Page No.
Date

Analysing the Algorithm

Characteristics of Algorithm

1) Input - 0 or more
2) Output - 1 result atleast
3) Definiteness - Solvable ex- $\sqrt{-1}$ x
4) Finiteness
5) Effectiveness - Every setep must be effective
   Not unnecessary

How to write algorithm

```
Algorithm swap (a, b) {
        temp = a;
        a = b;
        b = temp;
}
```

How to analyze algorithms

1) Time
2) Space
3) Network
4) Power consumption
5) CPU registers

```
algorithm swap (a, b) {
        temp = a;          — 1
        a = b;             — 1
        b = temp;          — 1
}
```

$f(n) = 3$

Scanned by TapScanner

$$x = \underbrace{\underbrace{5 * a}_{1} + \underbrace{6 * b}_{1}}_{1} ; \quad — 1$$

Space = $S(n) = 3$

Algorithm sum (A, n) {

s = 0

for (i=0; i<n; i++) { — 1 — n + 1

n — s = A[i] + s;

}

return s; — 1

}

n+1 n

$$f(n) = n + 1 + n + 1 + 1$$
$$= 2n + 3$$
$$= O(n)$$

Space = A — n

n — 1

s — 1

i — 1

$S(n) = n + 3$

$O(n)$

Algorithm (A, B, n) {

for (i = 0; i<n; i++) { — n+1

(n+1) — for (j = 0; j<n; j++) {

n — n — c[i][j] = A[i][j] + B[i][j];

}

}

$$f(n) = 2n^2 + 2n + 1$$
$$O(n^2)$$

Time complexity

1)
```
for (i=0; i<n; i++) {
    // statement          — O(n)
}
```

2)
```
for (i=n; i>0; i--) {
    // code               — O(n)
}
```

3)
```
for (i=1; i<n; i=i+2) {
    // code
}
```
$f(n) = n/2$

$O(n)$

4)
```
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        // code           — O(n^2)
    }
}
```

5)
```
for (i=0; i<n; i++) {
    for (j=0; j<i; j++) {
        // statement
    }
}
```

statement executed

$= 0 + 1 + 2 + \ldots n$

$= \frac{n(n+1)}{2} = O(n^2)$

Page No.
Date

6) $p = 0;$

```
for (i=1; p<=n; i++) {
        p = p+i;
}
```

i	p
1	$0+1=1$
2	$1+2=3$
3	$1+2+3$
4	$1+2+3+4$

$1+2+3+4+ \ldots k > n$

$\frac{k(k+1)}{2} > n$

$k^2 > n$

$k > \sqrt{n}$ $O(\sqrt{n})$

7)

```
for (i=1; i<n; i=i*2) {
        // statement
}
```

Over when $i > n$

$2^k > n$

$k \geq \log_2 n$, $O(\log_2 n)$

No. of times

$= \text{ceil}(\log_2 n)$

Page No.
Date

8) for (i = n; i >= 1; i = i/2) {
// code
}

Over when
$i < 1$

$\frac{n}{2^k} < 1$

$n < 2^k$

$k = \log_2 n$, $O(\log_2 n)$

9) for (i = 0; i * i < n; i++) {
// code
}

$i * i > n$

$i^2 > n$

$i > \sqrt{n}$, $O(\sqrt{n})$

10) p = 0;
for (i = 1; i < n; i = i * 2) {
p++;
}
for (j = 1; j < p; j = j * 2)
{ // code
}

$j >= p$

~~j >= do~~

Scanned by TapScanner

ans = $\log p$

$= \log \log n$

ii)
```
for (i=0; i<n; i++) {
    for (j=1; j<n; j=j*2) {
        // Code
    }
}
```

n

$\log n$

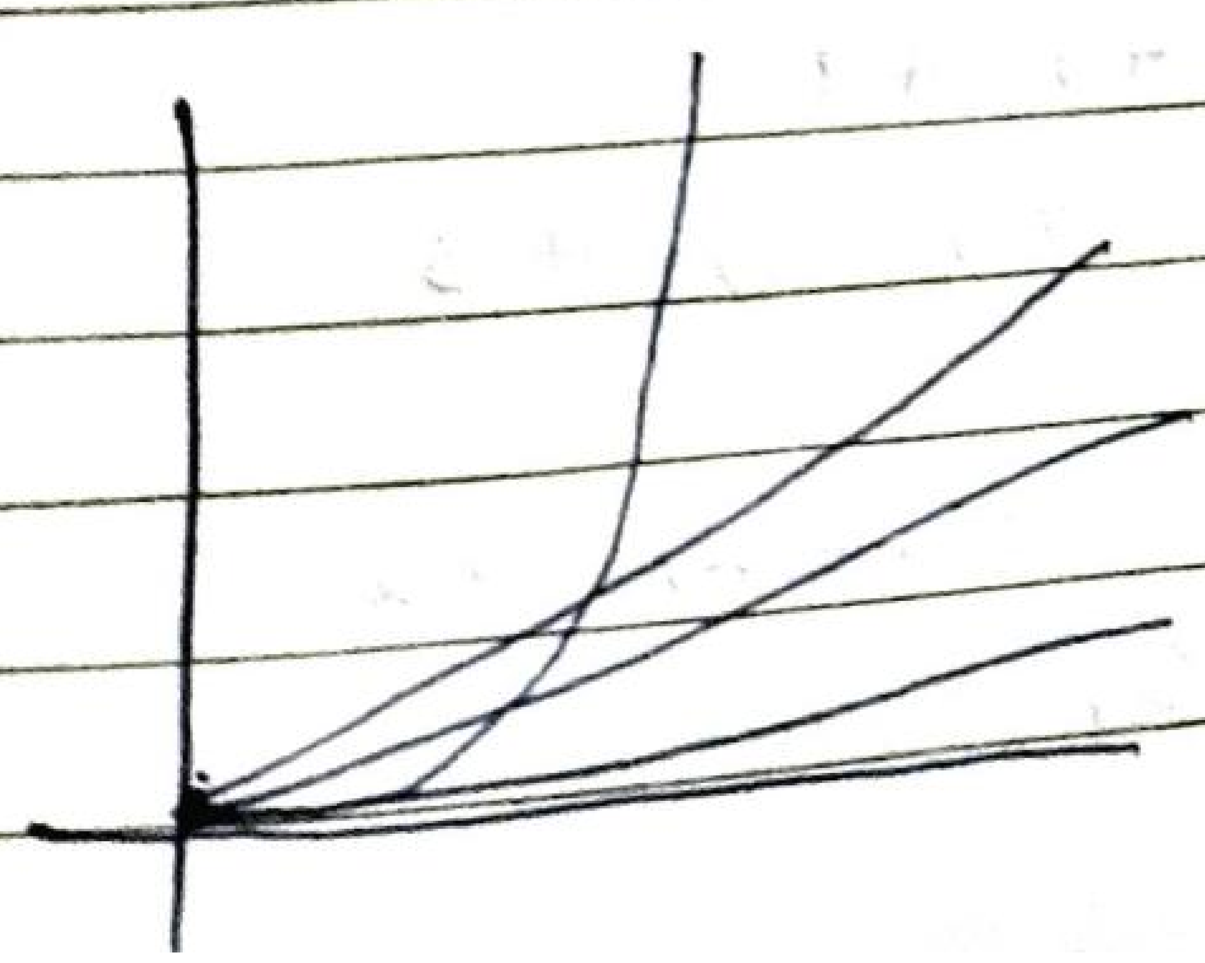
$O(n \log n)$

```
Algorithm Test (n)
    if (n < 5) {
        printf("%d", n);
    }
    else {
        for (i=0; i<n; i++) {
            printf("%d", i);
        }
    }
```

Best - $O(1)$

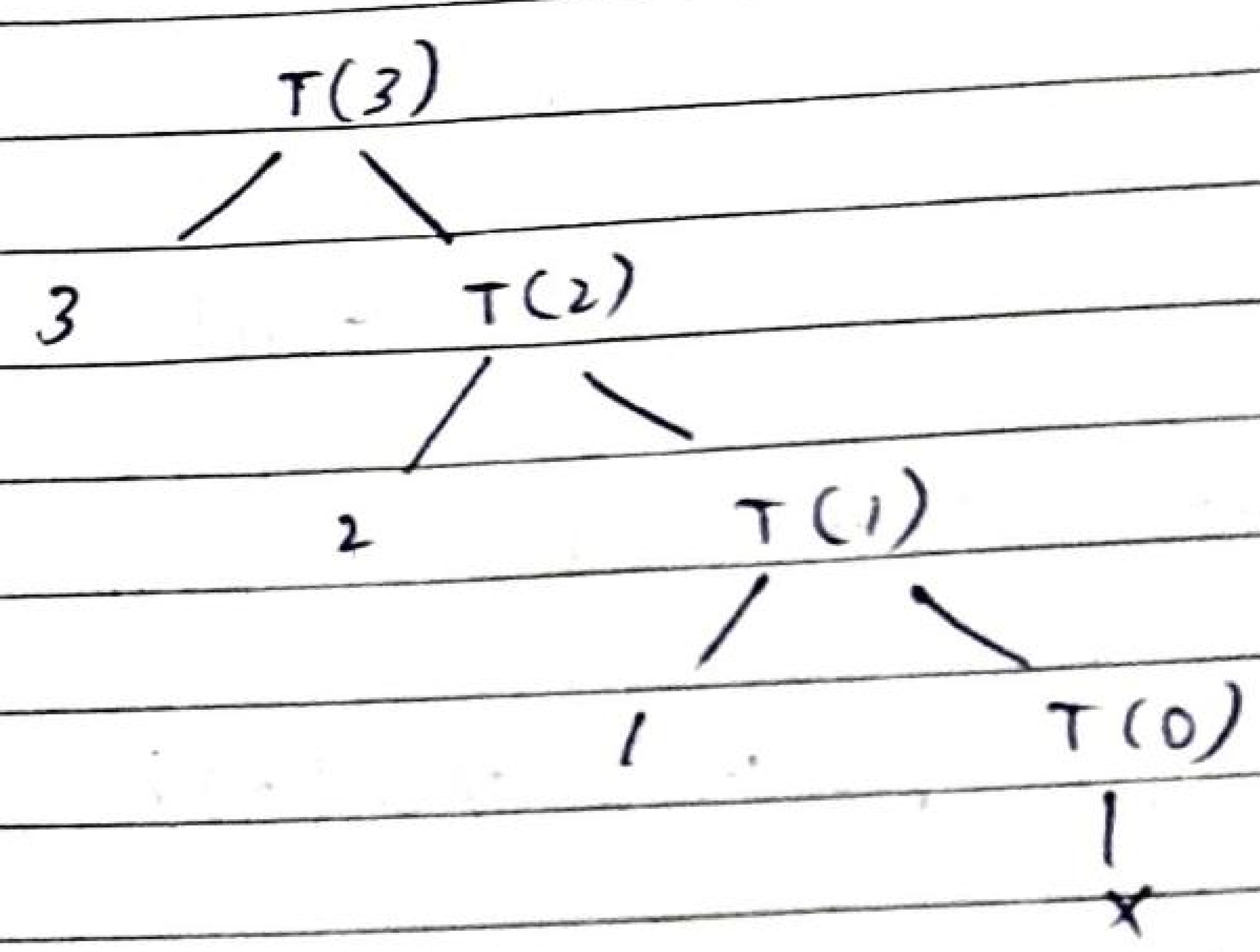
worst = $O(n)$

$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3$
$< 2^n < n^n$



Recurrence relation

```
void test (int n) {                 — T(n)
    if (n>0) {
        printf ("%d", n);           — 1
        test (n-1);                 — T(n-1)
    }
}
```



O/p = 3 2 1

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

$$T(n) = \begin{cases} 1 & , n = 0 \\ T(n-1) + 1 & , n > 0 \end{cases}$$

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

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

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

Similarly. $T(n) = T(n-3) + 3$

$\vdots$

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

if $n-k=0$, $n=k$,

$T(n) = T(0) + n$

$= 1 + n$

$O(n)$

2)

```
void Test (int n) {
    if (n > 0) {                              - T(n)
        for (i=0; i<n; i++) {
            printf("%d", n);                  } n
        }
        Test (n-1);                           - T(n-1)
    }
```

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

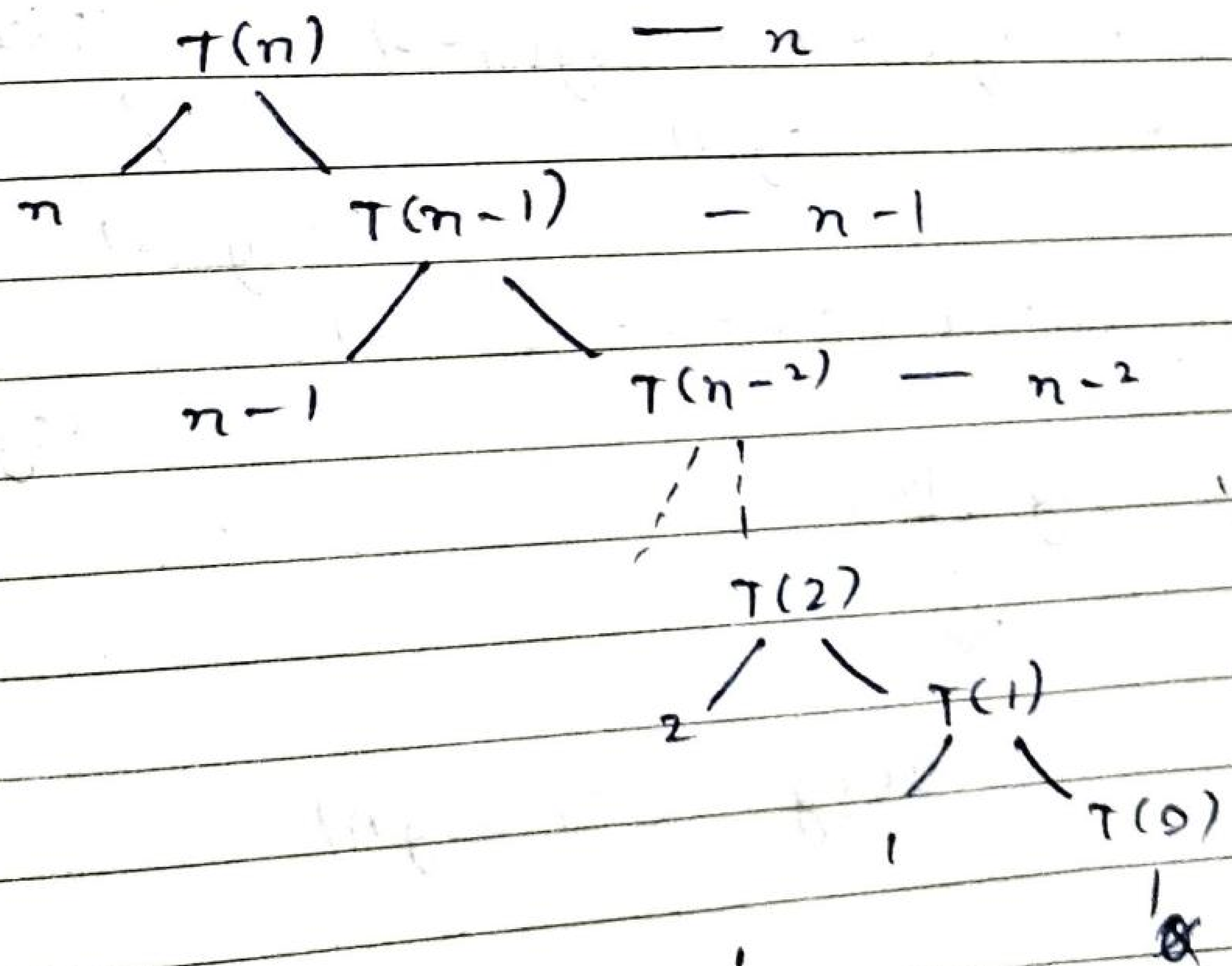
$T(n-1) = T(n-2) + (n-1)$

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

similarly,

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

Can be solved



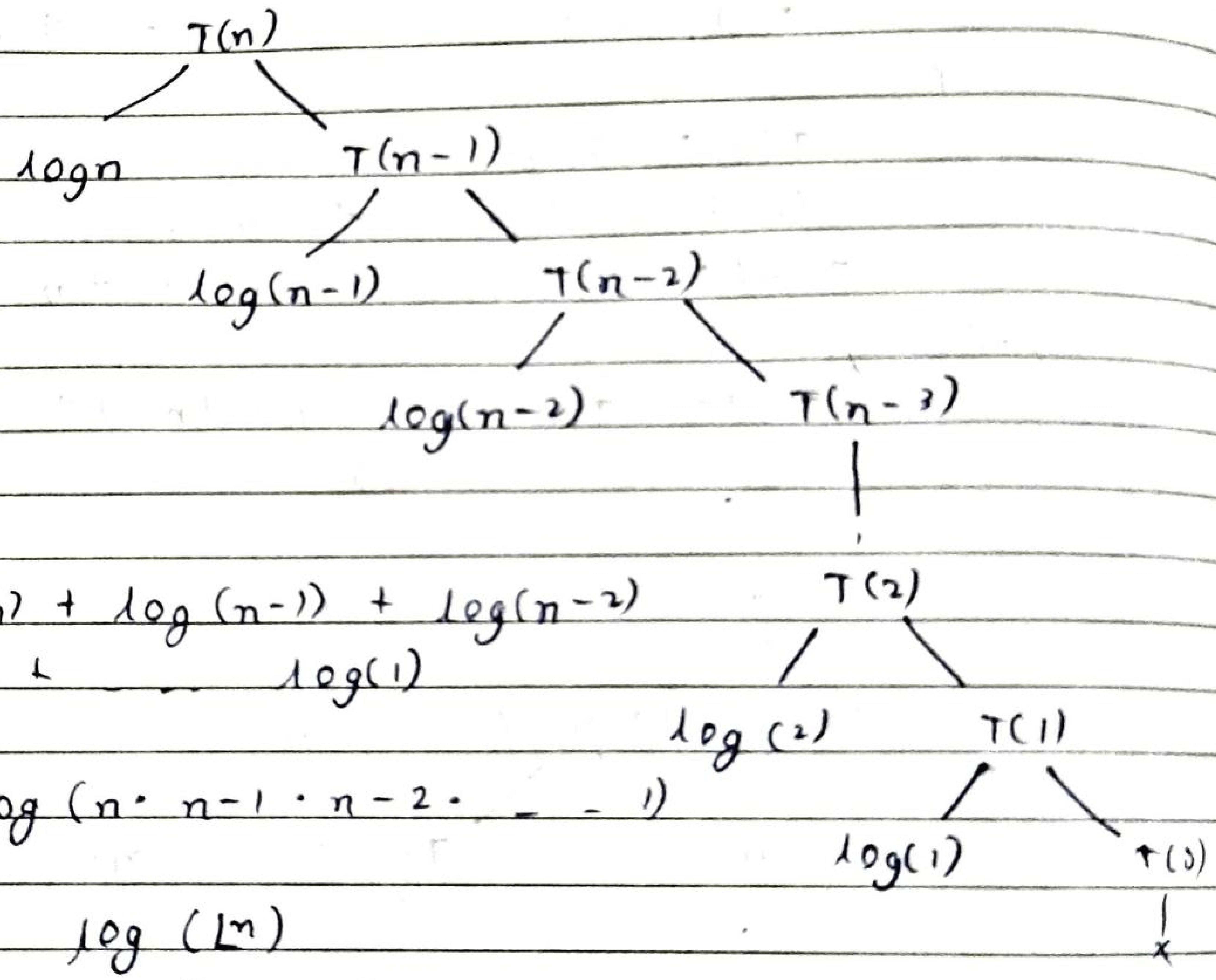
$T(n) = n + (n-1) + ---- 1 = O(n^2)$

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

Scanned by TapScanner

3)
```
void T(int n) {
    if (n > 0) {
        for (i=1; i<n; i=i*2) {
            printf("%d", i);
        }
        T(n-1);
}
```

$$T(n) = \begin{cases} 1 & , n = 0 \\ T(n-1) + \log n & , n > 0 \end{cases}$$



$\log(n) + \log(n-1) + \log(n-2)$
$+ \ldots \log(1)$

$= \log(n \cdot n-1 \cdot n-2 \cdot \ldots 1)$

$\log(\lfloor n)$

$\Downarrow$

~~O(log~~ $O(n \log n)$

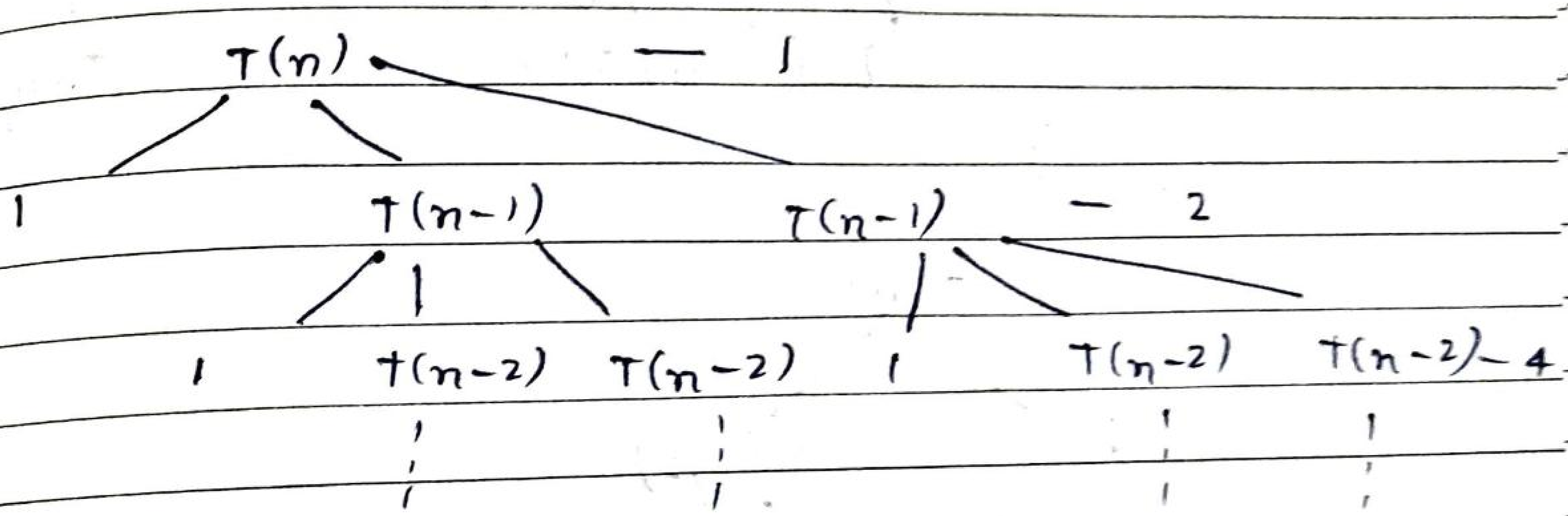
$$T(n) = T(n-k) + f(n)$$

$$T(n) = O(n * f(n))$$

1) Algorithm Test (int n) {

```
Algorithm Test (int n) {
        if (n > 0) {
            printf ("%d", n);
              Test (n-1);
              Test (n-1);
        }
}
```

$$T(n) = \begin{cases} 1 & , \\ 2T(n-1)+1 & , \quad n > 0 \end{cases}$$



$$1 + 2 + 2^2 + \ldots 2^k$$

$$2^{k+1} - 1 \qquad \frac{a(r^{k+1}-1)}{r-1}$$

$$\Downarrow$$

$$O(2^n)$$

Master's theorem

$T(n) = a\,T(n-b) + f(n)$

$a > 0, \quad b > 0 \qquad f(n) = O(n^k)$

if $a = 1$,

$T(n) = O(n * f(n))$

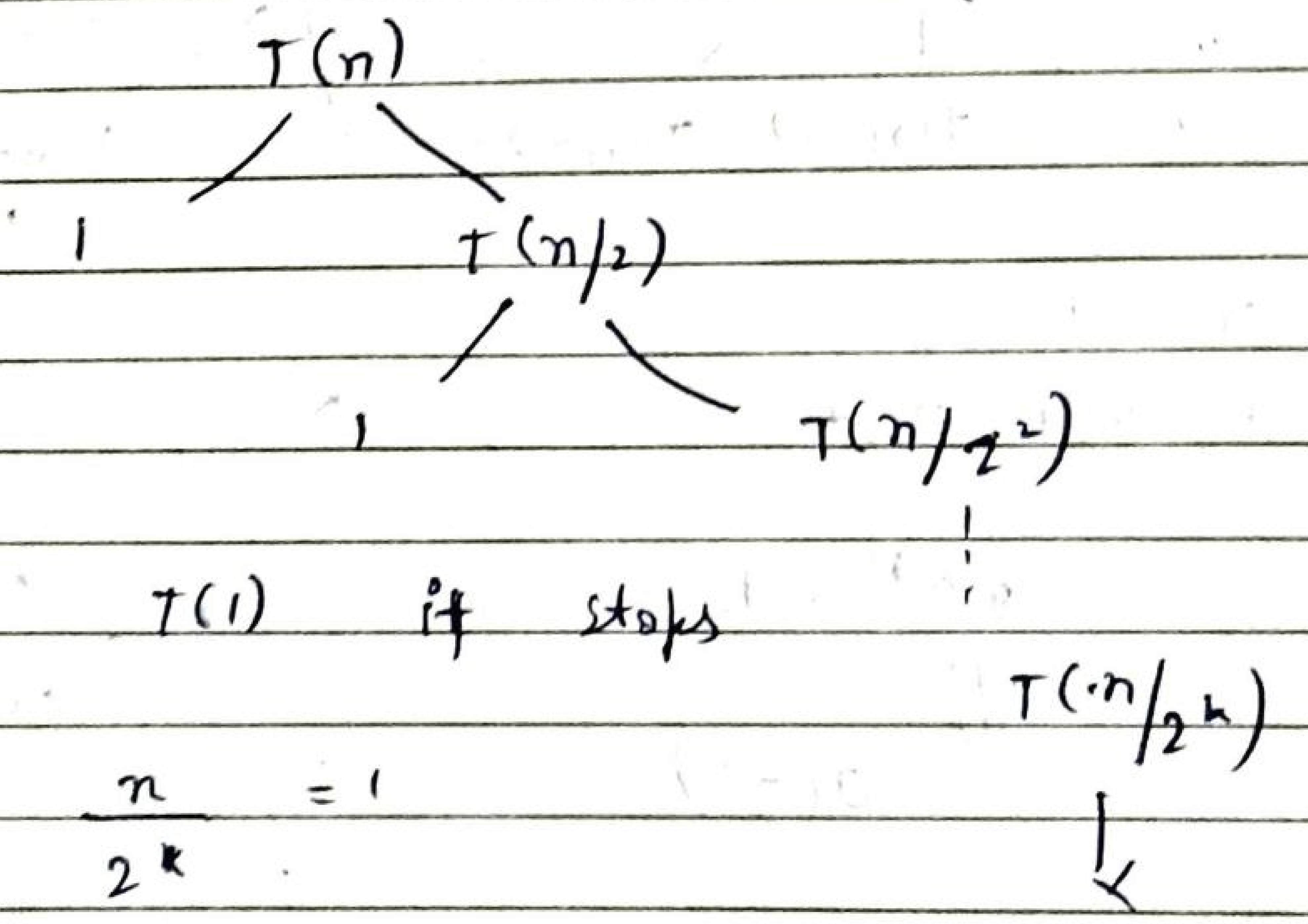
if $a > 0$

$T(n) = O(n^k * a^{n/b})$

$O(f(n) * a^{n/b})$

5) Algorithm Test (int n) { — $T(n)$

```
    if (n > 1) {
        printf("%d", n);      — 1
        Test (n/2);           — T(n/2)
    }
}
```



At $T(1)$ it stops

$\frac{n}{2^k} = 1$

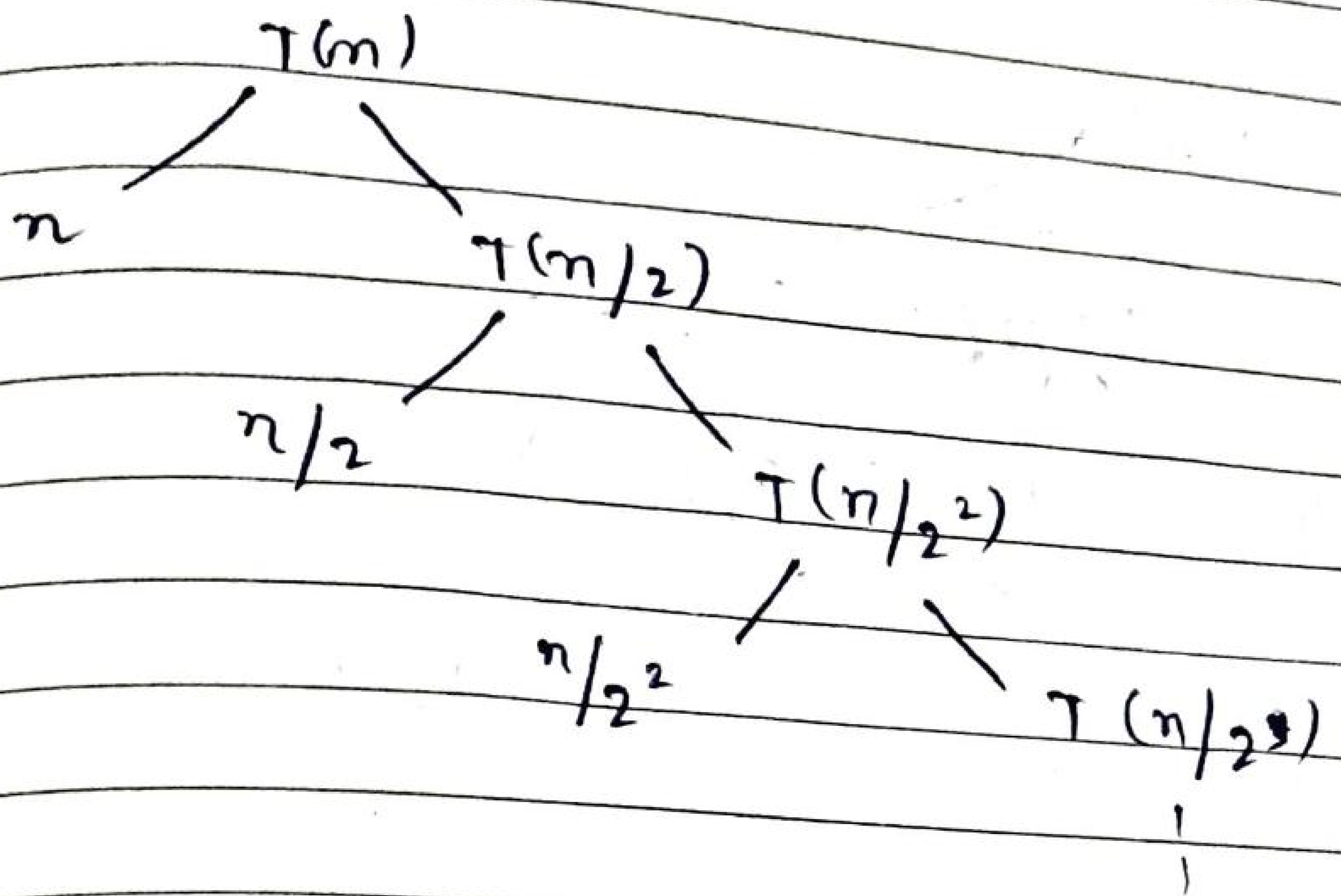
$k = \log_2 n$

$1 + 1 + 1 + \ldots$ k times

$= O(\log n)$

Page No.
Date

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + n & , n > 1 \end{cases}$$



$$\frac{n}{2^k} = 1, \qquad k = \log_2 n$$

$$T(n) = n + \frac{n}{2} + \frac{n}{2^2} + \cdots \frac{n}{2^k}$$

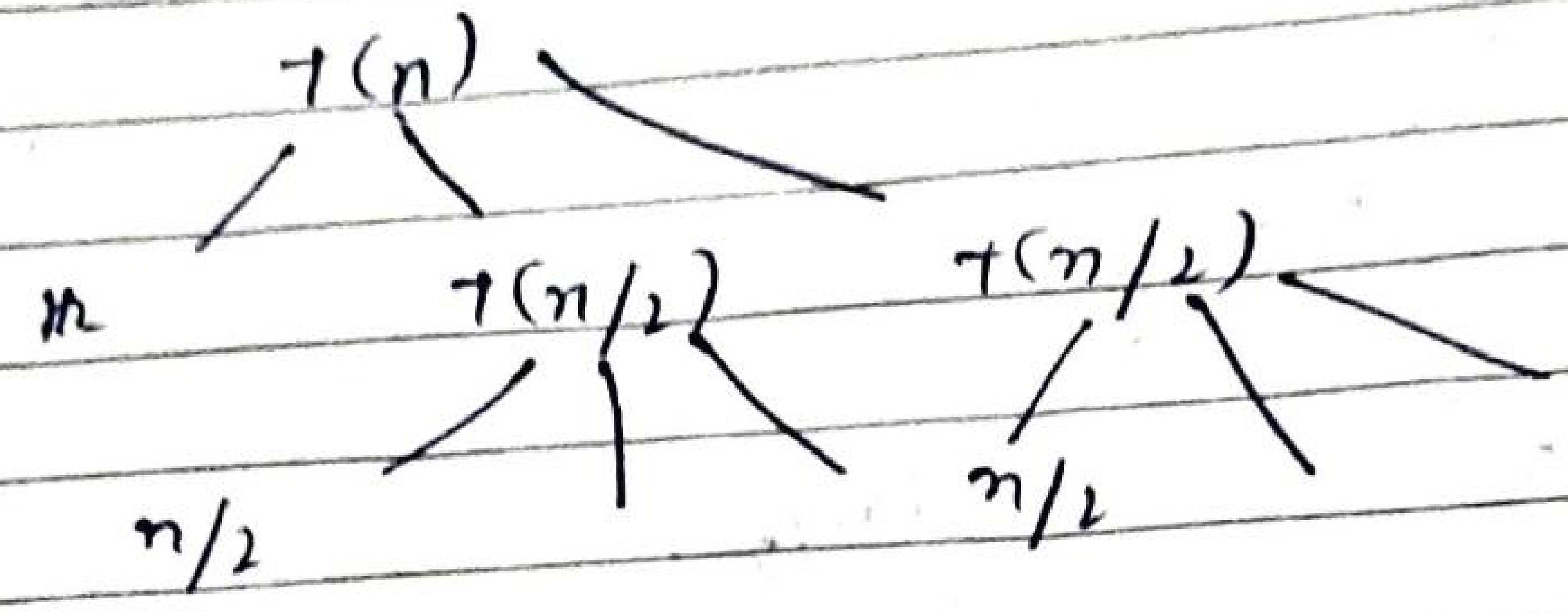
$$n\left[1 + \frac{1}{2} + \frac{1}{2^2} + \cdots \frac{1}{2^k}\right]$$

$$= n \sum_{i=0}^{k} \frac{1}{2^i}$$

$$= n \times 1$$

$$= n \quad \Rightarrow \quad O(n)$$

$$T(n) = \begin{cases} 1 & , \quad n = 1 \\ 2T(n/2) + n & , \quad n > 1 \end{cases}$$



$\frac{n}{2^k} = 1, \qquad k = \log_2 n$

$n + n + n + \ldots \; k$ times

nk

$= O(n \log n)$

Masters theorem

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

$a \geq 1, \quad b \geq 1, \qquad f(n) = O(n^k \log^p n)$

If $\log_b a > k$,

$T(n) = O(n^{\log_b a})$

If $\log_b a = k$,

$T(n)$ if $p > -1$

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

if $p = -1$

$O(n^k \log \log n)$

if $p < -1$,

$O(n^k)$

If $\log_b a < k$,

if $p \geq 0$, $O(n^k \log^p n)$

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