SoC 2022 : Competitive Coding

Week-4

Divide-Conquer Paradigm

Contents

Solving Recurrences

- Useful for computing time complexities of recursive functions.
- Most of the recurrences that we see here are of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + O\left(n^d\right) = aT\left(\frac{n}{b}\right) + cn^d$$

We see asymptotic solution (i.e., as n→∞), so base case doesn't really matter (should be a constant).

Master Theorem: [can prove by recurrence tree method]

Note: To use this theorem, we must have that a, b, d are constants (i.e., they shouldn't be functions of n), and $a \geq 1, b \geq 1, d \geq 0$.

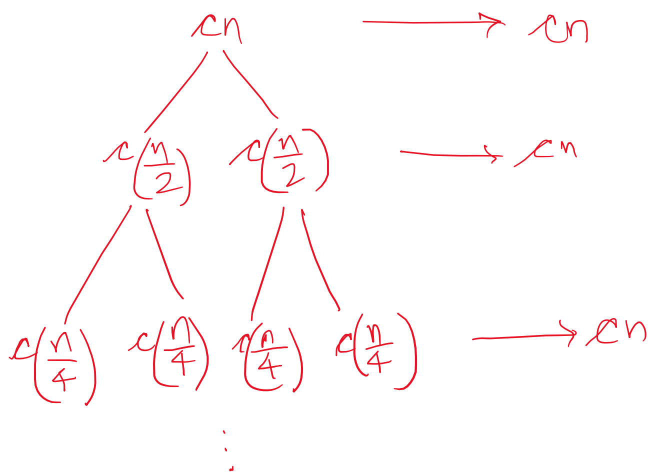
If $a < b^d, T(n) = O(n^d)$

If $a = b^d, T(n) = O(n^d \log n)$

If $a > b^d, T(n) = O\left(n^{\log_b a}\right)$

- Recurrence Tree method:

$T(n) = 2T\left(\frac{n}{2}\right) + \underbrace{O(n)}_{\text{e.g. } cn}$



$\#\text{levels} = \log_2 n \sim \log n$

$\Rightarrow T(n) = O(cn \cdot \log n) = O(n \log n)$

Master : $a = 2, b = 2, d = 1$: $a = b^d$

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{2n}{5}\right) + 7n$$

$$7n \longrightarrow 7n$$

$$\frac{7n}{2} \qquad \frac{2}{5}\times 7n \longrightarrow 7n\left(\frac{1}{2}+\frac{2}{5}\right)$$

$$\frac{1}{2}\left(\frac{7n}{2}\right) \qquad \frac{2}{5}\times\frac{1}{2}\times 7n \qquad \frac{1}{2}\times\frac{2}{5}\times 7n \qquad \left(\frac{2}{5}\right)^2 7n \rightarrow 7n\left(\frac{1}{2}+\frac{2}{5}\right)^2$$

$$\vdots$$

Can see GP. Tree will have ~ finite #levels,

so $T(n) \leq \underbrace{\infty\text{-summation}}_{\frac{7n}{1-\left(\frac{1}{2}+\frac{2}{5}\right)}} = 70n$

$\Rightarrow \quad T(n) \leq 70n$

Now, can try (put) $T(n) = kn$ in recurrence to get $k=70$.

Someone asked about the recurrence T(n) = T(n/2)+T(2n/3)+7n. [The infinite GP won't converge.]

To get a quick loose bound, we can re-write this as T(n) = 2*T(2n/3)+7n [since T(n) is an increasing function here, so T(n/2)≤T(2n/3)]. Using the Master Theorem [a=2, b=3/2 (≥1), d=1], we get T(n)=O($n^{\log_{\frac{3}{2}} 2}$)~O($n^{1.71}$). Of course, this is not a tight bound.

A tighter bound is shown in **recurrence.png**. It is just based on considering a finite number of levels.

[Wolfram Alpha gives the tight bound of Θ($n^{1.2932\cdots}$). This is the Big-Theta notation. Can think of O to be an upper bound, Ω to be a lower bound, Θ to be a tight bound.]

To prove the Master Theorem, just build a recurrence tree (root node value = n^d, it has a children, each having a node value of $\left(\frac{n}{b}\right)^d$, and so on). Add all the levels up, using a finite GP in the case $a > or = b^d$.

Divide & Conquer

- Basic idea is to divide the problem into subproblems, solve them (just make recursive calls), and then try to combine these solutions to get the final result.

- Have seen mergesort: $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n) \sim 2T\left(\frac{n}{2}\right) + O(n)$.
  Can think of the "~" as follows:
  Given an array of size $n \neq 2^k$ for any k, can put some number of -∞'s (say, -INT_MAX) at the end to get an array of size $n \leq n_1 \leq 2n$.
  Since $T(n)$ is increasing, $T(n) \leq T(n_1)$, and $T(n_1) = O(n_1 \log n_1)[Master\ Th.] = O(2n \log 2n) = O(n \log n)$. So, $T(n) = O(n \log n)$.

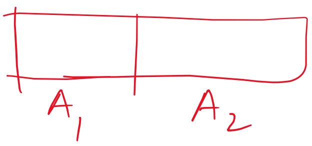
- Similarly, quick sort:
  Divide the problem: partition using pivot (most of the work done here)
  Solve them: Recursive calls
  Merge: (no need, already sorted in place)

1. Counting Inversions

i<j and a[i]>a[j]; (i,j) forms an inversion pair.
One way is to do it in O(n^2) [nested for loops]

Use Divide and Conquer [basically, it is modification of Mergesort idea]
https://www.geeksforgeeks.org/counting-inversions/

$A :=$



Observe: $\#$ inversions in $A =$

$$\# \text{ inv in } A_1 + \# \text{ inv in } A_2 + \underbrace{\sum_{e \in A_2} (\#\text{elems of } A_1 > e)}$$

Note: for unsorted A_1 and A_2, time cx-ity is

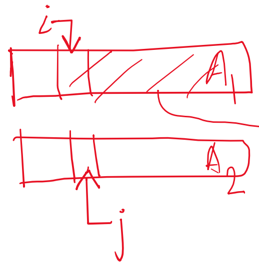
$$T(n) = 2\,T(n/2) + O(n^2) \leftarrow 3^{rd} \text{ part} \uparrow$$

$$= O(n^2)$$

$\Rightarrow$ sorted A_1, A_2 may help.

$N_A = N_{A_1} + N_{A_2}$

Now



} sorted

[merging ~ Mergesort]

all these will also be $> A_2[j]$

if $A_1[i] > A_2[j]$

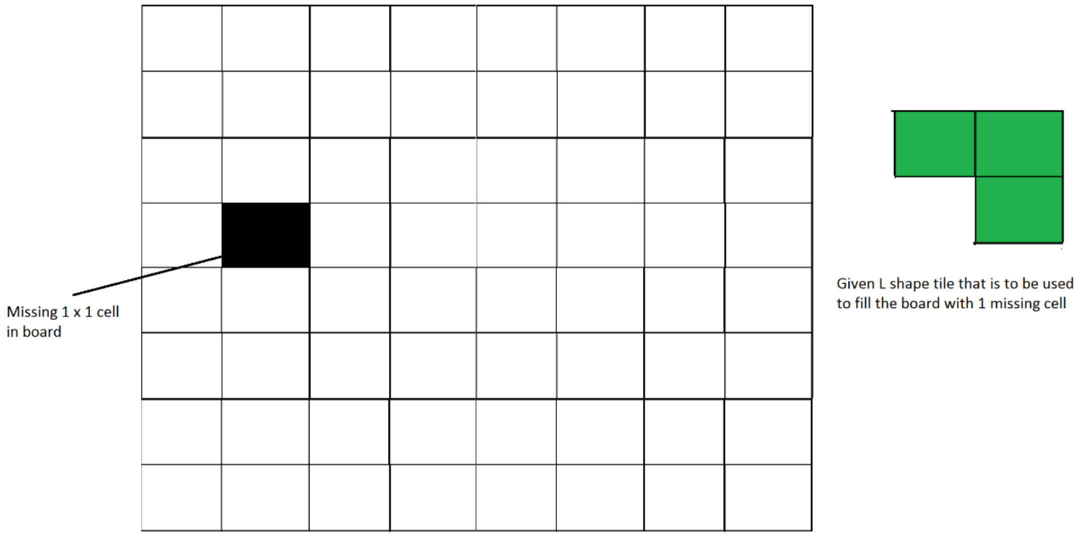
$N_A \mathrel{+}= \text{len}(A_1) - i$

$$\underbrace{\left.\begin{array}{l} i{+}{+} \\ j{+}{+} \end{array}\right\} \text{ till end of any one is reached.}}_{O(n)}$$

$\Rightarrow \quad T(n) = 2T\left(\frac{n}{2}\right) + O(n) \longrightarrow \underline{O(n \log n)}$

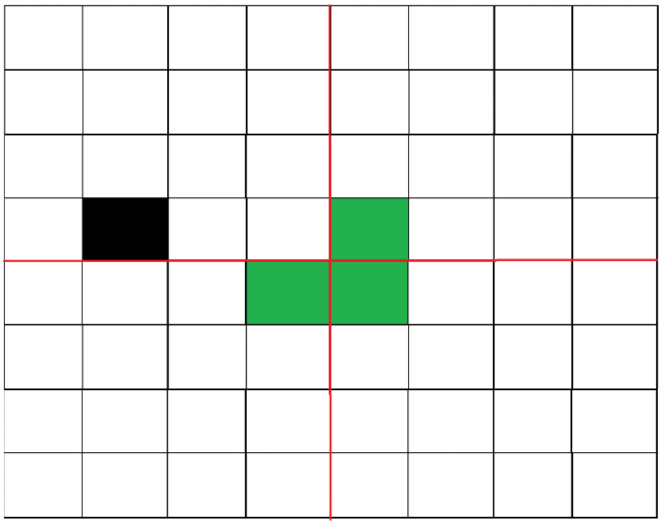
Code: count_inv.cpp

2. Tiling Problem



https://www.geeksforgeeks.org/tiling-problem-using-divide-and-conquer-algorithm/

Given a $2^k \times 2^k$ board, fill the board with these L-shaped trominos.
Proof of "can be filled": by induction (just like recursive calls).
Base case: k=1. 2×2 board. Can be filled.
Induction Hypothesis: for all i≤k, statement is true.
Induction Step: Showing for k+1.



Can see 4 smaller instances (of size $2^k \times 2^k$). Done.

To get the trominos' locations, just make recursive calls.

3. Calculating x^n

Repeated multiplication of 'x': O(n).
Better solution:
See fast-exp.cpp.

For the iterative solution:
Think in terms of binary representation of n (i.e., exponent).
Consider x^13, where 13=1101_2, so, x^13 = x * x^4 * x^8.
For every position in the binary representation (read it from the right), current power is squared (e.g., x -> x^2 -> x^4 ...), and whenever the bit is set (i.e., =1), then the current power is multiplied.
Time complexity: O(log n).

Code.

4. Polynomial Multiplication

Given 2 polynomials P (degree n) and Q (degree m) [coefficients given], give the coefficients of P.Q.

One way is:
PQ[k]=0 for all k in $\{0, \ldots, m+n\}$.
For i=0 to n
    For j=0 to m
        PQ[i+j] += (P[i]*Q[j])   // coeff of x^{i+j} in PQ incremented

Time complexity: O(mn) = O($(\max(m,n))^2$).

Using divide and conquer:
Pad these polynomials with 0 coefficient powers of x to make both to be "degree-k", where k is a power of 2.
Note that $k \leq 2 * \max\{n, m\}$, so the size won't increase "that much". This is because $n = 2^{\log n} \leq 2^{\lceil \log n \rceil}$. Similar for m.

Now, let the polynomials be P' and Q' [both have "degree" k (consider 0 coefficients too, in this definition of "degree")].
P' = A$x^{\frac{k}{2}}$+B and Q' = C$x^{\frac{k}{2}}$+D, where A, B, C, D are (at most) degree k/2 polynomials.
P'Q' = ACx^k+(AD+BC)$x^{\frac{k}{2}}$+BD.

If we just recursively compute AC, AD, BC, BD, then the time complexity is T(k) = 4*T(k/2) + O(k) [assuming constant time to add 2 numbers, so O(k) time to add these polynomials].

Plugging a=4, b=2, d=1 [a>b^d] in Master Theorem, we get O($k^{\log_2 4}$)=O(k^2). No improvement so far.

The trick is to write AD+BC = (A+B)(C+D) – AC – BD. Since we are calculating AC and BD in other calls, we use them in O(1) here.
So, we have T(k) = **3***T(k/2) + O(k). This gives T(k)=O($n^{1.59\cdots}$).

This is based on Karatsuba's algorithm for fast (same time complexity as above) integer multiplication.

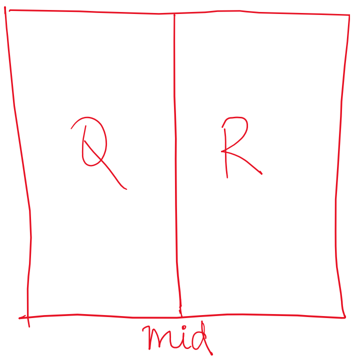
5. Closest pair of points

Given n points in a 2D plane, find the closest pair of points.

O(n^2) method is to check every pair.

Using Divide and Conquer:
Let the given set of points be P.
Sort these points according to x-coordinate [set Px], and according to y-coordinate [set Py].
The function call is: ClosestPair(Px,Py).

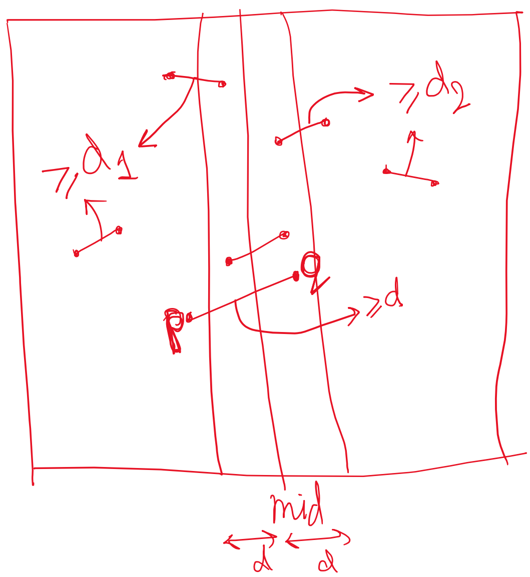
Define Q ≔ set of points in the left half of the plane, and R ≔ set of points in right half. [Here, we are dividing them by their x-coordinate.]



From **sorted** Px and Py, it is just O(n) to get Qx, Qy, Rx, Ry [just do linear scans in Px or Py]. Qx is the set of points in Q, sorted by their x-coordinates, etc.

By recursive calls, we will get (p1,q1) = ClosestPair(Qx,Qy) and (p2,q2) = ClosestPair(Rx,Ry). Now, only those pairs of points (p,q) remain, such that p∈Q and q∈R. [Note that p1, q1, p2, q2, p and q are all of the form (x, y).]

Let the distance p1-q1 be d1, and that of p2-q2 be d2. Also, d ≔ min(d1,d2).
Then, we only need to look in the region [mid-d,mid+d] (along the x-axis).



We don't need to consider points of the form (p,q) shown above, as distance p-q ≥d, so it will be ≥d1 or ≥d2, and thus won't get us any further in our solution [as we've already found pairs with d1 and d2].

Define Sy = set of points in P that have an x-coordinate ∈ [mid-d,mid+d], sorted by their y-coordinates [hence the **y** in Sy]. This can be built in O(n) time, using Py.

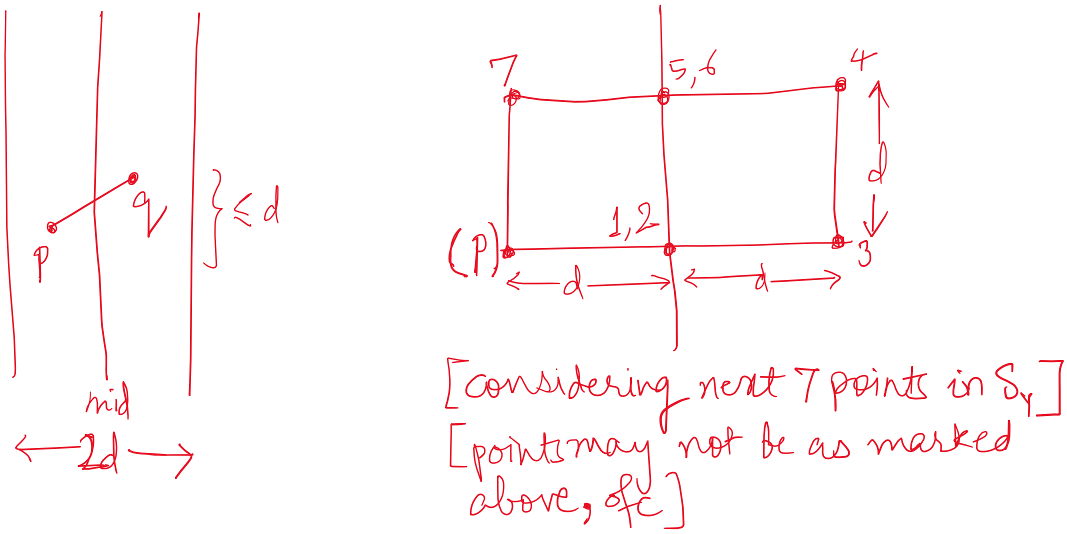
Now, algo is:

ClosestSplitPair(Px,Py,d)

    current_best = d
    Best_pair = (p1,q1) or (p2,q2), whichever gave the 'd'.
    For i=1 to |Sy|-7:
        For j=1 to 7:
            Let $p = i^{th}$ point of Sy, $q = (i+j)^{th}$ point of Sy.
            If distance p-q < current_best
                Best_pair = (p,q), current_best = distance p-q.

Why the 7?

We know that all pairs of points that lie entirely in Q or R, have a distance ≥ d [by the recursive calls].



For (p,q) to be the just next "best" (after (p1,q1) or (p2,q2)), we need distance p-q to be < d. So, the horizontal/vertical distance between p and q cannot be ≥d [otherwise distance p-q is ≥d].
So, for p, even in the worst case, we only need to consider the $2d \times d$ rectangle. Moreover, since any 2 points in Q or R are at least 'd' apart, so at best they can be at the corners of this rectangle. [If 2 points (both in Q or both in R) are inside this rectangle, then our work of d=min(d1,d2) is incorrect.]
Also, we are currently working on the case of "one point in Q and other in R".

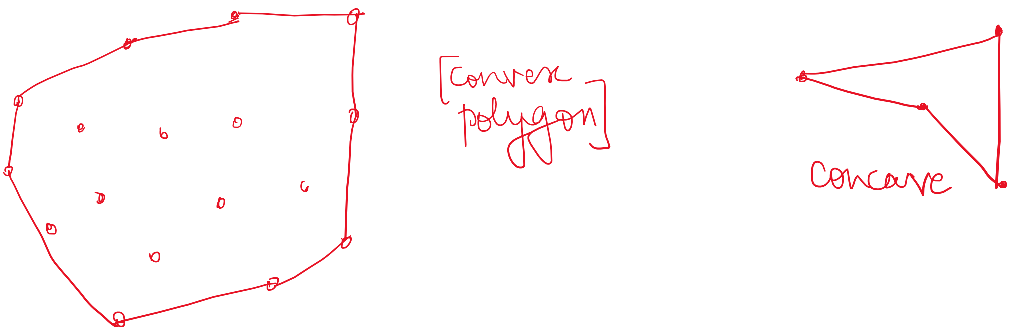
Since there can be overlapping points in Q and R at 'mid', so we need to consider the next 7 points in Sy. **It can't be that there are >7 points in this rectangle.**

Time complexity: T(n) = 2*T(n/2) + O(n). [O(n) worst case for searching in Sy.]
⇒ T(n) = O(n log n).

6. Convex Hull

[Quick Hull algorithm]

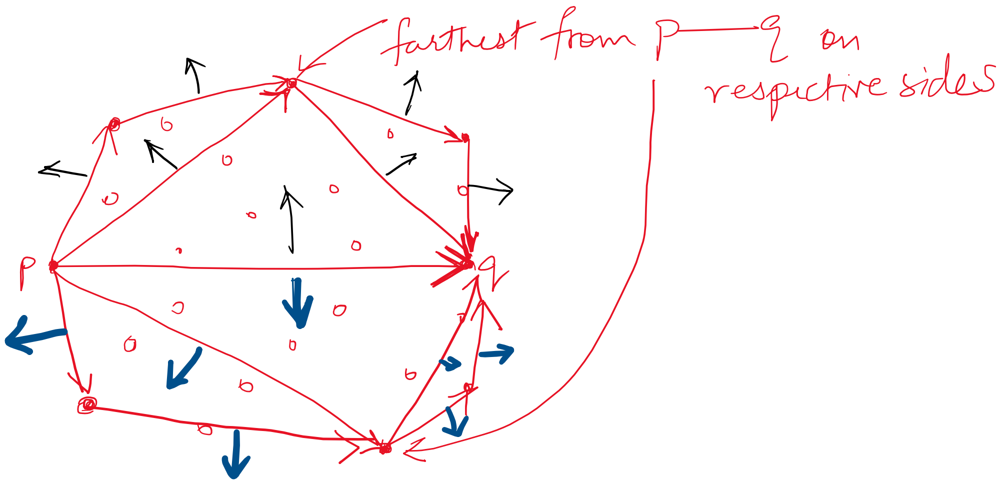
Given n points in a 2D plane, return the set of points in its convex hull.



[For convex hull, can think of the points as nails in a board, and releasing a stretched rubber-band completely outside all points. The final state of the band will be the convex hull.]

Algo 1: Consider tuples of size 3. They are nC3 in count, and form a triangle. For any point inside this triangle, this internal point cannot be part of the convex hull. Checking if a point is inside a triangle can be done in O(1). For 1 triangle, checking takes ~ O(n). We repeatedly remove points from our possible convex hull.
Time complexity = O(nC3*n) = O(n^4). [Note that $\binom{n}{3} = \frac{n(n-1)(n-2)}{3!}$.]

Algo 2: Quick hull.



At the start, select points p and q (all points are of the form (x, y)) that have the minimum and maximum x-coordinates respectively. These points will definitely be in the convex hull [can prove by contradiction].
Then, consider the vector (directed) p→q.
Find the farthest point in the anti-clockwise region of p→q [this point is shown by the top-most point in the diagram]. Call this point r. Recurse on vectors p→r and r→q [again in the anti-clockwise directions only].
Do similar operations for clockwise direction [shown by bottom region].

For finding the farthest point in the anti-clockwise direction:
Let the line p-q be $ax + by + c = 0$. Then for any point in the ACW region of p→q, $d = \frac{ax+by+c}{\sqrt{a^2+b^2}}$ will be positive. In the CW region, it will be negative. [Can think directly using the special case of x-axis: $y = 0$, and consider the point (3,4).]
[Note that the initial direction p→q (and not p←q) matters.]
Thus, for finding the farthest point in Case-1, just maximize d. For Case-2, minimize it.

Time complexity: T(n) = T(x) + T(n-x) + O(n), where x is the number of points in the ACW region of p→q [include p in "x" and q in "n-x", just for the sake of calculation].
If we are lucky, and x=n/2 at every step, then T(n)=2*T(n/2)+O(n) ⇒ T(n)=O(n log n). In the worst case, x can be 1 [just includes p], and we get T(n) = T(n-1)+O(n) = T(n-1)+cn. Solution is: T(n)=cn+c(n-1)+...+c = O(n^2).

7. CF Question

Jon fought bravely to rescue the wildlings who were attacked by the white-walkers at Hardhome. On his arrival, Sam tells him that he wants to go to Oldtown to train at the Citadel to become a maester, so he can return and take the deceased Aemon's place as maester of Castle Black. Jon agrees to Sam's proposal and Sam sets off his journey to the Citadel. However becoming a trainee at the Citadel is not a cakewalk and hence the maesters at the Citadel gave Sam a problem to test his eligibility.

Initially Sam has a list with a single element n. Then he has to perform certain operations on this list. In each operation Sam must remove any element x, such that $x > 1$, from the list and insert at the same position $\lfloor \frac{x}{2} \rfloor, x \mod 2, \lfloor \frac{x}{2} \rfloor$ sequentially. He must continue with these operations until all the elements in the list are either 0 or 1.

Now the masters want the total number of 1s in the range l to r (1-indexed). Sam wants to become a maester but unfortunately he cannot solve this problem. Can you help Sam to pass the eligibility test?

Input

The first line contains three integers n, l, r $(0 \le n < 2^{50}, 0 \le r - l \le 10^5, r \ge 1, l \ge 1)$ – initial element and the range l to r.

It is guaranteed that r is not greater than the length of the final list.

Output

Output the total number of 1s in the range l to r in the final sequence.

Examples

input	Copy
7 2 5	
output	Copy
4	

input	Copy
10 3 10	
output	Copy
5	

-

Note

Consider first example:

$$[7] \to [3,1,3] \to [1,1,1,1,3] \to [1,1,1,1,1,1,1] \to [1,1,1,1,1,1,1]$$

Elements on positions from 2-nd to 5-th in list is $[1, 1, 1, 1]$. The number of ones is 4.

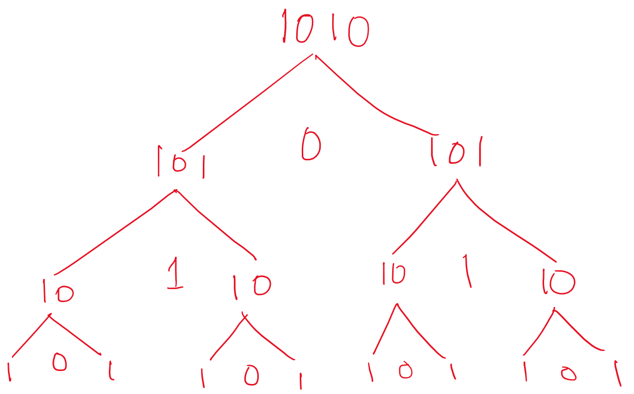
For the second example:

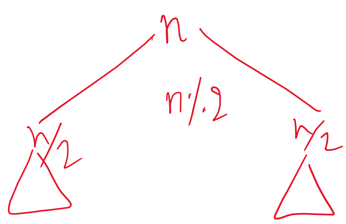
$$[10] \to [1,0,1,1,1,0,1,0,1,0,1,1,1,0,1]$$

Elements on positions from 3-rd to 10-th in list is $[1, 1, 1, 0, 1, 0, 1, 0]$. The number of ones is 5.

Code: code-for-2.cpp.

Can think by bit-representation.





This gives: 101_1_101_0_101_1_101 = 101110101011101

We need to find if a position 'x' in the final string is 0 or 1.
Clearly, we can use recursion [right figure].

To do the recursion, we need the position of the middle element [e.g., the top 0 (last bit of 1010) in the above example]. In the above example, we should be able to say that the 7th bit (0-indexed) is "0". For this, we need the length of the entire final array.

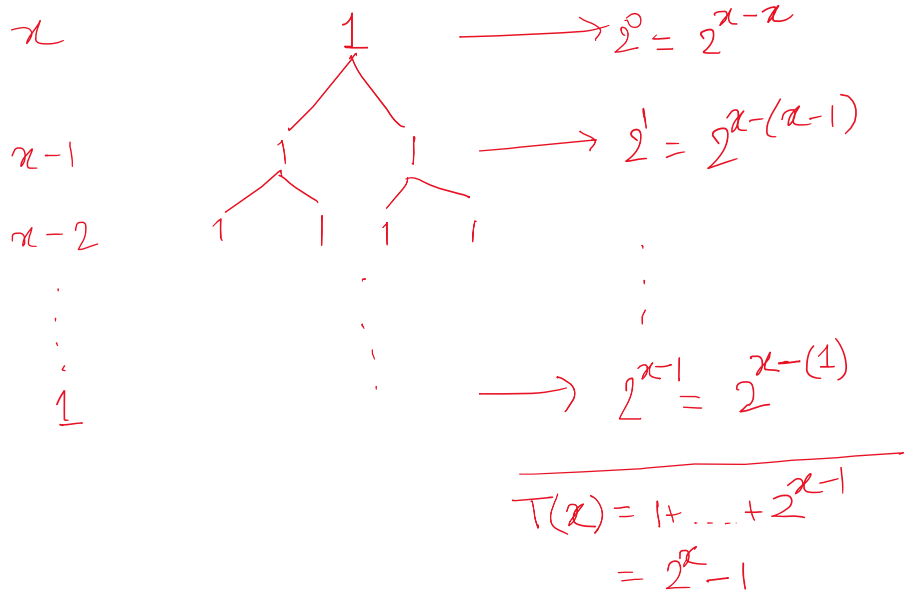
From the tree, we can see that the recurrence relation for T(n) = length of final array for n is

$$T(n) = 2T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + 1$$

To ease calculations, we re-write this recurrence in terms of the number of bits in n.
Let the number of bits in n be k. Then,

$$T(k) = 2 \cdot T(k-1) + 1$$

Where T(k) is the number of elements in the final array, that corresponds to a number having k bits. T(1)=1.
We can solve this using our recurrence tree:



Thus, T(n) = $2^{number\ of\ bits\ in\ n} - 1 = \mathbf{2^{\lfloor \log_2 n \rfloor + 1} - 1}$.
Consider n=10. Then number of bits = 4, and ⌊log2(n)⌋=3.

Note: T(n) is ~ n, and n can be as large as $2^{50} \sim 10^{15}$ [think this as $2^{10} \sim 1024 \sim 10^3$], so we cannot even store this array: it will be 10^{15} bools, i.e., 10^{15} bytes, that is 10^6 GB.

To calculate the log, we just use a function __builtin_clzll (count leading zeroes, ll).

Now we just need to use recursion.
Suppose we want to find the numbers corresponding to positions 2 and 9 (0-indexed) in the final array of 10.

Since 2<7, search in the left tree (call the function with '2' and 10/2=5). Since 9>7, search in the right tree (call the function with '9-7-1' and 10/2=5) [we had calculated this '7' earlier].

Code: code-for-1.cpp.

Problems

Codeforces: 1490D, 1167B, 1385D, 1676H2, 768B (above), 1111C.