Ch:1 Introduction

Analysis of Algorithms (Background)

Q. given a number n, write a function to find sum of first n natural number

I/P : n = 3
O/P : 6

I/P : n = 5
O/P : 15

Sol

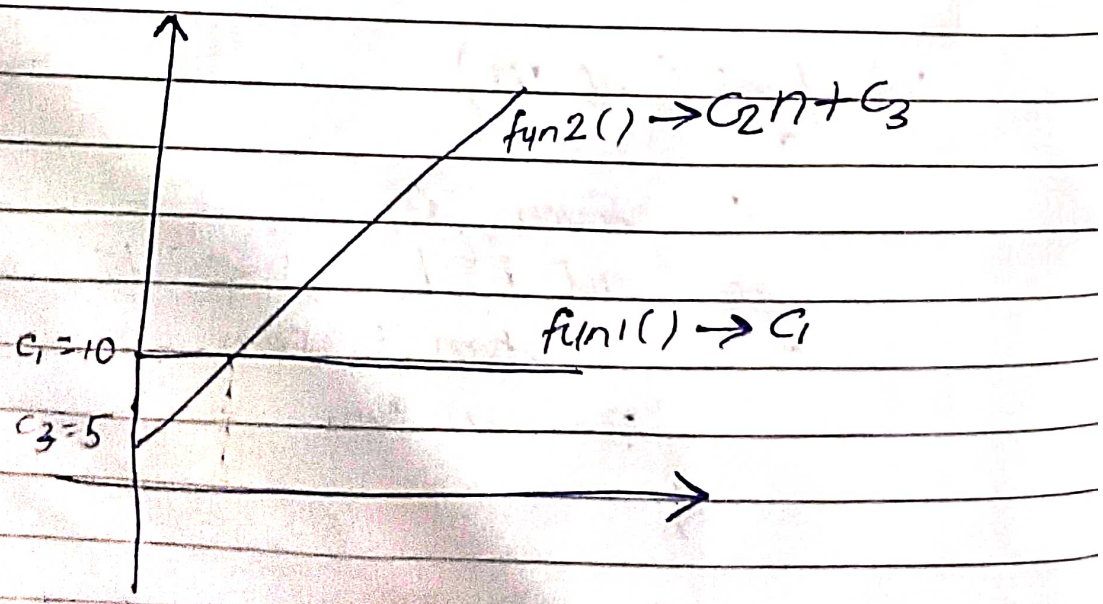
```
int fun1(int n)
  {
    return n*(n+1)/2;
  }

int fun2(int n)
  {
    int sum = 0
    for(int i=1; i<=n; i++)
        sum = sum+i;
    return sum;
  }
```

```
int fun3(int n)
{
    int sum = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= i; j++)
            sum++;
    return sum;
}
```

→ expression time taken of the above function

fun1() → C_1
fun2() → $C_2n + C_3$
fun3() → $C_4n^2 + C_5n + C_6$

→ comparison of fun1() & func2()



Let assume that there is two function

$C_2h + C_2$

$2n + 5 \geq 10$

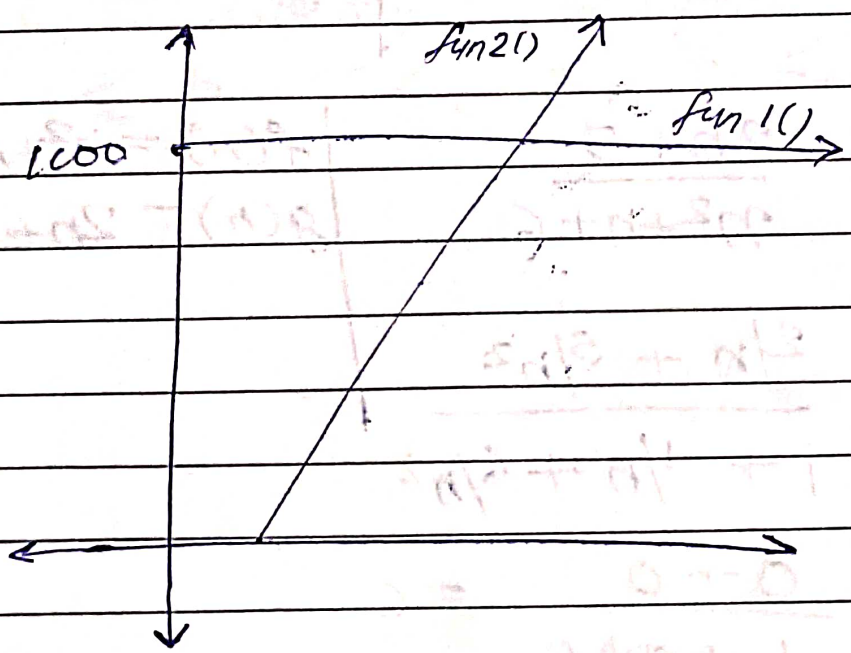
$n \geq 2.5$

$n \geq 3$

eg

fun 1() = 1000

fun 2() = n + 1



$n + 1 \geq 1000$

$n \geq 999$

Order of Growth

A function $f(n)$ is said to be growing faster than $g(n)$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

OR

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

$n \geq 0$

Time taken ≥ 0

$f(n), g(n) \geq 0$

Eg

$$\lim_{n \to \infty} \frac{2n+5}{n^2+n+6}$$

$f(n) = n^2+n+6$

$g(n) = 2n+5$

$$\lim_{n \to \infty} \frac{2/n + 5/n^2}{1 + 1/n + 6/n^2}$$

$$\lim_{n \to \infty} \frac{0+0}{1+0+0} = 0$$

hence it proves that $f(n)$ is growing faster than $g(n)$

Page No.:
Date:

Direct way

1. Ignore lower order terms
2. Ignore lower leading constant

→ How do we know which terms are lower order?

$c < \log\log n < \log n < n^{1/3} < n^{1/2} < n < n^2 < n^2 < n^4 < n^5 < n^6$

Asymptotic Notations

Best, Average and Worst cases

Asymptotic Notations

```
int getSum (int arr[], int n)
{
    if (n % 2 != 0)
        return 0;

    for (int i = 0; i < n; i++)
        sum = sum + i;
    return sum;
}
```

Best case : constant

Average case : linear

under the assumption that each and add is equaly likely

Worst case :

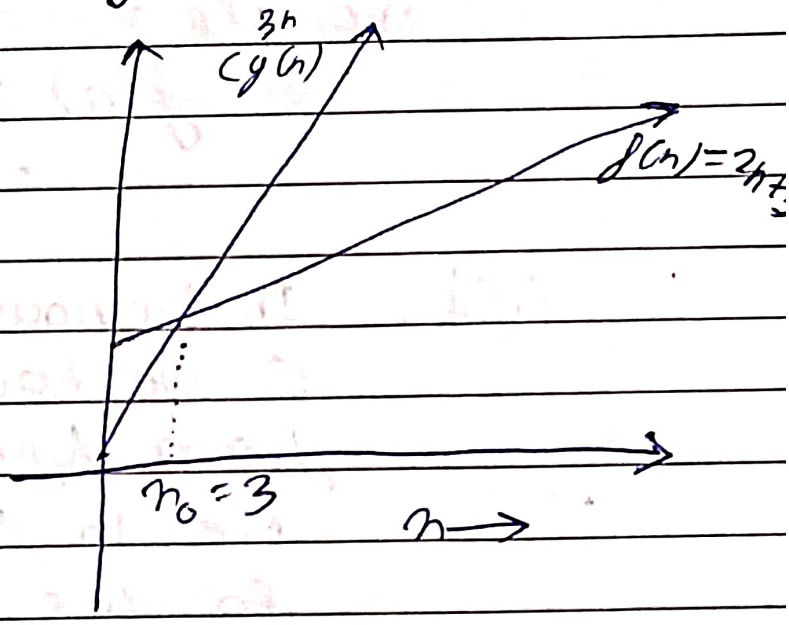
Big O : Represents exact bond or upper bond

Theta : Represent exact bond.

Omega : Represent exact or lower bond

Big O Notation (Upper Bond or Order Growth)

We say $f(n) = O(g(n))$ if there exist constant c and n_0 such that $f(n) \leq c\,g(n)$ for all $n \geq n_0$



example $f(n) = 2n+3$ can be written as $O(n)$

$f(n) \leq c\,g(n)$ for all $n \geq n_0$

$(2n+3) \leq cn$ for all $n \geq n_0$

$c = 3$ [go to $f(n)$ take constant from higher order term and just add one in it. eg $(2+) = 3$]

$2n+3 \leq 3n$

$3 \leq n$

$n_0 = 3$

just put the n_0 & greater value of n_0 a

Omega Notation (lower bond)

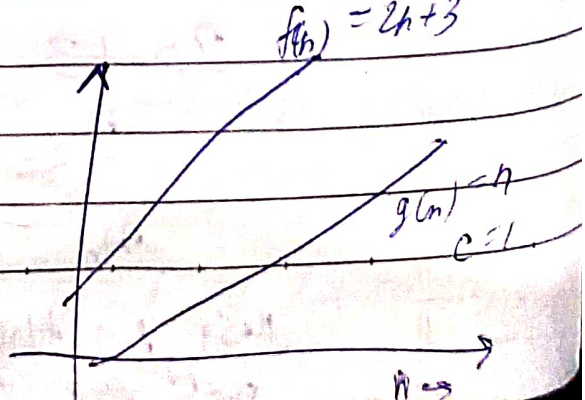
$f(n) = \Omega(g(n))$ if there exist positive constants c & n_0 such that $0 \leq c\,g(n) \leq f(n)$ for all $n \geq n_0$

Example:

$$f(n) = 2n+3$$
$$= \Omega(n)$$

Proof: In previous section to find c we have taken constant from higher order term i.e in our case is n so we take 2 and then subtract by 1

$$c = (2-1)$$
$$c = 1$$
$$c\,g(n) = n$$
$$n \leq 2n+3$$
$$n_0 = 0$$



① $\left\{\frac{n}{4}, \frac{n}{2}, 2n, 3n, 2n+3, n^2+2n^2 \ldots n^3\right\} \in \Omega(n)$

② If $f(n) = \Omega(g(n))$
then $g(n) = O(f(n))$

③ Omega notation is useful when we have lower bond on time complexity.

Theta Notation (Exact order of growth)

$f(n) = \Theta(g(n))$ if there exist positive constant C_1, C_2 and n_0 such that

$$0 <= C_1 g(n) <= f(n) <= C_2 g(n)$$

for all $n >= n_0$

Example

$$f(n) = 2n + 3 \quad \Theta(n)$$

$$1 \times n \leq 2n + 3 \leq 3 \times n$$

$n >= 0$ $\quad$ $n >= 3$

$n_0 = 3$

$C_1 = 1$

$C_2 = 3$

1. If $f(n) = \Theta(g(n))$ then $f(n) = O(g(n))$ & $f(n) = \Omega(g(n))$ and $g(n) = O(f(n))$ & $g(n) = \Omega(f(n))$

2. Theta is useful to represent time complexity when we know exact bound for example time-complexity to find sum, max & min in an array is $\Theta(n)$

③ $\left\{ \frac{n^4}{4}, \frac{n^2}{4} \ldots\ldots 2n^2, 2n^2 + 100n, 4n^2 + 2n\log n + 30 \ldots\ldots \right\} \in \Theta(n^2)$

Analysis of common loop

eg
```
for (int i=n ; i<0; i=i-c)
{
/*
   Θ(1) work
because it is not dependent
on n.
*/
}
```

eg2
```
for (int i=1; i<n; i=i*c){
{
// Θ(logn)
}
```

$1, c, c^2, c^3 \ldots c^{k-1}$

$c^{k-1} < n$

$k-1 < \log_c n$

$k < (\log_c(n)+1)$

eg3
```
for (int i=n ; i>=0; i = i/c)
    {
      // some θ(logn)
    }
```

eg4
```
for (int i=2 ; i<n; i = pow(i, c))
    {
        θ (log logn)
    }
```

$2, 2^c, (2^c)^c \ldots\ldots$

$2, 2^c, 2^{c^2} \ldots 2^{c^{k-1}}$

$2^{c^{k-1}} < n$

take log both side

$c^{k-1} < \log_2 n$

$k-1 < \log_c \log_2 n$

$k < \log_c \log_2 n + 1$

eg5
```
void fun(int n) {
  for (int i=0; i<n; i++)  ] θ(n)
  //
  for (int i=0 ; i<n ; i=i*2) ] θ(log n)
  for (int i=0; i<100 ; i++) ] θ(1)
```

$\theta(n) + \theta(\log n) + \theta(1)$

✓ ✗ ✗

eg 6 void fun (int n)

{

$\theta(n \log n)$ [for (int i = 0; i < n; i++)

for (int j = 1; j < n; j = j * 2)

for (int i = 0; i < n; i++)

for (int j = 1; j < n; j++)] $\theta(n^2)$

$\theta(n \log n) + \theta(n^2)$

Analysis of Recursion

```
Void fun (int n)
{
    if (n<=1)
        return;
    for (int i=0; i<n; i++)
        print ("GFG");
    fun (n/2);
    fun (n/2);
}
```

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

$T(1) = C$

```
Void fun (int n)
{
    if (n<=0)
        return 0;
    print (n);
    fun (n-1)
}
```

$T(n) = T(n-1) + C$

Recursive — Non-recursive

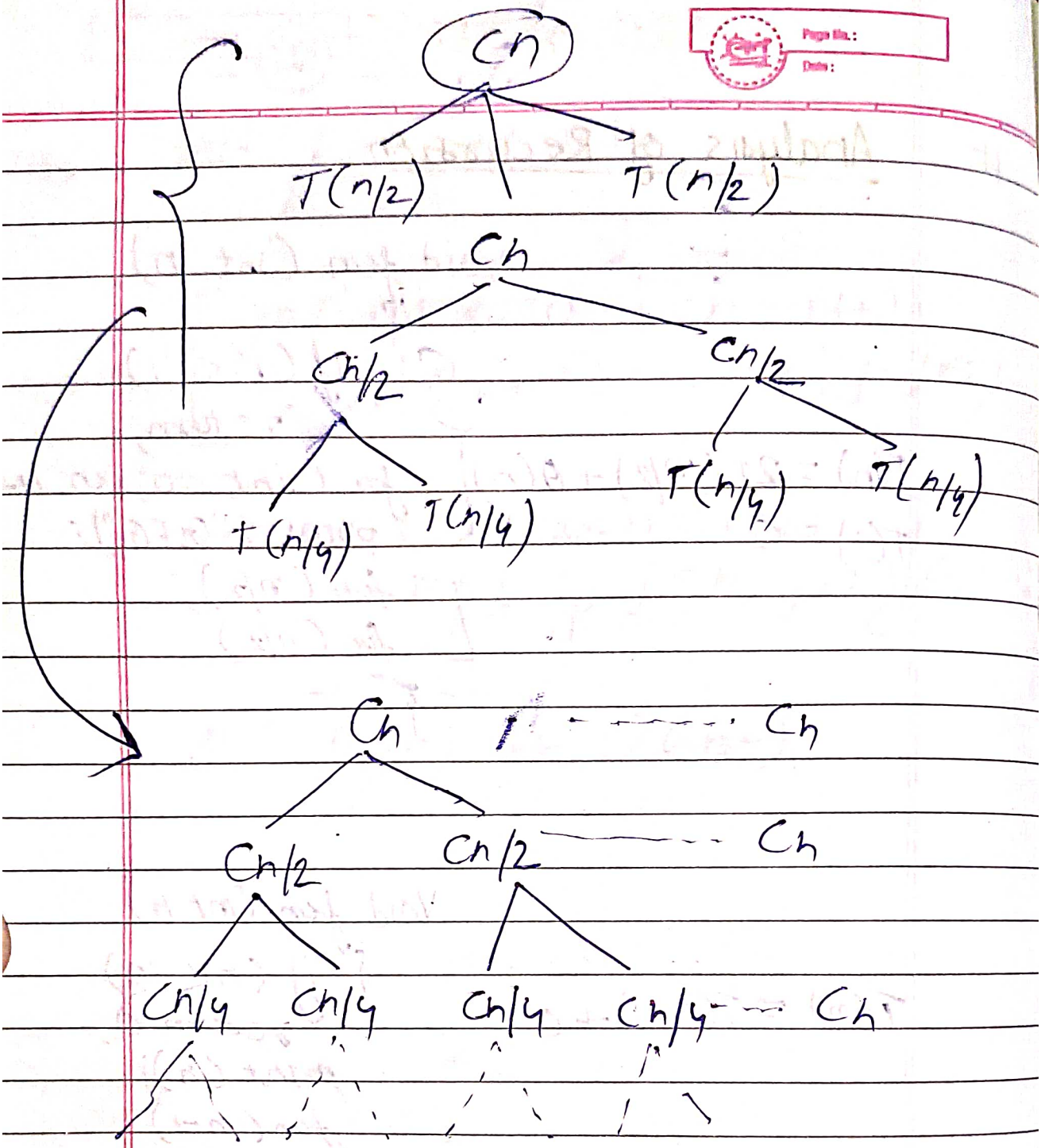
$T(n) = 2T(n/2) + Cn$

$T(1) = C$

Recursion Tree Method

→ We write non-recursive part as root of tree & recursive parts as children.

→ We keep expanding children until we see a pattern.



$$\underbrace{C_n + C_n + C_n + \dots C_n}_{\log n}$$

$C_n \times \log_2 n$

$O(n \log_2 n)$

Ch : 2 Mathematics

Mathematics

1. Finding number of digits in a number.

Sol

iterative solution

```
int countDigit(long n)
{
    int count = 0;
    while (n != 0) {
        n = n/10;
        ++count;
    }
    return count;
}
```

recursive solution

```
int countDigit(long n)
{ if (n == 0)
      return 0;
  return 1 + countDigit(n/10);
}
```

logarithmic solution

```
int countDigit(long n) {
    return floor(log10(n)+1);
```

2. Arithmetic And Geometric Progression

2, 4, 6, 8, 10 ....

$a = 2$
$d = 2$
$a + d$
$a + 2d$
$\vdots$
$a + (n-1)d$

$avg = \frac{sum}{n}$

$= avg \times n.$

$\left(\frac{FT + LT}{2}\right) \times n$

$= \frac{n}{2}(a + a + (n-1)d)$

$= \frac{n}{2} \cdot (2a + (n-1)d)$

2, 4, 8, 16, 32 .....

$r = 2, \quad a \cdot r, \quad ar^2, \quad \ldots\ldots \quad ar^{n-1}$ (n^{th} term)

$sum = \frac{a(1 - r^n)}{1 - r}$

3. Quadratic Equation

$$ax^2 + bx + c = 0$$

$$D = b^2 - 4ac$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$D < 0$, imagnary roots
$D = 0$, two equal roots
$D > 0$, two distinct roots

4. Mean & median

mean: The mean is found by adding up all of the given data and dividing by the number of data entries.

example: The mean of 4, 1 and 7 is
$(4 + 1 + 7)/3 = 12/3 = 4$

Median: The middle number, found by ordering all data points and picking out the one in the middle (or if there are two middle numbers, taking the mean of those two numbers).

example: The median of 4, 1, 4 7 is 4 because when the no. are put in order (1, 4, 7) the no. 4 is in the middle.

5. Prime Number

2, 3, 5, 7, 11 ... are the prime number.

→ Every prime number can be represented in form of $6n+1$ or $6n-1$ except 2 and 3, where n is natural number.

→ Two & those are only two consecutive natural numbers which are prime too.

6. LCM And HCF

factor : All the number that divide a number completely i.e without leaving any remainder are called factors of that number.

example : 24 is completely divisible by 1, 2, 3, 4, 6, 8, 12, 24

Each of these number is called a factor of 24 & 24 is called a multiple of each of these numbers.

LCM : The least common multiple or LCM of two or more number is the smallest number other Zero. that's multiple of each number.

LCM of 4 and 6?

example: Multiple of 4: 4, 8, 12, 16, 20, 24...
Multiple of 6: 6, 12, 18, 24....
common multiple: 12, 24
Least common Multiple: 12

HCF: HCf of two or more given numbers is the highest number which exactly divides all the no.

HCf of 12 & 16?
example: factor of 12: 1, 2, 3, 4, 6, 12
factor of 16: , 1, 2, 4, 8, 16
common factor: 1, 2, 4
HCF: 4

7 factorials:

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

$= n \times (n-1) \times (n-2) \ldots 1$

* $0! = 1$

8. Permutation & combination basics

Permutation: Permutation is defined as arrangement of r things that can be done out of total n things.

This is denoted by ${}^nP_r$ which is equal to $n!/(n-r)!$

$$ {}^nP_r = \frac{n!}{(n-r)!} $$

Combination: Combination is defined as selection of r things that can be done out of total n things.

This is denoted by ${}^nC_r$ which is equal to $n!/r!(n-r)!$

$$ {}^nC_r = \frac{n!}{r!(n-r)!} $$

9. Modular Arithmetic

a % b = c

a is divided by b it give the remainder c.

eg. 7 % 2 = 1

17 % 3 = 2

DO Quiz & Problem