Fundamental Algorithms, Home work-2

Tulsi Jain

1. First Problem

1.1

$$Recurrence, \quad T(n) = 4T(\frac{n}{3}) + n$$

Here, $a = 4$ and b $= 3$, $f(n) = \Omega(log\frac{4}{3} - \epsilon)$. For some $\epsilon > 0$. So as per case 1, solution is $n^{log\frac{4}{3}}$. Let's prove this by substitution method.

Time Complexity of divide part is $O(n^{\log_3 4})$ and Time complexity of combine part is $O(n)$. Exponent of the former is greater than the later. Hence, master method is applicable. So, Time Complexity of this recurrence is $O(n^{\log_3 4})$. Let's prove the same by Substitution proof method.

Let's use induction method.

Induction Hypothesis: $T(k) \leq d_1(k^{\log_3 4}) - d_2 k \quad \forall \quad 1 \leq k < n$

$$UseDefinition, \quad T(n) = 4T(\frac{n}{3}) + n$$

$$Using \quad Induction \quad Hypothesis, \quad T(n) \leq 4(d_1(\frac{n}{3}^{\log_3 4}) - d_2\frac{n}{3}) + cn$$

$$T(n) \leq d_1(n^{\log_3 4}) - 4d_2\frac{n}{3} + cn$$

$$T(n) \leq \underbrace{d_1(n^{\log_3 4}) - d_2 n}_{\text{A}} - \underbrace{(d_2\frac{n}{3} - cn)}_{\text{B}}$$

A is equivalent to IH. But, for IH to hold true for n B needs to be non-negative. It means $d_2 \geq 3c$. We need to choose c sufficiently large enough.

Recurrent solution is $T(n) = O(n^{\log_3 4})$

1.2

$$Recurrence, \quad T(n) = 4T(\frac{n}{2}) + n^2$$

Here, $a = 4$ and b $= 2$, $f(n) = \Theta(n^{log\frac{4}{2}})$. So as per case 2 solution is $n^2 \log n$. Let's prove this by substitution method.

Time Complexity of divide part is $\Theta(n^{\log_2 4})$ and Time complexity of combine part is $\Theta(n^2)$. Exponent of the both terms are equal. Hence, master mathod is applicable. So, Time Complexity of this recurrence is $\Theta((\log_2 n)n^2)$. Let's prove the same by Substitution proof method.

Let's use induction method.

Induction Hypothesis: $T(k) \leq d_1(\log_2 k)k^2 - d_2 k \quad \forall \quad 1 \leq k < n$

$$UseDefinition, \quad T(n) = 4T(\frac{n}{2}) + n^2$$

$$Using \quad Induction \quad Hypothesis, \quad T(n) \leq 4(d_1(\log_2 \frac{n}{2})\frac{n^2}{4}) + cn^2$$

$$T(n) \leq 4(d_1(\log_2 n - \log_2 2)\frac{n^2}{4}) + cn^2$$

$$T(n) \leq 4(d_1(\log_2 n - 1)\frac{n^2}{4}) + cn^2$$

$$T(n) \leq d_1(\log_2 n - 1)n^2 + cn^2$$

$$T(n) \leq \underbrace{d_1 \log_2 nn^2}_{\text{A}} - \underbrace{(d_1 n^2 - cn^2)}_{\text{B}}$$

A is equivalent to IH. But, for IH to hold true for n B needs to be non-negative. It means $d_1 \geq c$. We need to choose d_1 sufficiently large enough.

Recurrent solution is $T(n) = O(n^2 \log_2 n)$

1.3

$$Recurrence, \quad T(n) = 4T(\frac{n}{2}) + n^2 \log_2 n$$

Here, $a = 4$ and b = 2, $f(n) = \Omega(n^2)$. But $f(n) = \Omega(n^{2+\epsilon})$ is not true for some $\epsilon > 0$ Master method is not applicable for this recurrence. Let's calculate by pictorial method.

1, 4, 16, 64   2, 4, 8

2^{K+1}

PAGE No

DATE : / /201

1, 3 - Part

$T(n) \longrightarrow n^2 \log_2^n$

T(n/2) T(n/2) T(n/2) T(n/2) $\longrightarrow 4 \times \left(\frac{n}{2}\right)^2 \log^{n/2}$

$$= \sum_{i=1}^{K} \frac{4^{(K-1)} n^2}{2^{(K-1)*2}} \log^{n/2^{K-1}}$$

$$= 2^{(\cancel{2}K - \cancel{2} - K\cancel{2K})} n^2 \sum_{i=1}^{K} \log^n - \log^{2^{(K-1)}}$$

$$= 2^{0} n^2 \sum_{i=1}^{K} \log^n - (K-1)$$

$\cancel{\log}$

$$= 2^{0} n^2 \left[\left(\log^n\right)^2 + \log^n - \frac{\log n\,(\log^n + 1)}{2} \right]$$

$$= \left[\frac{(\log^n)^2}{2} + \frac{\log^n}{2} \right]$$

$$\boxed{= O\left(n^2 \log^n \log^n\right)}$$

2. Second Problem

Here, master theorme is not valid as $b = 1$.
T(0) = c
T(1) = ac +k
T(2) = a(ac + k) + k = a$^2c + ak + k$
T(3) = a(a$^2c + ak + k) + k = a^3c + a^2k + ak + k$
T(n) = a$^nc + (a)^{n-1}k + ... + a^2k + ak + k$

Looks like T(n) is exponential in a. Let's make a guess that solution is $\Theta(a^n)$, exponential.

Let's use induction method.
Induction Hypothesis: $T(k) \leq d_1(a^k) - d_2 \quad \forall \quad 1 \leq k < n$

$$UseDefinition, \quad T(n) = aT(n-1) + k$$

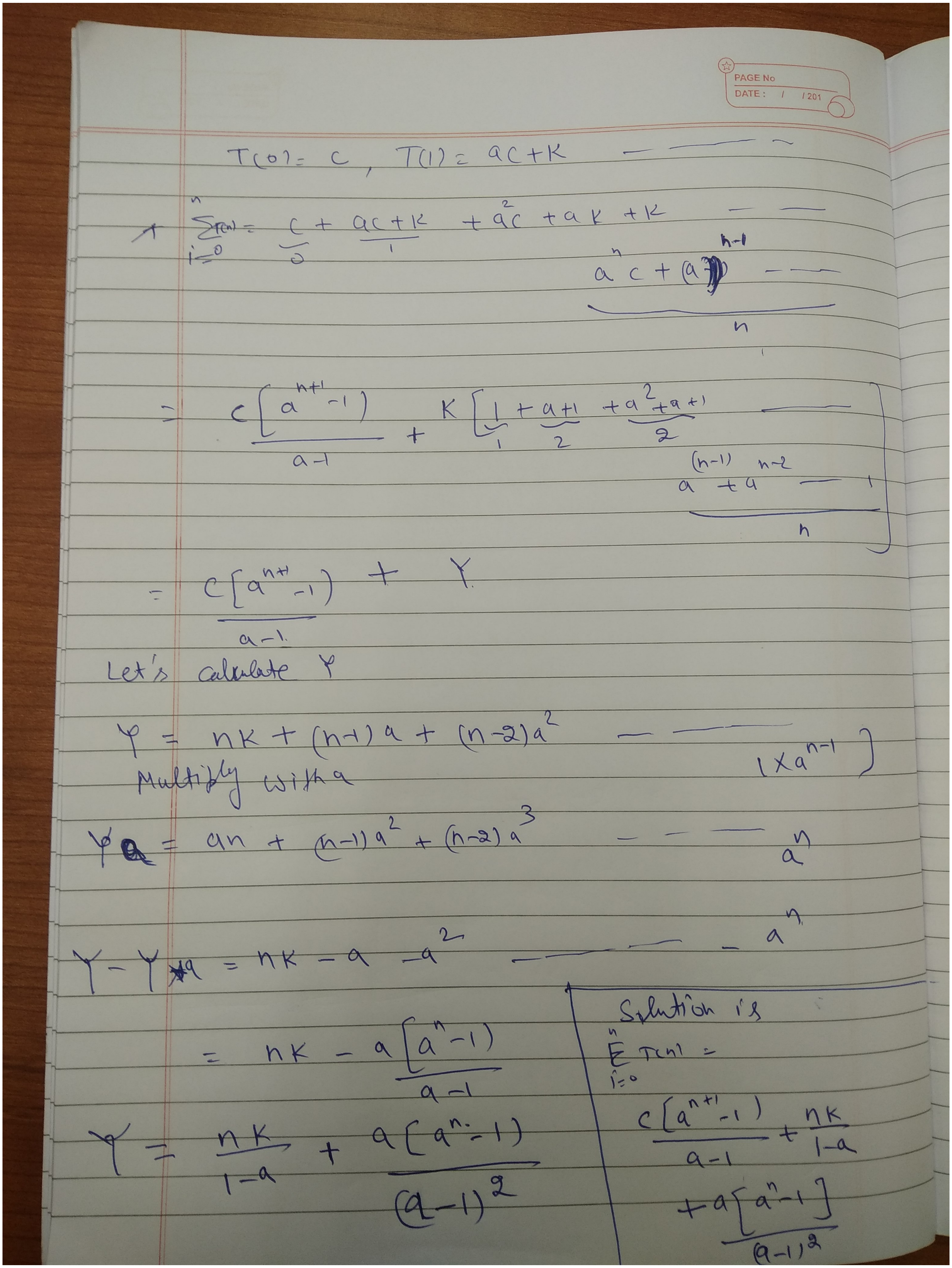
$$Using \quad Induction \quad Hypothesis, \quad T(n) \leq a(d_1a^{n-1} - d_2) + k$$

$$T(n) \leq d_1(a^n) - ad_2 + k$$

$$T(n) \leq \underbrace{d_1a^n}_{\text{A}} - \underbrace{(ad_2 - k)}_{\text{B}}$$

A is equivalent to IH. But, for IH to hold true for n B needs to be non-negative. It means $ad_2 \geq k$. We need to choose a and d_2 sufficiently large enough.

Solution is $T(n) \leq d_1(a^n) - d_2$
Recurrent solution is $T(n) = O(a^n)$



$T(0) = c$, $T(1) = ac + K$ ...

$$\sum_{i=0}^{n} T(n) = \underbrace{c}_{0} + \underbrace{ac+K}_{1} + a^2c + aK + K \cdots \underbrace{a^n c + (a)^{n-1} \cdots}_{n}$$

$$= \frac{c[a^{n+1}-1)}{a-1} + K\left[\underbrace{1}_{1} + \underbrace{a+1}_{2} + \underbrace{a^2+a+1}_{2} \cdots \underbrace{a^{(n-1)} + a^{n-2} \cdots 1}_{n}\right]$$

$$= \frac{c[a^{n+1}-1)}{a-1} + Y$$

Let's calculate Y

$$Y = nK + (n-1)a + (n-2)a^2 \cdots 1 \times a^{n-1}]$$

Multiply with a

$$Ya = an + (n-1)a^2 + (n-2)a^3 \cdots a^n$$

$$Y - Ya = nK - a - a^2 \cdots - a^n$$

$$= nK - \frac{a[a^n-1)}{a-1}$$

$$Y = \frac{nK}{1-a} + \frac{a[a^n-1)}{(a-1)^2}$$

Solution is

$$\sum_{i=0}^{n} T(n) = \frac{c[a^{n+1}-1)}{a-1} + \frac{nK}{1-a} + \frac{a[a^n-1]}{(a-1)^2}$$

Figure 2.1: Solution 2

3. Third Problem

Algorithm 1 Iterative Binary Search

```
function ITEBINARYSEARCH(A, l, r, x)    ▷ where Array A, leftmost l, rightmost r, find x
    while r ≥ l do
        mid = l + (r - l)/2;
        if A[mid] > x then
            r = r - 1;
            IteBinarySearch(A, l r, x)
        else
            if A[mid] > x then
                l = l+1;
                IteBinarySearch(A, l r, x)
            else
                return mid;
            end if
        end if
        return not found;
    end while
end function
```

Invariant: At any point of time, before each iteration of the while loop either x is present between l and r index (inclusive both) or x is not present in the original array at all.
Condition: l $\leq$ r

Proof of Correctness

Initialization

At starting, $l = 1$ and $r = A.length$. So A = [1, ... n] signify the whole array. Either x is present in A or is not present in A is completely a valid statement.
Hence loop invariant holds in the beginning.

Maintenance

Assumption:
Invariant holds for k -1 iteration
After $(k-1)th$ iteration, index r to j (inclusive both) represents a array of size $n-(j-1)$. As per assumption either x exist between index r to j or does not exist in original array at all.
In kth iteration, either element is equal to x or is not. If it does not match, it means either x exist between index r to j or does not exist in original array at all.

Hence, invariant also holds during maintaince.

Termination

At termination, l > r, hence index l to r represents a empty array. To loop invariant to be hold true either x be present between l and r index or not be present in original array, A. x can't be present between l and r as it represents a empty array. So x be not present in array, A. This is also response of the pseudo code. Hence loop invariant hold at termination.
Condition is false as l is exceeded r.

Loop invariant is valid at all three steps, before iteration, during iteration and after iteration. Also this loop invariant also shows at the termination tempMax in maximum of in array A (hence useful).

4. Fourth Problem

Algorithm 2 Recursive binary Search

```
function RecBinarySearch(A, l, r)        ▷ where Array A, leftmost l, rightmost r, find x
    while l ≥ r do
        mid = l + (r - l)/2;
        if A[mid] > x then
            l = mid;
            RecBinarySearch (A, l, r, x);
        else
            if A[mid] > x then
                r = mid;
                RecBinarySearch (A, l, r, x);
            else
                return mid;
            end if
        end if
    end while
    return not found;
end function
```

Recurrence Formation:

$$T(n) = \left\{ \begin{array}{c} \text{c, if n = 1} \\ \text{T(n/2), if n > 1} \end{array} \right\}$$

Here, $a = 1$ and b = 2, $f(n) = \Theta(n^{log1}) = Constant$. So as per case 2 worst case running time is $\log n$.
Worst case would occur when element does not exist in array. In this case, iteration would occur logn times, until while condition fails to execute.

5. Fivth Problem

Pseudo Code for inversion count

Algorithm 3 Merge Sort

```
function INT FINDINVERSIONCOUNT(A, l, r)
    while l > r do
        mid = l + (r − l)/2
        count_1 = FindInversionCount(A, l, mid)
        count_2 = FindInversionCount(A, mid + 1, r)
        count_3 = count_1 + count_2 + MERGE(A, p, q, r)
    end while
    return count_3 end function
     int Merge A, p, q, r                    ▷ Where A - array, p - left, q - middle, r - right
     n_1 = q − p + 1;   n_2 = r − q;   count = 0;
     Let L[1 . . . n_1] and R[1 . . . n_2] be new arrays
     for i = 1 to n_1 do
         L[i] = A[p + i − 1]
     end for
     for j = 1 to n_2 do
         R[j] = A[q + j]
     end for
     i = 0;   j = 0
     for k = p to r do
         if i < n_1   &&   i < n_2 then
             if L[i] < R[j] then
                 A[k] = L[i]
                 i = i + 1
             else if L[i] > R[j] then
                 Comment: If left array has bigger value then, increase inversion count.
                 count++;
                 A[k] = R[j]
                 j = j + 1
             else
                 A[k] = R[j]
                 j = j + 1
             end if
         else if i >= n_1 then
             A[k] = R[j]
             j = j + 1
         else if j >= n_2 then
             A[k] = L[i]
             i = i + 1
             Comment: If right array has empty before left then increase inversion by n_2 for
each element left in left array.
             count = count + n_2
         end if
     end for
     return count
```

Inversion count is equal to number of time $A[i] > A[j] \quad when \quad i < j$.

Recurrence Formation:

$$T(n) = \left\{ \begin{array}{c} \text{c, if n = 1} \\ \text{2T(n/2) + n, if n > 1} \end{array} \right\}$$

Here, $a = 2$ and b $= 2$, $f(n) = \Theta(n^{log^{2}_{2}})$. So as per case 2 solution worst case running time is $n \log n$. Here best, average and worst running time are equal as merge would be run for entire length irrespective of the any condition.