

Ques-1. What is the time complexity of below code and how.

```
void fun(int n)
```

```
{ int j=1, i=0;
```

```
  while (i < n)
```

```
  { i = i + j;
```

```
    j++; }
```

```
}
```

Solⁿ: $i = 0, 1, 3, 6, 10, 15, 21, \dots, n$

Let the sum of above k terms is S_k

$$S_k = 1 + 3 + 6 + 10 + 15 + 21, \dots + T_k \dots \textcircled{1}$$

$$S_{k-1} = 1 + 3 + 6 + 10 + 15 + 21 + \dots + T_{k-1} \dots \textcircled{2}$$

Subtracting $\textcircled{2}$ from $\textcircled{1}$

$$T_k = S_k - S_{k-1} = 1 + 2 + 3 + 4 + 5 + 6 + \dots + k$$

we have $T_k = n$

$$\therefore 1 + 2 + 3 + 4 + 5 + \dots + k = n$$

$$\frac{k(k+1)}{2} = n \Rightarrow k^2 + k - 2n = 0$$

$$\Rightarrow k = \frac{-1 \pm \sqrt{8n+1}}{2}$$

taking only positive value we get total no. of times
the loop runs for $i = k+1 = \sqrt{8n+1}$

$$\therefore \text{Time complexity, } T(n) = O\left(\frac{\sqrt{8n+1}^2}{2}\right) = O(\sqrt{n})$$

Ques.2 Write Recurrence Relation for the recursive function that prints fibonacci series. Solve the recurrence relation to get time complexity of the program. What will be time space complexity of this program and why?

Solⁿ:

Recursive function

```
int fib(int n)
```

```
{ if (n <= 1) → O(1) = c
```

```
    return n;
```

```
    return fib(n-1) + fib(n-2) → T(n-1) + T(n-2)
```

```
}
```

Recurrence Relation, $T(n) = T(n-1) + T(n-2) + c$

$$T(n-1) \approx T(n-2)$$

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

$$\therefore T(n-2) = 2 * (2T(n-2-2) + c) + c$$

$$= 4T(n-2) + 3c$$

$$T(n-4) = 2 * (4T(n-2) + 3c) + c$$

$$= 8T(n-3) + 7c$$

Generalising

$$= 2^k T(n-k) + (2^k - 1)c$$

Put $n-k = 0$

$n = k$

Put $n = k$

$$T(n) = 2^n * T(0) + (2^n - 1)c$$

$$= 2^n * 1 + 2^n c - c$$

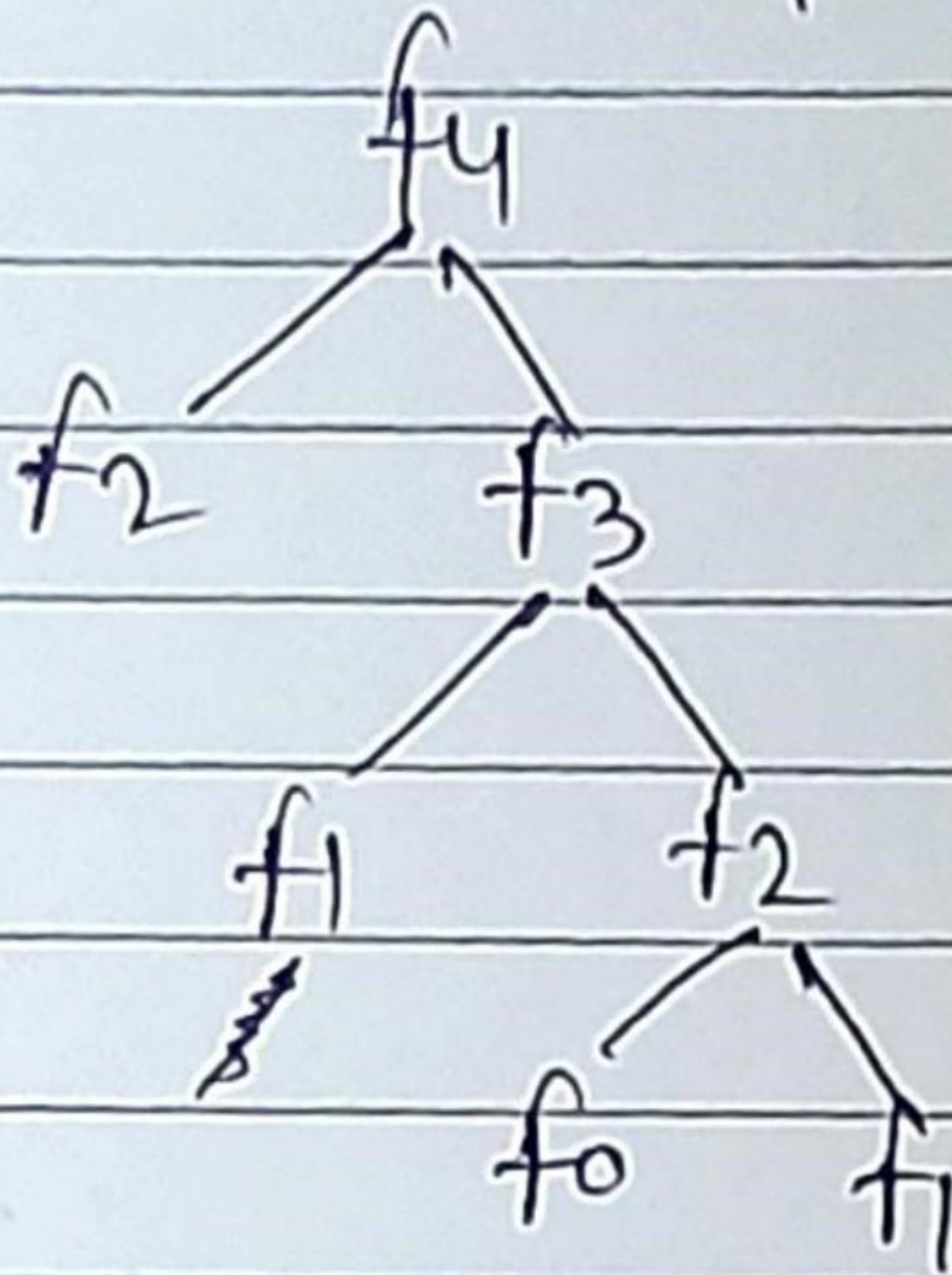
$$= 2^n (1 + c) - c$$

$$= 2^n$$

$$\text{Time Complexity} = O(2^n)$$

Date_____

Space Complexity: Space is proportional to the maximum depth of the recursion tree



Hence space complexity of Fibonacci recursion is $O(N)$

Ques-3: Write programs which have complexity.

Solⁿ:

1. $n(\log n)$.

```
for (i=1; i<=n; i++)
```

```
{ for (j=1; j<=n; j=j*2)
```

```
{ sum = sum + j;
}
```

```
}
```

2. n^3

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

```
{ for (j=0; j<n; j++)
```

```
{ for (k=0; k<n; k++)
```

```
{ sum = sum + k;
}
```

```
}
```

```
}
```

3. $\log n(\log n)$.

```
for (i=1; i<=n; i=i*2)
```

```
{ for (k=1; k<=n; k=k*2)
```

```
{ sum = sum + j;
}
```

```
}
```


Ques 4: Solve the recurrence relation $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + cn^2$

Solⁿ

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + cn^2$$

$$\therefore T\left(\frac{n}{4}\right) = T\left(\frac{n}{2}\right)$$

$$\Rightarrow T(n) = 2T\left(\frac{n}{2}\right) + cn^2$$

As $a \geq 1$ and $b > 1$

\therefore Using master's Method.

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$c = \log_b a$$

$$c = \log_2 2 = 1$$

$$f(n) > n^c$$

$$\therefore T(n) = O(f(n)) \\ = O(n^2)$$

Ques 5: What is the time complexity of the following function.

```
int fun(int n)
```

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

```
  for(int j=1; j<=n; j+=i)
```

```
    { some O(1) task
```

```
  } } }
```


Solⁿ

for $i = 1$, j is $1, 2, 3, 4, \dots$ run for n times
 for $i = 2$, j is $1, 3, 5, \dots$ upto $n/2$ times
 for $i = 3$, j is $1, 4, 7, \dots$ run for $n/3$ times

$$\begin{aligned}
 T(n) &= n + n/2 + n/3 + n/4 + \dots \\
 &= n \left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots \right) \\
 &= n \int_1^n \frac{dx}{x} \\
 &= [\log x]_1^n
 \end{aligned}$$

\Rightarrow Time complexity = $n \log n$.

Ques: What should be the time complexity of
 for ($i = 2$; $i \leq n$; $i = \text{pow}(i, k)$)

{ some $O(1)$ expression or statements }

where k is a constant.

Solⁿ

for first iteration $i = 2$

second iteration $i = 2^k$

third iteration $i = (2^k)^k = 2^{k^2}$

!

n^{th} iteration, $i = 2^k$ loop ends at $2^i = n$

apply $\log n = \log_2 2^{k^i}$

$k^i = \log n$

$i = \log_k (\log n)$

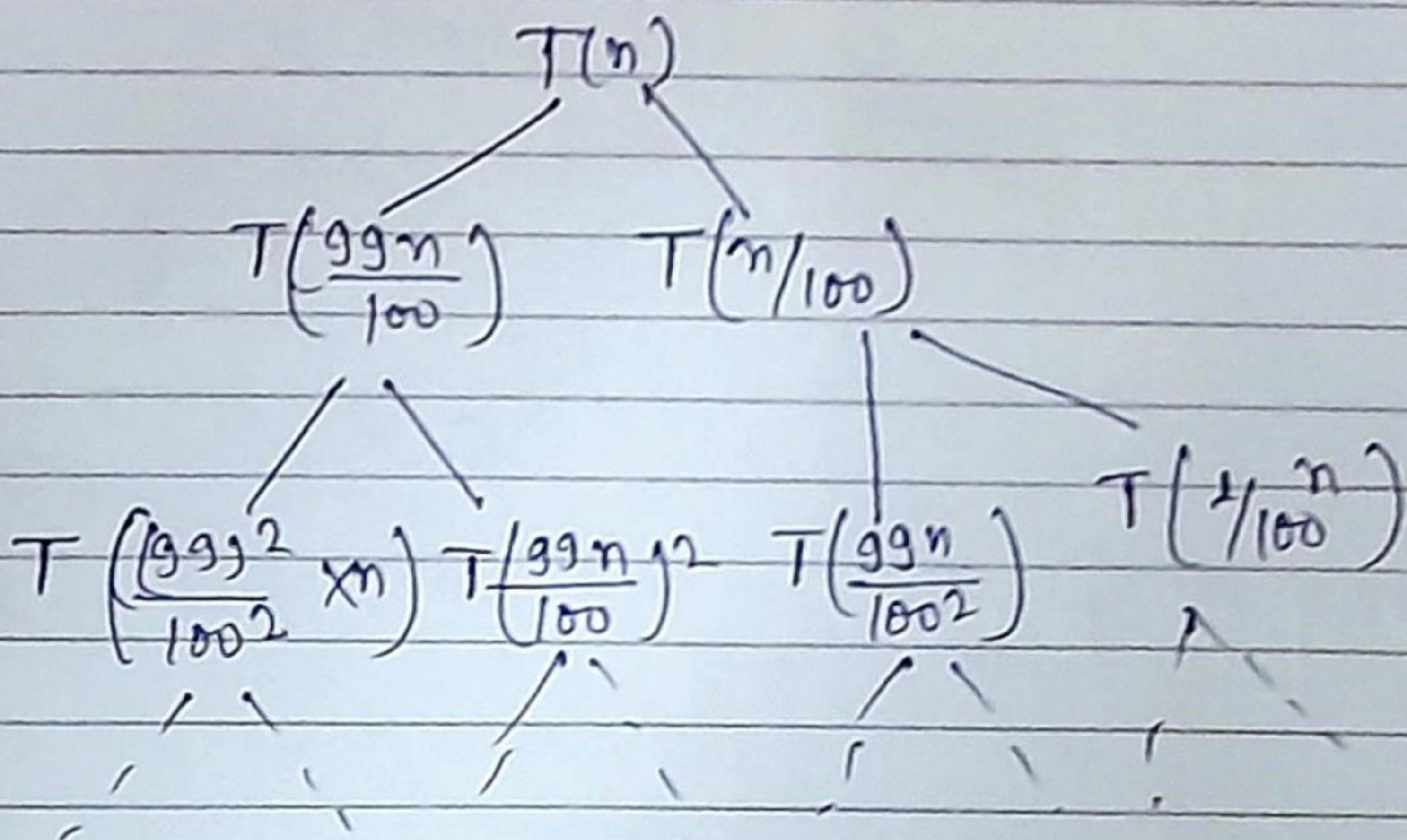
Ques-7 Write a recurrence relation when quick sort repeatedly divides the array into two parts of 99% and 1%.
Derive the time complexity in this case. Show the recursion tree while deriving time complexity and find the difference in heights of both the extreme parts. What do you understand by this analysis.

Solⁿ
99 to 1 in quick sort
when pivot is where from front or end always.

So,

$$T(n) = T(99n/100) + T(n/100) + O(n)$$

$$T(n) = T(99n/100) + T(n/100) + O(n)$$



$$\frac{n}{99}$$

$$n = \left(\frac{99}{100}\right)^k$$

$$\log n = k \log \frac{99}{100}$$

$$k = \log n \frac{100}{99}$$

$$\therefore T.C = n * \log_{\frac{100}{99}}(n)$$

Ques. Arrange the following in increasing order of rate of growth,

Solⁿ.

$$a. 100 < \log \log(n) < \log^2 n < \log n < \log n! < n < n \log n < n^2 < 2^n < 4^n < 2^{(2^n)} < n!$$

$$b. 1 < \log \log(n) < \sqrt{\log n} < \log(n) < 2 \log(n) < \log(2n) < n < 2n < 4n < \log n! < n \log(n) < 2^{(2^n)}$$