Case 1: $\log_b a > k$ then $\Theta(n^{\log_b a})$
Case 2: $\log_b a = k$ if $p > -1$ $O(n^k \log^{p+1} n)$
if $p = -1$ $O(n^k \log(\log n))$
if $p < -1$ $O(n^k)$
Case 3: $\log_b a < k$ if $p \geq 0$ $O(n^k \log^p n)$
if $p < 0$ $O(n^k)$

CS3343 Analysis of Algorithms Fall 2019

Homework 2

Due 9/19/19 before 11:59pm (Central Time)

$T(n) = aT(n/b) + f(n)$
$a \geq 1$
$b > 1$

1. Master theorem (6 points)

Use the master theorem to find tight asymptotic bounds for the following recurrences. If the master theorem cannot be applied, state the reason and give an upper bound (big-Oh) which is as tight as you can justify. Justify your answers (by showing which case of the master theorem applies, and why.)

(1) $T(n) = 9T(n/3) + \sqrt{n}$
(2) $T(n) = 2T(n/2) + n \log n$
(3) $T(n) = 36T(n/6) + n^2$
(4) $T(n) = T(2n/5) + n$
(5) $T(n) = T(2n/3) + T(n/3) + n$
(6) $T(n) = T(n-2) + n$

2. Recursive Program (5 points)

Consider the following recursive function for $n \geq 0$:

Algorithm 1 int recFunc(int n)

```
//Base Case:
if n == 0 then
    return 1;
end if
if n == 1 then
    return 2;
end if
//Recursive Case:
int a = 0
int j = recFunc(n/2)
while j > 0 do
    print("Simulating Trammel of Archimedes...");
    a = a + recFunc(n/2);
    j = j - 1;
end while
return a;
```

(1) Use strong induction to prove the value returned by recFunc(n) is $\leq 2^n$.
(2) Set up a runtime recurrence for the runtime $T(n)$ of this algorithm.
Hint: Use the fact that recFunc(n) $\leq 2^n$ to help in setting up the recurrence.
(3) Solve this runtime recurrence using the master theorem or if the master theorem cannot be applied, state the reason and give an upper bound (big-Oh) which is as tight as you can justify.

3. Big-Oh Induction (4 points)

Use strong induction to prove that $T(n) \in O(\log n)$ where $T(1) = 90$ and $T(n) = T(n/2) + 10$ (for $n \geq 2$). Given $T(n/2) + 10$ where $T(1) = 90 \; n \geq 2$

4. Recursive Algorithm and Analysis (6 points) prove $T(n) \in O(\log n)$!

Suppose company X has hired you as a software engineer.

The bigwigs at company X heard about how Divide and Conquer can be used to improve the runtime of **matrx multiplication**. Thus, rather than allow you to use the standard **matrix addition** algorithm, they want you to also create a Divide and Conquer algorithm to add matrices.

Reminder: Suppose your input matrices are A and B. Likewise suppose your output matrix is C. Specifically for all i, j your code computes $C[i][j] = A[i][j] + B[i][j]$. You can assume A, B, and C are all $n \times n$ matrices and, for simplicity, that $\exists_{x \in \mathbb{N}} : n = 2^x$ (i.e. n is a power of 2).

(1) (2 points) Identify the Divide, Conquer, and Combine steps of your new algorithm.
(**Note:** a one sentence explanation of each is fine; see notes for examples).
(2) (1 point) Create a recurrence to represent the runtime of your new algorithm.
(3) (2 points) Use master theorem to solve your above recurrence relation. Compare this asymptotic run time to the standard $O(n^2)$ matrix addition algorithm.
(4) (1 point) Do you think there's a trick like Strassen's to improve the asymptotic runtime of your Divide and Conquer matrix addition algorithm? Justify why or why not.

$T(n) = aT(n/b) + f(n)$

$a \geq 1$

$b > 1$

Case 1: $\log_b a > k$ then $\Theta(n^{\log_b a})$

case 2: $\log_b a = k$ if $p > -1$ $O(n^k \log^{p+1} n)$

if $p = -1$ $O(n^k \log\log n)$

if $p < -1$ $O(n^k)$

Case 3: $\log_b a < k$ if $p \geq 0$ $O(n^k \log^p n)$

if $p < 0$ $O(n^k)$

(1) T(n) = 9T(n/3) + √n

$T(n) = 9T(n/3) + n^{1/2}$

$a = 9 \quad b = 3 \quad k = \frac{1}{2}$

$\log_3 9 = 2 > k$

case 1: $\Theta(n^2)$

(2) T(n) = 2T(n/2) + n.log n

$a = 2 \quad b = 2 \quad k = 1$

$\log_2 2 = 1 = k \quad p = 1$

case 2: $\Theta(n \log^2 n)$

(3) T(n) = 36T(n/6) + n^2

$a = 36 \quad b = 6 \quad k = 2 \quad p = 0$

$\log_6 36 = 2 = k$

case 2: $\Theta(n^2 \log n)$

(4) T(n) = T(2n/5) + n

$a = 1 \quad b = \frac{2}{5} \quad k = 1$

$\log_{\frac{2}{5}} 1 = 0$

Case 3:

$O(n \log n)$

(5) T(n) = T(2n/3) + T(n/3) + n

$T(n) \leq c \lg n$

$\leq (c 2n/3) \lg(2n/3) + c(n/3) \lg(n/3) + n$

$\leq c\left(\frac{2n}{3}\right)\left(\lg n - \lg\frac{3}{2}\right) + c\frac{n}{3}\left(\lg n - \lg\frac{3}{1}\right) + n$

$\leq \frac{c2n}{3}(\lg 3 - \lg 2) + cn\lg - (cn/3)\lg 3 + n$

$\leq cn\lg - cn\lg 3 + 2cn/3 + n$

So when we let $\frac{n + 2cn}{3cn\lg 3} < 0$

$c = 3 \quad n > 0$

then $(3 - 3\lg 3)n < 0$

So $T(n) = O(n \lg n)$

(6) T(n) = T(n − 2) + n

$T(n)$

2. Recursive Program (5 points)

(1) Use strong induction to prove the value returned by recFunc(n) is ≤ 2^n.

Given: $T(n) = 2T(n/2) + O(n)$ where $n \geq 0$

prove: $f(n)$ is $\leq 2^n$

Base case: assume $T(n)$ is true for all $n \geq 0$

$T(0) = 2\left(\frac{0}{2}\right) + 0 \qquad 2^0$

$= 0 \leq 1$ ✓

inductive case: assuming $T(n)$ is true for all $n \geq 0$

prove $T(n+1)$ is also true

$T(0+1) = 2\left(\frac{0+1}{2}\right) + 0 + 1$

$T(1) = 2\left(\frac{1}{2}\right) + 1$

$T(1) = 1 + 1 = 2 \leq 2$ ✓

So, recfunc(n) is $\leq 2^n$ for all $n \geq 0$

(2) Set up a runtime recurrence for the runtime T(n) of this algorithm. Hint: Use the fact that recFunc(n) ≤ 2n to help in setting up the recurrence.

$T(n) = 2T(n/2) + O(n)$ for all $n \geq 0$

(3) Solve this runtime recurrence using the master theorem or if the master theorem cannot be applied, state the reason and give an upper bound (big-Oh) which is as tight as you can justify.

$T(n) = 2T(n/2) + O(n)$ for all $n \geq 0$

$a = 2 \quad b = 2 \quad k = 1 \quad p = 0$

$\log_2 2 = 1$

Case 2: $O(n^k \log^{p+1} n)$

So, $\Theta(n \log n)$

3. Big-Oh Induction (4 points)
Use strong induction to prove that $T(n) \in O(\log n)$ where $T(1) = 90$ and $T(n) = T(n/2) + 10$ (for $n \geq 2$).

Base case: $T(n) = T(n/2) + 10 \quad T(1) = 90 \quad n \geq 2$

$T(n) \leq c \cdot \log n$

$\log_2 1 = 0$

$n_0 = 2$

LHS: $T(2) = 2\,T(2/2) + 10$

$T(1) + 10 = 90 + 10$

$= 100$

RHS: $c \cdot \log_2(2) = c$

$c = c$

$c \geq 100$

Ind. step:

Assume $T(k) \leq c \cdot \log k$ for all $k < n$, $k \geq n_0$

Prove $T(n) \leq c \cdot \log n$

$T(n) = T(n/2) + 10 \leq c \log_2(n/2) + 10$

$c \log \frac{n}{2} + 10$

$c(\log n - \log 2) + 10$

$c(\log_n - 1) + 10$

$c \geq 100$

$RHS \leq c \log n - c + 10$

$\leq c \log n - 90 + 0$ (100)

So, $T(n) \in O(\log n)$

4. Recursive Algorithm and Analysis (6 points) Suppose company X has hired you as a software engineer. The bigwigs at company X heard about how Divide and Conquer can be used to improve the runtime of matrx multiplication. Thus, rather than allow you to use the standard matrix addition algorithm, they want you to also create a Divide and Conquer algorithm to add matrices.
Reminder: Suppose your input matrices are A and B. Likewise suppose your output matrix is C. Specifically for all i j your code computes C[i][j] = A[i][j] + B[i][j]. You can assume A,B, and C are all n × n matrices and, for simplicity, that ∃x∈N : n = 2x (i.e. n is a power of 2).
(1) (2 points) Identify the Divide, Conquer, and Combine steps of your new algorithm. (Note: a one sentence explanation of each is fine; see notes for examples).
(2) (1 point) Create a recurrence to represent the runtime of your new algorithm.
(3) (2 points) Use master theorem to solve your above recurrence relation. Compare this asymptotic run time to the standard O(n^2) matrix addition algorithm.
(4) (1 point) Do you think there's a trick like Strassen's to improve the asymptotic runtime of your Divide and Conquer matrix addition algorithm? Justify why or why not.

(1) assumy we are mutipyiny 2 matrices we can divide eaen matrices in to 4 sub matrices and mutipy.

(2) assume $n=2$ $\quad T(n) = 8T(n/2) + n^2$

(3) $\longrightarrow$ $n\log_b a = 2 \quad b=2 \quad A=8$

$n\log_2 8 = n^3$ $\qquad c = n^2$

case 1: $n^3 \in O(n^{3-\varepsilon}) \quad \varepsilon = 1$

$n^2 \in O(n^2)$

$\Rightarrow T(n) \in \Theta n^3$

(4) yes, but it wouldn't make significant difference if the matrix is not huge.