Data Structure and Algorithms

Lab4

Submitted to:	**Ms. Sehar Waqar**
Submitted by:	**SABA**
Roll no:	**2019-CE-04**

Department of Computer Engineering

UET,Lahore

Homework Question:

Matrix multiplication: Suppose that we have n × n matrices X and Y and we'd like to multiply them.

a) What is the running time of the standard algorithm (that computes the inner product of rows of X and columns of Y)? You can assume that simple arithmetic operations, like multiplication, take constant time (O(1)).

SQUARE-MATRIX-MULTIPLY(A,B)

1. n = A.rows → cost = c_0, 1 time
2. let C be a new n×n matrix
3. for i = 1 to n → cost = c_1, n times
4. for j = 1 to n → cost = c_2, (n*n)times
5. $c_{ij} = 0$ → cost = c_3, (n*n)times
6. for k = 1 to n → cost = c_4, (n*n*n)times
7. $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ → cost = c_5, (n*n*n)times
8. return C → cost = c_6, 1 time

$T(n) = c_0 + c_6 + c_1 n + (c_2 + c_3)(n*n) + (c_4 + c_5)(n*n*n)$

$\mathbf{T(n) = O(n^3)}$

Because each of the triply-nested for loops runs exactly n iterations, and each execution of line 7 takes constant time, the SQUARE-MATRIX-MULTIPLY procedure takes $O(n^3)$ time.

---

b) Now let's divide up the problem into smaller chunks like this, where the eight n/2 x n/2 sub-matrices (A, B, C, D, E, F, G, H) are each quarters of the original matrices, X and Y : We now have a divide and conquer strategy! Find the recurrence relation of this strategy and the runtime of this algorithm.

SQUARE-MATRIX-MULTIPLY-RECURSIVE (A,B)

1. n = A.rows
2. let C be a new n×n matrix
3. if n == 1
4. $c_{11} = a_{11} \cdot b_{11}$
5. else partition A,B and C
6. C_{11} = SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{11},B_{11}) + SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{12},B_{21})
7. C_{12} = SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{11},B_{12}) + SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{12},B_{22})
8. C_{21} = SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{21},B_{11}) + SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{22},B_{21})
9. C_{22} = SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{21},B_{12}) + SQUARE-MATRIX-MULTIPLY-RECURSIVE (A_{22},B_{22})

Ans: **Recurrence relation:**

Base Case : When n = 1 , we perform just one scalar multiplication.

So, T(1) = Θ(1)

Recursive Case : When n >1 , we partitioning the matrices in Θ(1) time, and recursively call SQUARE-MATRIX-MULTIPLY-RECURSIVE (A,B) a total of 8 times.

Time taken by one recursive call = T(n/2)

Time taken by eight recursive calls = 8T(n/2)

Time taken by addition of matrices = $\Theta(n^2)$

$$T(n) = \begin{cases} \Theta(1), & if\ n = 1 \\ 8T\left(\frac{n}{2}\right) + \Theta(n^2), & if\ n > 1 \end{cases}$$

Running Time by using Master theorem:

a = 8 , b=2 , d = 2

$b^d = 2^2 = 4$

As, $a > b^d$ → T(n) = $O(n^{\log_b a})$ = $O(n^{\log_2 8})$)

T(n) = $O(n^3)$

---

c) Can we do better? It turns out we can by calculating only 7 of the subproblems:

P1 = A(F − H) P2 = (A + B)H P3 = (C + D)E P4 = D(G − E)

P5 = (A + D)(E + H) P6 = (B − D)(G + H) P7 = (A − C)(E + F)

And we can solve XY by

$$XY = \begin{bmatrix} P4 + P5 - P2 + P6 & P1 + P2 \\ P3 + P4 & P1 + P5 - P3 - P7 \end{bmatrix}$$

We now have a more efficient divide and conquer strategy! What is the recurrence relation of this strategy and what is the runtime of this algorithm?

Ans: We use Strassen's method**.** The key to Strassen's method is to make the recursion tree slightly less bushy. That is, instead of performing eight recursive multiplications of n/2×n/2 matrices, it performs only seven.

So, the recurrence relation is :

$$T(n) = \begin{cases} \Theta(1), & if\ n = 1 \\ 7T\left(\frac{n}{2}\right) + \Theta(n^2), & if\ n > 1 \end{cases}$$

Running Time by using Master theorem:

a = 7 , b=2 , d = 2

$b^d = 2^2 = 4$

As, $a > b^d$ → T(n) = $O(n^{\log_b a})$ = $O(n^{\log_2 7})$)

T(n) = $O(n^{2.81})$

---

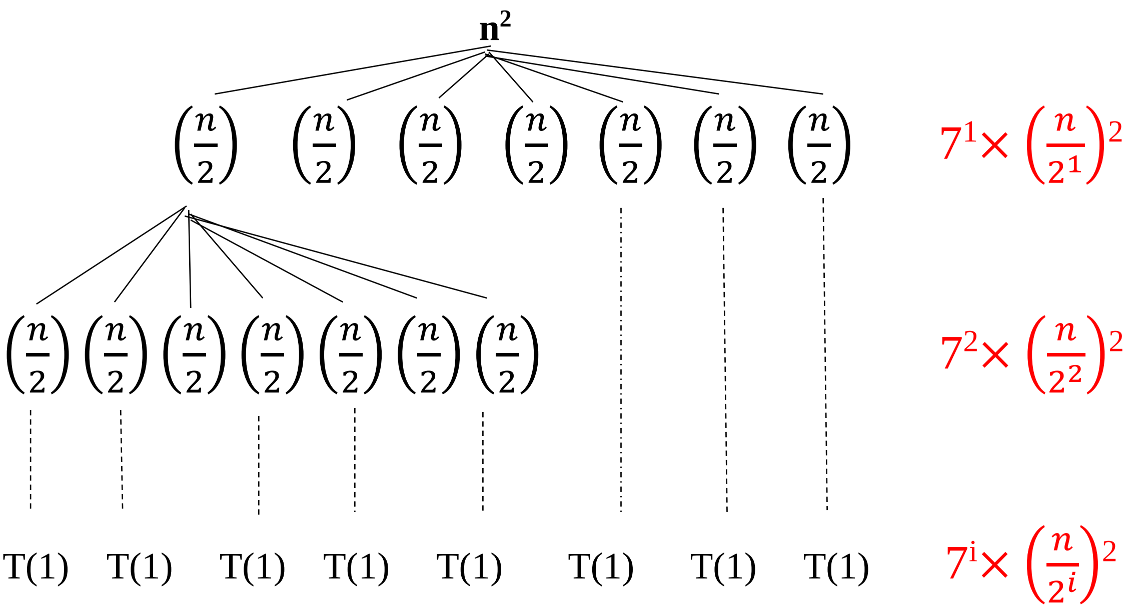
d) Claim: The algorithm runs in time T(n) = $O(n^2 \log(n))$.

Proof: At the top level, we have 7 operations adding/subtracting n/2 × n/2 matrices, which takes $O(n^2)$ time. At each subsequent level of the recursion, we increase the number of subproblems by a constant factor (7), and the subproblems only become smaller. Therefore, the running time per level can only increase by a constant factor. By induction, since for the top level it is $O(n^2)$, it will be $O(n^2)$ for all levels. There are O(log(n)) levels, so in total the running time is $T(n) = O(n^2\log(n))$. What is the error in this reasoning?

Ans: Error: The above proof says that it is $O(n^2)$ for all levels which is wrong.

Proof:

$$T(n) = \begin{cases} \Theta(1), & if\ n = 1 \\ 7T\left(\frac{n}{2}\right) + \Theta(n^2), & if\ n > 1 \end{cases}$$



Branching factor: 7

Cost at each level: $7^i \times \left(\frac{n}{2^i}\right)^2$

Depth of tree: $\frac{n}{2^i} = 1 \rightarrow i = \log n$

Number of leaves: $7^{\log n} = n^{\log 7} = n^{2.81}$

Total cost = $\sum_{i=0}^{\log(n-1)} 7^i \times \left(\frac{n}{2^i}\right)^2 + O(n^{2.81})$

$T(n) = O(n^{2.81})$

Hence, we proves that cost at each level is not $O(n^2)$.

Recurrence Questions:

Question1:

There exist different ways to solve the recurrence relation T(n) = 2·T(n/2)+n with T(1)= 1. From lectures, we have seen that T(n) is exactly n(1 + log(n)) when n is a power of 2.

a) What is the exact solution to T(n) =3T(n/3)+n with T(1) = 3, when n is a power of 3?

Ans: T(n) = 3T($\frac{n}{3}$) +n → $3^1T(\frac{n}{3^1})$ +1n

= 3 [3T($\frac{n}{9}$) + ($\frac{n}{3}$)] + n

= 9T($\frac{n}{9}$) + 2n → $3^2T(\frac{n}{3^2})$ +2n

= 9 [3T($\frac{n}{27}$) + ($\frac{n}{9}$)] + 2n

= 27 T($\frac{n}{27}$) + 3n → $3^3T(\frac{n}{3^3})$ +3n

General equation:

T(n) = $3^iT(\frac{n}{3^i})$ $+in$

Let, $\frac{n}{3^i} = 1$ → n = 3^i → i = $\log_3 n$

So, now general equation becomes

T(n) = $3^{\log_3 n}$ T(1) + $n \log_3 n$

= 3n + $n \log_3 n$ $\quad\quad\quad\quad \therefore T(1) = 3$

T(n) = n(3+$\log_3 n$)

b) What is the exact solution to T(n) = 3T(n/3) + 3n with T(1) = 1, when n is a power of 3?

Ans: T(n) = 3T($\frac{n}{3}$) +3n → $3^1T(\frac{n}{3^1})$ +3×1n

= 3 [3T($\frac{n}{9}$) + n] + 3n

= 9T($\frac{n}{9}$) + 6n → $3^2T(\frac{n}{3^2})$ +3×2n

= 9 [3T($\frac{n}{27}$) + ($\frac{n}{3}$)] + 6n

= 27 T($\frac{n}{27}$) + 9n → $3^3T(\frac{n}{3^3})$ +3×3n

General equation:

T(n) = $3^iT(\frac{n}{3^i})$ + $3in$

Let, $\frac{n}{3^i} = 1$ → n = 3^i → i = $\log_3 n$

So, now general equation becomes

T(n) = $3^{\log_3 n}$ T(1) + $3n \log_3 n$

= n + $3n \log_3 n$ $\quad\quad\quad\quad \therefore T(1) = 1$

T(n) = n(1+3 $\log_3 n$)

Question2:

Use any of the methods we've seen in class so far to give big-Oh solutions to the following recurrence relations. You may treat fractions like n/2 as either floor(n/2) or ceiling(n/2), whichever you prefer.

a) $T(n) = 3T(n/9) + \sqrt{n}$ for $n \geq 9$, and $T(n)=1$ for $n < 9$.

Ans: a =3 , b=9 , d =1/2

$b^d = 9^{1/2} = 3$

As, $a = b^d \rightarrow T(n) = O(n^d \log n)$

So, $T(n) = O(\sqrt{n}\log_3 n)$

b) $T(n) = T(n-4) + n$ for $n \geq 4$, and $T(n) = 1$ for $n < 4$.

Ans: $T(n) = T(n-4) + n$

$= [T(n-8) + (n-4)] + n$

$= [T(n-12) + (n-8)] + (n-4) + n$

$= T(n-16) + (n-12) + (n-8) + (n-4) + n$

General equation:

$T(n) = T(n-k) + (n-(k-4)) + (n-(k-8)) + ------ + (n-4) + n$

Let , $n-k = 0 \rightarrow n=k$

$T(n) = T(n-n) + (n-n+4) + (n-n+8)) + ------ + (n-4) + n$

$= T(0) + 4 + 8 + 12+ ------ + n$

$= 1 + 4[1+2+3+ ------ + n]$

$= 1 + 4\left(\frac{n(n+1)}{2}\right)$

$= 1 + 2n(n+1)$

$= 1 + 2n^2 + 2n$

$= 2n^2 + 2n + 1$

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

c) $T(n) = 6T(n/4) + n^2$ for $n \geq 4$, and $T(n) = 1$ for $n < 4$.

Ans : a =6 , b=4 , d =2

$b^d = 4^2 = 16$

As, $a < b^d \rightarrow T(n) = O(n^d)$

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

d) $T(n) = 5T(n/2) + n^2$ for $n \geq 2$, and $T(n) = 1$ for $n < 2$.

Ans: a = 5 , b=2 , d =2

$b^d = 2^2 = 4$

As, $a > b^d \rightarrow T(n) = O(n^{\log_b a})$

So, $T(n) = O(n^{\log_2 5}) = O(n^{2.32})$

Question3:

Consider the following algorithm, which takes as input an array A:

```
def printStuff(A):
    n = len(A)
    if n <= 4:
        return
    for i in range(n):
        print(A[i])
    printStuff(A[:n/3]) # recurse on first n/3 elements of A
    printStuff(A[2*n/3:]) # recurse on last n/3 elements of A
    return
```

What is the asymptotic running time of printStuff?

Ans: Recurrence relation:

$$T(n) = \begin{cases} \Theta(1), & if\ n \leq 4 \\ 2T\left(\frac{n}{3}\right) + \Theta(\text{n})\ , & if\ n > 5 \end{cases}$$

a =2 , b=3 , d =1

$b^d = 3^1 = 3$

As, a < b^d → T(n) = O(n^d)

So, T(n) = O(n)
