the above algorithm.

   (b) Solve the recurrence relation using the expansion of the recurrence tree.

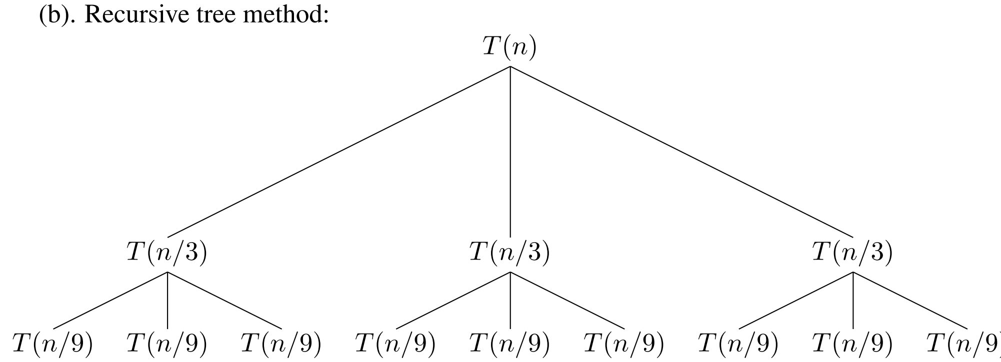
   (c) Solve the recurrence relation using the Master method.

   (d) Compare the running time of the proposed algorithm to that of the regular merge sort algorithm shown on page 34 of the textbook.

Answer:

(a). $T(n) = 3T(n/3) + \theta(n)$

(b). Recursive tree method:



Each level has a runtime of cn and there are totally d level(first leverl is 0) On level d, there are n T(1) adding together, therefore, we get $\frac{n}{3^{d-1}} = 1$ ⇒d=$\log_3$ n + 1⇒T(n) = c n d = cn($\log_3$ n + 1)=$\theta(nlog_3n) = \theta(nlogn)$

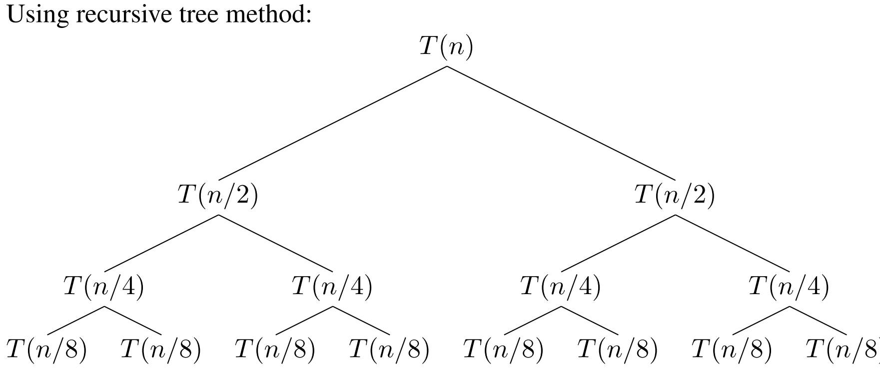
(c). Using master theory:
$a = 3, b = 3, f(n) = \theta(n)$ ⇒n$^{log_b a}$ = n$^{log_3 3}$ = n
⇒f(n)=$\theta(n^{log_b a})$ case 2 hold ⇒T(n)=$\theta(f(n)log_n) = \theta(nlogn)$

(d). Both algorithm has the same complexity

2. Use the recurrence tree expansion method to find a tight asymptotic bound to the following recurrence relation. For simplicity, assume that n is always a power of two and $T(1) = 1$.

$$T(n) = 2T(n/2) + n \lg n$$

Using recursive tree method:



following this pattern, we have total d levels (first level is 0)
for level 0, the non-recursive part is $nlogn$
for level 1, the non-recursive part for each of two subcase is $\frac{n}{2}log\frac{n}{2}$(where the total runtime is $nlog\frac{n}{2}$)
for level 2, the non-recursive part for each of four subcase is $\frac{n}{4}log\frac{n}{4}$(where the total runtime is: $nlog\frac{n}{4}$)
following this pattern, the total runtime is
$T(n) = n(log\frac{n}{n} + ... + log\frac{n}{4} + log\frac{n}{2} + logn) = nlog\frac{n^(d+1)}{2^{(d+1)d/2}}$ and we know $d = logn + 1$ by substituting d, we get $T(n) = \theta(nlog^2n)$

3. Solve the following recurrences using the Master theorem.

   (a) $T(n) = 2T(n/8) + n \log n$
   (b) $T(n) = 2T(n/4) + \sqrt{n}$
   (c) $T(n) = 9T(n/3) + n$

   answer:
   (a). $a = 2, b = 8, f(n) = nlogn \Rightarrow n^{log_b a} = n^{log_8 2} \Rightarrow f(n) = \Omega(n^{log_8 2})$
   $af(n/2) = 2 \cdot (n/8) \cdot log(n/8) \leq c \cdot f(n) = c \cdot nlogn$, c can be $\frac{1}{2}$
   therefore, case 3 hold, $T(n) = \theta(f(n)) = \theta(nlogn)$

   (b).$a = 2, b = 4, f(n) = \sqrt{n} \Rightarrow n^{log_b a} = n^{log_4 2} = n^{0.5} \Rightarrow f(n) = \theta(n^{0,5}) \Rightarrow$ case 2 hold $\Rightarrow T(n) = \theta(f(n)logn) = \theta(\sqrt{n}logn)$

   (c). $a = 9, b = 3, f(n) = n \Rightarrow n^{log_b a} = n^{log_3 9} = n^2 \Rightarrow f(n) = O(n^{2-\epsilon})$ and ϵ can be 0.5 , therefore, case 1 hold. $T(n) = \theta(n^2)$

4. Suppose that we want to create a divide and conquer matrix multiplication algorithm for square matrices. Assuming that n is a power of three, the algorithm divides each matrix of A, B, and C into nine equi-sized matrices. Then, it performs some recursive calls to compute each submatrix of c.

   (a) How many recursive calls need to be made so that the algorithm will have an asymptotic running time of $\Theta(n^3)$?

   (b) What is the maximum number of recursive calls that can be made while having an asymptotic running time that is lower than that of Strassen's algorithm?
   answer:
   (a) 27 calls.
   (b) 21 calls.

   Note: You do not have to show an actual algorithm that works for this case. You just need to find the number of recursive calls for a hypothetical algorithm that divides the matrix into nine submarices.

5. Let X be a $kn \times n$ matrix and Y by an $n \times kn$ matrix, for some integer k.

   (a) Describe an algorithm that computes the product XY using Strassen's algorithm as a subroutine, i.e., use it as a black-box without modifying it. Only describe your algorithm in words; pseudo-code is not required. Justify your answer, i.e., argue that your algorithm does compute XY correctly. Establish its running time.

   (b) Repeat part (a) for computing the product YX.
   answer:
   (a). divide X to be k $n \times n$ size sub-matrix $[X_1, X_2, X_3....X_n]$, and do same thing to Y$[Y_1, Y_2, Y_3......Y_n]^T$, and use strassen's algorithms to multiply two matrix.
   The run time for strassen's algorithm is $\theta(n^{log_2 7})$
   Because the matrix X has kn rows and Y has kn columns, the result matrix size is $kn \times kn$. Therefore, there are k^2 times of multiplication. The total runtime is $\theta(k^2 n^{log_2 7})$
   (b). using the same idea to divide the two matrix. Because the result matrix has the size of $n \times n$, there are k times multiplication. Therefore, the runtime is $\theta(kn^{log_2 7})$