CS-600 WebCampusAdvanced Algorithm Design and Implementation

Homework 2

Recurrences: Solve the following recurrences using the substitution method. Sub- tract off a lower-order term to make the substitution proof work (if a lower- order term exists) or adjust the guess in case the initial substitution fails.

1. **T(n)=T(n−1)+2lgn. Our guess: T(n)=O(nlgn).**

   Show T(n) ≤ cnlgn for some constant c > 0.

   (Note: lg n is monotonically increasing for n > 0)

Answer:

If we use the guess: T(n)=O(nlgn), T(n) <=cnlgn, and assume

T(n-1)<=c(n-1)lg(n-1), (1<=n-1<=n) ---> (n>=2) works

Since lgn is monotonically increasing for n>0,

T(n)<=c(n-1)lg(n) +2lgn

=cnlgn-clgn+2lgn<=cnlgn

clgn>=2lgn

So for all c>=2,n>=2, the equation T(n) ≤ cnlgn works

2. **$T(n) = 5T(\frac{n}{4}) + n$. Our guess: $T(n) = O(n^{\log_4 5})$.**

   Show $T(n) \le cn^{\log_4 5}$ for some constant c > 0.

Answer:

If we use the guess: T(n)= O($n^{\log_4 5}$), T (n) ≤ $cn^{\log_4 5}$, and assume

T(n/4) <=c$(\frac{n}{4})^{log_4{}^5}$, (1<=n/4<=n)→n>=4 works

T(n)<=5c($(\frac{n}{4})^{log_4{}^5}$)+n,

=5c($n^{log_4{}^5}$/5) +n

=c$n^{log_4{}^5}$+n, the initial proof fails

Let T(n) <=c $n^{\log_4 5}$ –bn

T(n)= 5c($n^{log_4{}^5}$/5) –5b(n/4)+n<= c$n^{log_4{}^5}$-bn

So if b>=4, and we let b =4,we can see for all n>=4, c>0, the equation T(n)<= c $n^{\log_4 5}$ –4n works

3. **T(n)=T($\frac{n}{2}$)+T($\frac{n}{5}$)+t($\frac{n}{10}$) +n. Our guess: T(n)=O(n).**

   Show T(n) ≤ cn for some constant c > 0.

Answer:

If we use the guess: T(n)= O($n^{\log_4 5}$), and assume T(10/n) ≤ c(n/10), n/10>=1 works

T(n)<=cn/2+cn/5+cn/10+n,

=4cn/5+n

=($\frac{4c+5}{5}$)n

Let ($\frac{4c+5}{5}$)n<=cn

For n>=10, c>=5, the equation T(n)<=cn works

4. T(n)=8T(n/2)+n^2. Our guess: T(n)=O(n^2).

Show T (n) ≤ cn^2 for some constant c > 0.

Answer:

$T(n) \le 8(cn^2/4)+n^2$

$=2cn^2+n^2$

Obviously it is impossible that $(2c+1)\, n^2 \le n^2$

The guess of $T(n) \le cn^2-bn (b>0)$ and $T(n) \le cn^2-bn^2 (b>0)$ all failed

Use Master method:

$f(n)=n^2$

$n^{\log_b a} = n^{\log_2 8}$ we can find $\epsilon = 4$, $f(n)=O(n^{\log_2 8-\epsilon})$

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

$T(n) \le 8c(n^3/8)+n^2=cn^3+n^2 \le cn^3$ failed

So let $T(n) \le cn^3 - bn^2$

$T(n) \le 8c(n^3/8)-8b(n^2/4)+n^2$

if $8c(n^3/8)-8b(n^2/4)+n^2 \le cn^3 - bn^2$

$cn^3-2bn^2+n^2 \le cn^3 - bn^2$

$bn^2 \ge n^2$

$b \ge 1$

let b=1, then for all c>0, the equation $T(n) <= cn^3 - n^2$ works

Strassen's Algorithm: Use Strassen's algorithm to compute the following matrix product. Show all of the intermediate steps

$$\begin{pmatrix} 5 & 3 \\ 2 & -1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix}$$

Answer:

$S_1 = B_{12} - B_{22} = -2$

$S_2 = A_{11} + A_{12} = 8$

$S_3 = A_{21} + A_{22} = 1$

$S_4 = B_{21} - B_{11} = -6$

$S_5 = A_{11} + A_{22} = 4$

$S_6 = B_{11} + B_{22} = 5$

$S_7 = A_{12} - A_{22} = 4$

$S_8 = B_{21} + B_{22} = -1$

$S_9 = A_{11} - A_{21} = 3$

$S_{10} = B_{11} + B_{12} = 3$

$P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22} = -10$

$P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$=24

$P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}$=5

$P_4 = A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}$=6

$P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}$=20

$P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}$=-4

$P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}$=9

$C_{11} = P_5 + P_4 - P_2 + P_6$ =-2

$C_{12} = P_1 + P_2$=14

$C_{21} = P_3 + P_4$ =8

$C_{22} = P_5 + P_1 - P_3 - P_7$=-1

Result:$\begin{pmatrix} \mathbf{-2} & \mathbf{14} \\ \mathbf{8} & \mathbf{-1} \end{pmatrix}$

$$\begin{pmatrix} 5 & 3 & 4 & 3 \\ 2 & -1 & -2 & 0 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 \\ -4 & 3 \\ -2 & -3 \\ 1 & 4 \end{pmatrix}$$

Divide the matrix to be:

$C = (A1\ A2)\binom{B1}{B2}$

$A1=\begin{pmatrix} 5 & 3 \\ 2 & -1 \end{pmatrix}$ $B1=\begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix}$

$A2=\begin{pmatrix} 4 & 3 \\ -2 & 0 \end{pmatrix}$ $B2=\begin{pmatrix} -2 & -3 \\ 1 & 4 \end{pmatrix}$

$C=A1\cdot B1+A2\cdot B2$

Use the result of the first question $A1\cdot B1 = \begin{pmatrix} -2 & 14 \\ 8 & -1 \end{pmatrix}$

For $A2\cdot B2=\begin{pmatrix} 4 & 3 \\ -2 & 0 \end{pmatrix} \cdot \begin{pmatrix} -2 & -3 \\ 1 & 4 \end{pmatrix}$

$S_1=B_{12}-B_{22}=-7$

$S_2=A_{11}+A_{12}=7$

$S_3=A_{21}+A_{22}=-2$

$S_4=B_{21}-B_{11}=-3$

$S_5=A_{11}+A_{22}=4$

$S_6=B_{11}+B_{22}=2$

$S_7=A_{12}-A_{22}=3$

$S_8=B_{21}+B_{22}=5$

$S_9=A_{11}-A_{21}=6$

$S_{10}=B_{11}+B_{12}=-5$

$P_1 =A_{11} \cdot S_1 =A_{11} \cdot B_{12} -A_{11} \cdot B_{22}=-28$

$P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22} = 28$

$P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11} = 4$

$P_4 = A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11} = 0$

$P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} = 8$

$P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} = 15$

$P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12} = -30$

$C_{11} = P_5 + P_4 - P_2 + P_6 = -5$

$C_{12} = P_1 + P_2 = 0$

$C_{21} = P_3 + P_4 = 4$

$C_{22} = P_5 + P_1 - P_3 - P_7 = 6$

$$result = \begin{pmatrix} -2 & 14 \\ 8 & -1 \end{pmatrix} + \begin{pmatrix} -5 & 0 \\ 4 & 6 \end{pmatrix} = \begin{pmatrix} \mathbf{-7} & \mathbf{14} \\ \mathbf{12} & \mathbf{5} \end{pmatrix}$$

Exercise 2.3-6 Observe that the while loop in lines 5–7 of the INSERTION-SORT procedure in section 2.1 (Week 1, Slide 19, lines 5–7) uses a linear search to scan (backward) through the sorted subarray A[1..j – 1]. Can we use a binary search (see Exercise 2.3-5) instead to improve the overall worst-case running time of insertion sort to Θ(n lg n).

Answer:

If we use binary search to insert the element into the sorted ones, in the worst case, it will take O(nlgn) to find the right place, but when we need to insert the element, here in this case, it adopts arrays to store the input, so it will still take $O(n^2)$ to move the rest sorted elements because every elements will have to move right for one position in the worst case. It cannot improve the overall worst-case running time of insertion sort.

See the following code as explanation:

```
For (2<=j<=n) do {
    key = A[j], i=j-1, low = 1, high =j-1
    while (low<high) {
    mid = (A[mid]+A[high]])/2
    switch(mid==key) {
        case mid ==key
            break
        case mid < key
            high = mid-1
        case mid > key
            low = mid+1
    }
}
for (i=mid; i<=j-1; i++)
    A[i+1]=A[i] , A[mid] = key    ← in the worst case all the
elements move one position right, for n input, it will be O(n^2)
}
```