Binary Search Recursion:

calling the same function again inside the method definition.

Ex: (10, 20, 30, 40, 50, 60). $x = 50$
0 1 2 3 4 5

binarySearch (arr, 0, 5):

mid = (i+j)/2 = 2 (lower bound)

while i <= j:

if arr(mid) == x; ← 90 ≠ 50 ⊗

return mid;

elif arr(mid) < x:

90 < 50: ✓

binarySearch (arr, 3, 5): right move

mid = 4

arr(mid == x) = ✓

arr(mid) > x:

← else

binarySearch (arr, 0, 1)

└ left move.

* for mid, we can use

mid = i + (j-i)/2   mid = i+j/2

✓

most preferrable.

code Binary search:

```
def binarySearch(arr, x, left, right):
    if left > right:
        return -1          → C
    mid = left + (right-left)/2
    if arr[mid] == x:      → C
        return mid
    elif arr[mid] < x:
        return binarySearch(arr, x, mid+1, right)     T(n/2)
    else
        return binarySearch(arr, x, left, mid-1):     T(n/2)

arr = [10, 20, 30, 40, 50]
left = 0
```

0 + (1-0)/2
1/2 = 0.
left = 1
mid 1
1 + (-1-1)/2
-2/2
-1
1 - 1 = 0

```
right = len(arr)-1
x = int(input("enter no.")), result = binarysearch(arr, left,
if result != -1:
    print(result)
else:
    print("not found")
```

Reccurence Relation:

$T(n) = T(\frac{n}{2}) + c$ ✓ ← Substitution, master's Theorem.

(or)

$T(n) = 2T(\frac{n}{2}) + c$ [wrong]

① master's Theorem:

$a = 1 \quad k = 0$

$b = 2 \quad p = 0$

$\log_b^a = \log_2^1 = 0.$

$\log_b^a = k$. [case-2]

(i) $p > -1 \Rightarrow \Theta(n^k \log^{p+1} n)$

$\Theta(\log n)$ ✓

② By substitution,

$T(n) = T(\frac{n}{2}) + c$

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

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

$= T(\frac{n}{2^3}) + 3c$

{ k times

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

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

$n = 2^k$

$= T(\frac{n}{2^{\log_2 n}}) + c.k$ $\boxed{\log_2^n = k}$

$= O(\log_2^n)$ ✓

Interview Questions on Binary Search:

① Facebook (Technical Round):

↳ Not Sorted manner.

$n \rightarrow$ size of an array.

(-23	40	55	1	2	27	-89	∞	∞	∞	∞	∞)
0	1	2	3	4	5	6	7	8	9	10	11

Index of first infinite.

output = 7.

① since, this is not sorted, ⇒ Binary Search is not applicable.

Going for linear Search.

```
for i=0 to n-1
    if arr[i] == 'inf'
        return i
```
} O(n)

② modified Binary Search.

mid = 0 + (11-0)/2 = 0+5 = 5

if arr(mid) == int;. → move to right side Search Space

left = mid+1;

arr(mid) <= 'inf'

right = mid-1

Note: 1. Don't judge by looking whether the array is physically sorted or not.

② -8 [20 27 47 29 35 ∞ ∞ ∞ ∞ ... ∞

0 1 2 3 4 - - - - - n

1 2 3 4 5 6 7 8 9 ..... n

n → not defined

right

∞ no's are only available after all integers.

Sol: Do exponential increase → range of numbers.

↑

Binary Search.

[47 29 35 ∞ ∞]

4 5 6 7 ↑ 8

mid = 6.

move right

mid $\frac{7+8}{2} = \frac{15}{2} = 7$

check

if arr[i-1] == int

return arr[i]

(5). Array is sorted.

20, 40, 60, 80, 90, 120, 240. 130.
0 1 2 3 4 5 6

a+b = 210 → (a,b).

Approach-1:
```
for i=0 to n-1 :
    for j=0 to n :
        arr[i]+arr[j] == target
            return i,j
```
} O(n²) Brute-force

Approach-2: one loop for fixing key value. → n

Say key=20.

210-20=190

and search for this 190. } O(n²)

using Binear search → n

Approach-3:

one loop for fixing key value. } → O(nlogn)

Search the remaining Target value using Binary Search

Approach-4: optimized approach : Greedy (Two-pointers).

Two-Sum.

20, 40, 60, 80, 90, 120, 240

↑ (l) ↑ (r)

```
while (l<=r):
    if arr(l)+arr(r) == target:
        return (l,r)
    elif arr(l)+arr(r) > target:
        r=r-1
    else:  l=l+1
        arr(l)+arr(r) < sum-value)
```

In worst case. We might traverse Whole array i.e- O(n). → optimized