Recurrence Relation

Date.....................

Recurrence Relation

A recurrence is a relation, it is an equation or inequality that describe a function in terms of its value on smaller inputs.

code (Binary Search) :- (without recursion)

```
#include<iostream>
using namespace std;

int main()
{
    int arr[] = {15, 22, 27, 31, 36, 39, 56 };
    int n = 7;
    int search, mid, low = 0, high = n-1,
                              found = 0;

    cout << "\n current array :";
    for(int i = 0; i< n; i++)
    {
        cout << arr[i] << " ";
    }

    cout << "\n Enter element you want
                  to search :";
    cin >> search;

    for(int i = 0; i< n; i++)
    {
        mid = (low+high)/2;
```

Date.................

```
    if (arr[mid] == search)
    {
        found = 1;
        break;
    }
    else
    {
        if(search < arr[mid])
        {
            high = mid - 1;
        }
        else
        {
            low = mid +1;
        }
    }
  }
  if(found == 1)
  {
     cout << "\n element found at" <<
           mid +1 << "position" << endl;
  }
  else
  {
     cout << "\n Element not found";
  }

  return 0;
}
```

Date....................

Algorithm of Binary search using recursion

```
binarySearch(arr, low, high, search)
{
    mid = (low + high)/2;
    if(arr[mid] == search)
    {
        return mid;
    }
    else
    {
        if(arr[mid] > search)
        {
            binarySearch(arr, low, mid-1, search);
        }
        else
        {
            binarySearch(arr, mid+1, high, search);
        }
    }
}
```

Here, we take constant time to find the mid & checking the condition.

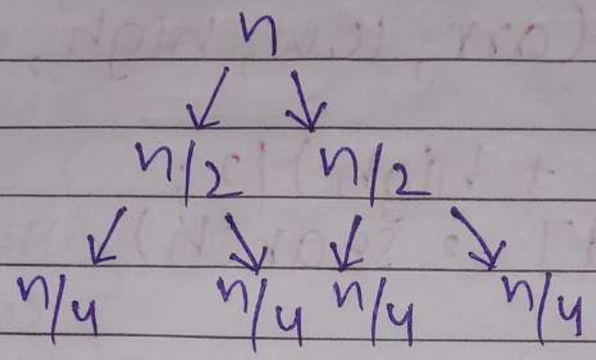
Here our problem are dividing into parts, until constant reach

Basically, in Binary Search, we have a problem to search an element & at every step we are dividing the problem into 2 parts. let suppose, we're having a problem, say 'n'. At every step, we are dividing this to n/2.

$$n \to \frac{n}{2}, \quad n \to \frac{n}{2}$$

Date.....................

At next step, we're dividing next parts :-



So here we have figure out a trend that our problem is dividing into half at every step, until we find our constant.
So, our recurrence relation for Binary search is :-

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

main problem — dividing into parts — constant time

How to solve recurrence relation :-
There are 4 methods to solve recurrence :-
i) Substitution Method.
ii) Iteration Method.
iii) Recursion Tree Method.
iv) Master Method.

Date....................

Substitution Method :-

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

Here, our function T is decreasing every time by half.
If our first step is,

$$T(n) = T(n/2) + C \quad \text{eq. (1)}$$

then our second step is,

$$T(n/2) = T(n/4) + C \quad \text{eq. (2)}$$

our third step is,

$$T(n/4) = T(n/8) + C \quad \text{eq. (3)}$$

and so on.

Now, we start back substitute method, substitute the value of eq. (2) in eq. (1),

eq. (1) substitute eq. (2) $\boxed{T(n/2) = T(n/4) + C}$

$$T(n) = T(n/2) + C$$
$$\underbrace{T(n) = T(n/4) + C + C,}_{\text{after substituting}}$$

eq. 2 in eq. 1

Date..................

Now, substitute eq. 3 in new eq,

$$T(n) = \underbrace{T(n/4)}_{\text{we can also write it as, } n/2^2} + c + c$$

so,

$$T(n) = T(n/2^2) + 2c \rightarrow \text{new equation}$$

substitute

eq. 3

$$T(n/4) = T(n/8) + c$$

$$T(n) = \underbrace{T(n/8)}_{\text{we can also write it as, } n/2^3} + c + 2c$$

so,

$$T(n) = T(n/2^3) + 3c \rightarrow \text{new equation.}$$

Here, we found a trend, i.e.,

$$T(n/2^2) + 2c$$
$$T(n/2^3) + 3c$$

} the no. of power we have, the no. of c we get.

it means, the next step is :-

$T(n/2^4) + 4c$, and the next step is,

$T(n/2^5) + 5c$, and so on.

Date..................

If we have 'k' steps, then our recurrence equation is,

$$\underbrace{T(n/2^k)}+kc.$$

now we have to make this T(1).

Assume, $2^k = n$,

$$T(\cancel{n}/\cancel{n}) + kc \rightarrow T(1) + kc.$$

$$\underline{\underline{1+kc}}.$$ final equation.

Calculating the power / value of k,

$2^k = n$. take log common here,

k will come forward

$$\log 2^k = \log n.$$

value of log 2 is 1,

$$k \log 2 = \log n.$$

$$\boxed{k = \log n.}$$ value of k is log n.

Put the value of k in final equation, i.e., $1 + kc$,

$$1 + kc \rightarrow 1 + \log n \cdot c$$ 1 & c both are constant, so ignore them.

The final value is, $\boxed{\log n}$

Date..................

Means, the time complexity of Binary search is,

$$O(\log_2 n).$$

Substitution Method

We make a guess for the solution and then we use mathematical induction to prove the guess is correct or incorrect.
In the substitution method, instead of trying to find an exact closed-form solution, we only try to find a closed-form bound on the recurrence. It is a very powerful approach which is able to prove upper bounds for almost all recurrences.

Date.....................

[Master Method]

Master method is a direct way to get the solution. The master method works only for the following type of recurrences or for recurrences that can be transformed into to following type :-

$T(n) = aT(n/b) + f(n)$, where $a \geq 1$ and $b > 1$.

if the problem is in this format then only we can use Master Theorem.

After this, our solution is $T(n) = n^{\log_b a} \cdot U(n)$

* What is $U(n)$?

- $U(n)$ depends on $h(n)$.
- $h(n) = \dfrac{f(n)}{n^{\log_b a}}$

- Relation between $h(n)$ & $U(n)$ is :-

$h(n)$	$U(n)$
$n^r, r > 0$	$O(n^r)$
$n^r, r < 0$	$O(1)$
$(\log_2 n)^i, i \geq 0$	$\dfrac{(\log_2 n)^{i+1}}{i+1}$

Spiral

Date..................

Ques| $T(n) = \underbrace{8T(n/2) + n^2}$

$aT(n/b) + f(n)$, $a = 8$
$b = 2$
$f(n) = n^2$

Solution, $n^{\log_b a} . U(n)$
$n^{\log_2 8} . U(n)$
$n^3 . U(n)$

this is converted to n^3 because 2^3 is 8 i.e., $\log_2 8$

value of $U(n)$ is calculated with $h(n)$, and $h(n) = \frac{f(n)}{n^{\log_b a}}$ we can find it in previous table.

$h(n) = \frac{f(n)}{n^{\log_b a}}$ → $f(n)$ is n^2

→ this is already calculated, i.e., n^3.

$h(n) = \frac{n^2}{n^3}$ → can be written as $\frac{1}{n}$

also written as n^{-1}

Date.....................

Now, $h(n) = n^{-1}$, we check it with the table,

if $h(n) = n^{r}$, $r < 0$ then $U(n)$ is $O(1)$.
Here, $r = -1$, i.e., < 0, so $U(n) = O(1)$.

Back to our solution,

$\rightarrow n^3 . U(n)$
$\rightarrow n^3 . O(1) \longrightarrow \boxed{O(n^3)}$

Ques2. $T(n) = T(n/2) + C$.

$a = 1$
$b = 2$
$f(n) = C$ constant.

Solution, $n^{\log_b a} . U(n)$
$n^{\log_2 1} . U(n)$
$n^0 . U(n)$.

value of $\log_2 1$ is 0

$h(n) = \frac{f(n)}{n^{\log_b a}}$

f(n) is c.

already calculated, i.e., 1

constant

So, $h(n) = C$ in our table, we have 3rd case for this situation,

Date..................

if $h(n) = (\log_2 n)^i,\ i \geq 0$ then

$$U(n) = \frac{(\log_2 n)^{i+1}}{i+1}$$

Here in our case,

$$h(n) = (\log_2 n)^0 \cdot c \Rightarrow 1 \cdot c \Rightarrow c$$

$$U(n) = \frac{(\log_2 n)^{0+1}}{0+1} \Rightarrow (\log_2 n) \cdot c$$

ignore the constant.

$$U(n) = \log_2 n.$$

<u>Back to our solution,</u>

$\rightarrow n^0 \cdot U(n)$

$\rightarrow 1 \cdot \log_2 n \rightarrow \boxed{O(\log_2 n)}$

Date..................

Recursive Tree Method

It is a way of solving recurrence relatio-ns. In this method, a recurrence relation is converted into recursive trees. Each node represents the cost incurred at various level of recursion. To find the total cost, costs of all levels are summ-ed up.

Steps to solve recurrence relation using Recursive Tree Method :-

i) Draw a recursive tree for given recurre-nce relation.

ii) calculate the cost at each level & count the total no. of levels in the recursion tree.

iii) count the total no. of nodes in the last level & calculate the cost of the last level.

iv) sum up the cost of all the levels in the recursive tree.

Date.....................

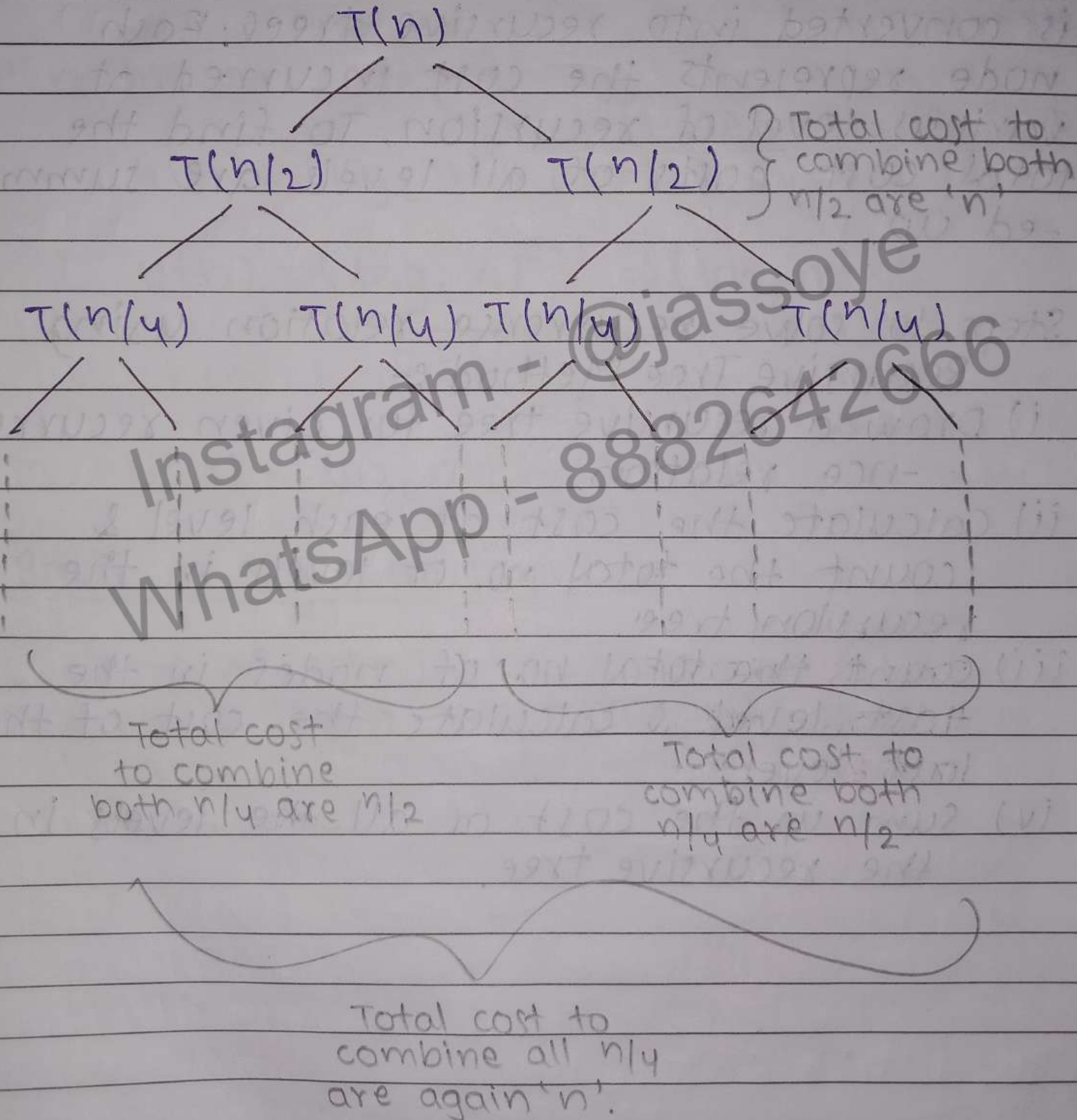
n is converted in n/2

→ constant.

Ques] $T(n) = 2T(n/2) + cn$

→ n ko n/2 me divide krne me 'cn' cost lg rhi hai.

Step1: Draw a recursive tree:-



Date.....................

Total cost

$T(n) \rightarrow cn$

$\downarrow$

$2\, n/2 \rightarrow cn$

$\downarrow$

$4\, n/4 \rightarrow cn$

$\downarrow$

$8\, n/8 \rightarrow cn$

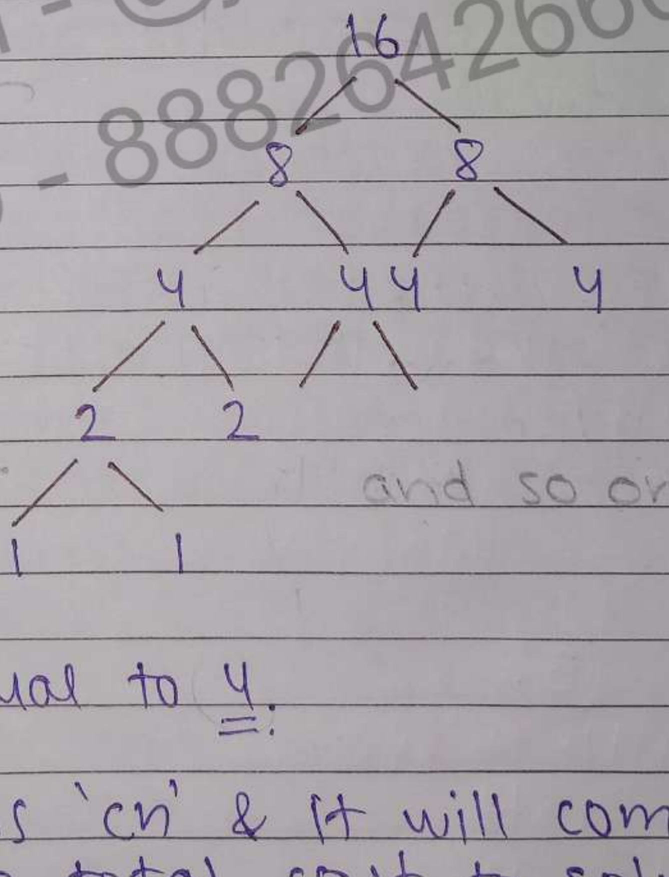
$\downarrow$

$16\, n/16 \rightarrow cn$

* Total cost is dependent on the height of the tree.

For example, suppose our main problem is 16, then at every step it will divided in half.

Means,



Total 4 step a taken to solve the problem of 16.

and so on....

If we take, $(\log_2 16)$, its eaual to $\underline{\underline{4}}$.

Means, our cost is 'cn' & it will come 'log n' times. So, total cost to solve this recurrence relation is :-

$$\boxed{O(n\log n)}$$

Spiral