16) a) We are given the following information:
$x = 1$ and the updates allowed are $x = x+1$, $x = x+5$ and $x = 3x-1$

In case of $x = x+1$, we know that addition of positive number to positive number results in positive number, so if add the positive number to 1 (Given positive number), the result will be positive number. (+ add + = +
e.g $1+1=2$
$2+1=3$) → (1)

If we consider the case $x = x+5$, the result will be positive since if we apply the general theory of addition of two positive numbers is positive ∴ if (e.g:- $1+5=6$) → (2)

In case of $x = 3x-1$, if we try to solve using initial given value, we observe that $3x-1$
i.e, $3(1)-1 = 3-1 = 2$
The above expression results in positive value.
Similarly, $3x-1$ increasing value of 1 results i.e, taking $x = 2$
$\therefore 3(2)-1 = 6-1 = 5 > 0$
Similarly, $3(3)-1 = 9-1 = 8 > 0$
$3(4)-1 = 12-1 = 11 > 0$
$3(5)-1 = 15-1 = 14 > 0$
$3(6)-1 = 18-1 = 17 > 0$
$3(10)-1 = 30-1 = 29 > 0$
} result is positive → (3)

We can see that if we substitute x as positive value then the result in positive.

We can also observe that the result is greater than the given value of x i.e, 1.

∴ from expression (1), (2), (3)

we can see that: A) all the updates result in positive value,
B) All the updates are greater than the initial x value ie 1

∴ from A, B, we can say that x will always increase after each step.

16)b) for y in range (1, n)

$T(a+1) = T(a+1)+1$
$T(a+5) = T(a+5)+1$
$T(3a-1) = T(3a-1)+1$
return T(n)

Algorithm Justification :→

i) We will loop till range 1 to n
ii) It will take

15) a) Run recursively by recursing on the left and right halves of A

~~Maximum Contiguous Subarray~~

```
Run( A, x, y)
    if (x == y)
        return 0;
    addLeft → Run(A, x, y/2)
    addRight → Run(A, x/2 +1, y)
    if ( addLeft == 0 && addRight == 0)
        if (A[y/2] < A[y/2 +1])
            return 1
        else
            return 0
    else if (addLeft > 0 && addRight > 0)
        if ( A[y/2] < A[y/2 +1])
            return addLeft
        else
            return addLeft + addRight.
    else
        return addLeft + addRight
```

b) * We can see from the above algorithm that execution time of each call is $\frac{n}{2}$

* The recursion relation would be

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

By applying the master's theorem,

we get $a = 2$

$b = 2$

$d = 1$

$\log_b a = \log_2 2 = 1$

As, $d = 1$

$\therefore d = \log_b a$

$\therefore$ Running time =

$O(n \log n)$

16) b)

```
n = 1
T[2 ... n] = 0
Runs(n, T)
    for i in range(1, n-1)
        T[i+1] = T[i+1] + 1
        if (i+5 ≤ n)
            T[i+5] = T[i+5] + 1
        if (3i-1 ≤ n)
            T[3i-1] = T[3i-1] + 1
    return T[n]
```

∵ As we can see there is update $n = n+1$, also, as we are initiating from 1 we get result anywhere between 2 to n

∴ Therefore, all updates such as $n = n+5$ & $n = 3n-1$ to all values that lies between the range 2 to n. Also, array T stores the number of value i where i is in the range 2 to n.

∴ In end, T[n] is returned that contains the no. of ways we achieve n

Time Complexity

L1 → 1 time
L2 → n-1 times (assignment
L4 to 9 executes ≤ 6(n-1) times
L10 runs 1 time

$$\therefore T(n) \leq 2 + (n-1) + 6(n-1)$$

$$\leq 2 + 7(n-1)$$

$$\therefore T(n) = O(n)$$