1st Choice

Page No.
Date 30/12/21

Master Method

of Time Complexity

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

If $f(n) = O(n^c)$ where $c < \log_b a$

then $T(n) = \Theta(n^{\log_b a})$

If $f(n) = \Theta(n^c)$ where $c = \log_b a$

then $T(n) = \Theta(n^c \log n)$

If $f(n) = \Theta(n^c)$ where $c > \log_b a$

then $T(n) = \Theta(f(n))$

IB - Array - Find Duplicate in Array

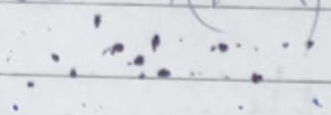
Given

Read only Array of (n+1) of no. b/w 1 to n

Find repeating no. in O(1) pass and less than O(n) space.

Soln: Analogy with linked list cycle detection.

- It is given no. (1 to n) so



i is linked to A[i] and then A[i] to A[A[i]].

Aise linked list ko karaah lenge.

Each number should have single occurance than it comes only once in our pass of linked list. Agar koi multiple times hain to toh A[i] cyclic linked list bana dega.

Now do, find cycle and then find cycle start.

floyd algorithm.

slow pointer = slow → next

fast pointer = fast → next → next

when slow = fast.

→ Loop found

To find start

fast = head.

while (slow != fast)

{

slow = slow → next

fast = fast → next

}

return slow;

MODULO INVERSE AND GCD

Euclidean Algorithm for GCD :-

→ If we subtract a smaller no. from a larger, GCD doesn't change. So, if we keep subtracting repeatedly the larger of two, we end up with GCD.

→ Now instead of subtraction, if we divide the smaller number, the algorithm stops when we find remainder 0.

```
int gcd(int a, int b)
{
    if(a==0)
        return b;
    return gcd(b%a, a);
}
```

Time Complexity

$O(\text{Log}\min(a,b))$

Extended Euclidean Algorithm:-

→ Extended euclidean algo is in addition to the GCD and the coefficients x and y such that:

$$ax + by = gcd(a,b)$$

Thus, he finds the coefficients by which the GCD of two numbers is expressed in terms of the number themselves.

Algorithm:-

Example:- Input a=30, b=20

output: GCD=10

x=1, y=-1

as 30x1 + 20x-1=10.

Input a=35, b=15

output: GCD=5

x=1, y=-2

as 35x1 + 15x-2=5.

When passing from a pair (a, b) to pair $(b \% a, a)$

Suppose we have found a solution to the (x_1, y_1) problem for a new pair $(b \% a, a)$

$$(b \% a) \cdot x_1 + a \cdot y_1 = g, \quad \text{---} (1)$$

and we would like get a solution (x, y) for our pair (a, b):

$$a \cdot x + b \cdot y = g \quad \text{---} (2)$$

put $b \% a = b - \left\lfloor \frac{b}{a} \right\rfloor \cdot a$ in (1)

$$g = \left(b - \left\lfloor \frac{b}{a} \right\rfloor \cdot a\right) \cdot x_1 + a \cdot y_1.$$

$$\Rightarrow g = b \cdot x_1 + a\left(y_1 - \left\lfloor \frac{b}{a} \right\rfloor \cdot x_1\right)$$

Comparing (1) and (2) we get

$$x = y_1 - \left\lfloor \frac{b}{a} \right\rfloor \cdot x_1$$

$$y = x_1$$

* Euclidean algorithm in this implementation works correctly also for negative numbers.

Implementation of extended Euclidean for GCD.

```
int gcd (int a, int b, int &x, int &y) {
    if (a==0) {
        x=0; y=1;
        return b;
    }
    int x1, y1;
    int d = gcd (b%a, a, x1, y1);
    x = y1 - (b/a) * x1
    y = x1
    return d;
}
```

Base case a=0, then gcd is b. The required x and y are 0 and 1.

x and y are being passed by reference.

★ The extended euclidean algorithm is particularly useful when a and b are co prime. Since x is modular multiplicative inverse of "a%b" and y is modular multiplicative inverse of "b%a".

Modular multiplicative inverse :-

Given two integers 'a' and 'm'. The modulo multiplicative inverse is an integer 'x' such that

$$ax \cong 1 \pmod{m}$$

Meaning :- $ax \% m = 1$

m (evenly) divides the quantity ax-1.
remainder after dividing ax by m is 1.

The multiplicative inverse of "a%m" exists if and only if a and m are relatively prime.

★ The value of x can never be 0. as $(a*0) \% m$ will never be 1.

1st Choice

Page No.
Date: 10/12/21

Method 1.

Loop 1 to m-1 check if((a * x) % m == 1)

Method 2 (When a and m are co-prime)

Extended
Use Eudidean that takes 'a' and 'b' finds their gcd and also finds 'x' and 'y' such that

$$ax + by = gcd(a,b) \quad \text{—} (1)$$

To find multiplicative inverse of 'a' under 'm', put b = m in (1) and a and m are relatively prime, hence put gcd as 1.

$$\Rightarrow ax + my = 1$$

Take modulo on both sides

$$\Rightarrow (ax + my) \% m = 1 \% m$$

Remove 'my' as my % m will always be 0.

Hence,

$\Rightarrow (ax) \% m = 1$

$\therefore ax \cong 1 \pmod{m}$

So 'x' that we can find using Extended euclidean algorithm is the multiplicative inverse of 'a'.

Algorithm

```
void modInverse(int a, int m)
{ int x, y;
  int g = gcdExtended(a, m, &x, &y)
  if(g != 1)
    print(Mod inverse does not exist)
  else
    int res = (x%m + m)%m;
    print("%d", res);
}
```

1st Choice

Page No.
Date

Same as befor

```
int gcdExtended(int a, int b, int *x, int *y)
{
    if(a==0)
    {
        *x=0, *y=1;
        return b;
    }
    int x1, y1;
    int gcd= gcdExtended(b%a, a, &x1, &y1);
    *x = y1 - (b/a) x1;
    *y= x1;
    return gcd;
}
```

Modular Division

Given a, b and m.

Compute (a/b) % m.

Meaning: find c, such that

$(b * c) \% m = a \% m$

Example: a = 8, b = 4, m = 5

Output: c = 2.

Input: a = 8, b = 3, m = 5

Output: c = 1

Note that (1 * 3) % 5 is same as 8 % 5

Input: a = 11, b = 4, m = 5

Output: 4

Note that (4 * 4) % 5 is same as 11 % 5.

We need to compute d.

$$\left(\frac{a}{b}\right) \% m = d \quad \text{—} (1)$$

where c is modulo multiplicative inverse of b. under m.

$$(b * c) \% m = 1 \quad \text{—} (2)$$

Rewriting (1) as

$$\left(\frac{a * 1}{b}\right) \% m = d$$

Substituting value of 1 from (2) into (1)

$$\Rightarrow \left(\frac{a * b * c}{b}\right) \% m = d$$

$$\Rightarrow (a * c) \% m = d$$

$\therefore$ (a * (modulo inverse of (b))) % m will be our answer.

Q:- Array description

CSES - DP

Given:- An Array with some values fixed and some those are variable are marked as 0. It is given that values of an array can be between 1 and a given m. Also given that diff b/w adjacent can be at most 1.

Find:- Number of possible arrays that fit the description

Example:
3, 5
2 0 2

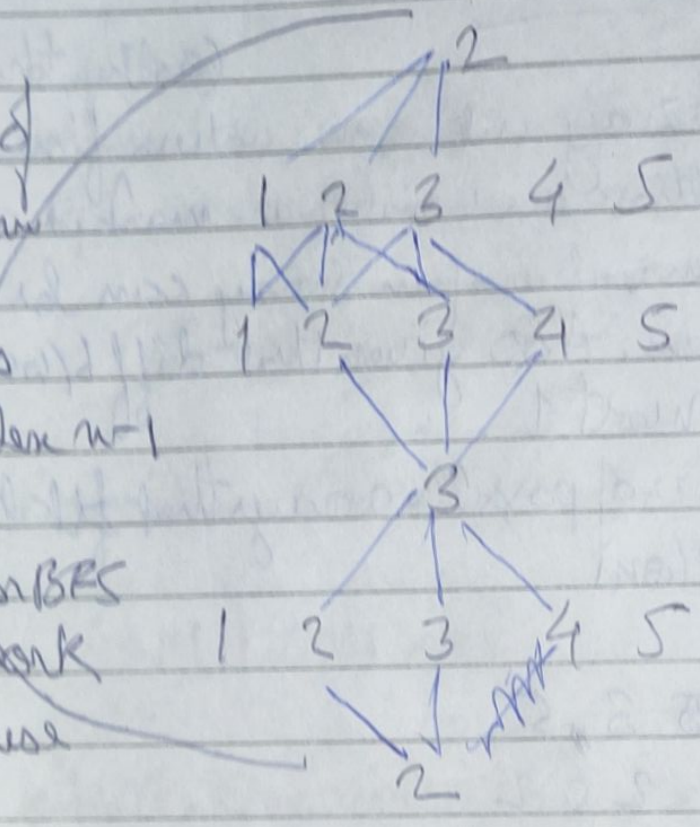
Output: 3

Possible arrays are [2,1,2], [2,2,2] and [2,3,2]

Constraints
N upto 10^5
$1 \leq m \leq 100$

Page No.
Date: 13/12/21

6 5
2 0 0 3 0 2



Number of paths from index 0 to index n-1

Brute for BFS won't work hence use DP.

dp[start][possible values] = 1

dp[i][j] = Σ dp[i-1][possible values]

Code:

```
for (i = 0 to <= n)
    if (value[i] == 0)
        for (j = 1 to <= m)
            dp[i][j] += (possible of (j-1, j-4) for j-1)
    else
        j = value[i]
        dp[i][j] += (possible of (j-1, j-4) for j-1)
```

Given array can end with any j.

So

```
cout << sum_{j=1}^{m} (dp[n][j]) % 2 << endl.
```

Time Complexity $O(n*m)$.

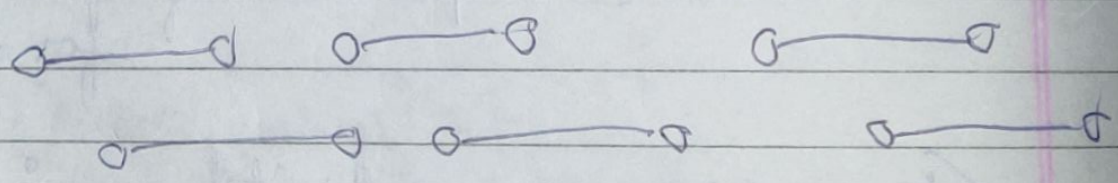
★ Projects CSES dp

Given
n projects: start day, end day, reward.

We can only attend one event / day. Reward is earned when we attend completely from s[i] to e[i] day.

Find maximum reward that can be earned.

Intuition: Sort w.r.t end day or start day for clarity.



Now there is choice to select each project and question asks for optimal answer. Hence, stay to use dp.

dp[i] stores maximum answer upto i^{th} project.

Now, if we do not select i^{th} project answer is simply

$$dp[i] = dp[i-1]$$

but if we select i^{th} process,

$$dp[i] = reward[i] + dp[\text{last process that can finish before } i]$$

To find last process that can finish before i since all our processes are sorted w.r.t end days.

We can use binary search to lower bound.

hence,

$$dp[i] = max(dp[i-1], reward[i] + dp[\text{lower bound of start}[i] \text{ from end in array}])$$

Complexity ($N \log N$)

1st Choice

Page No.
Date

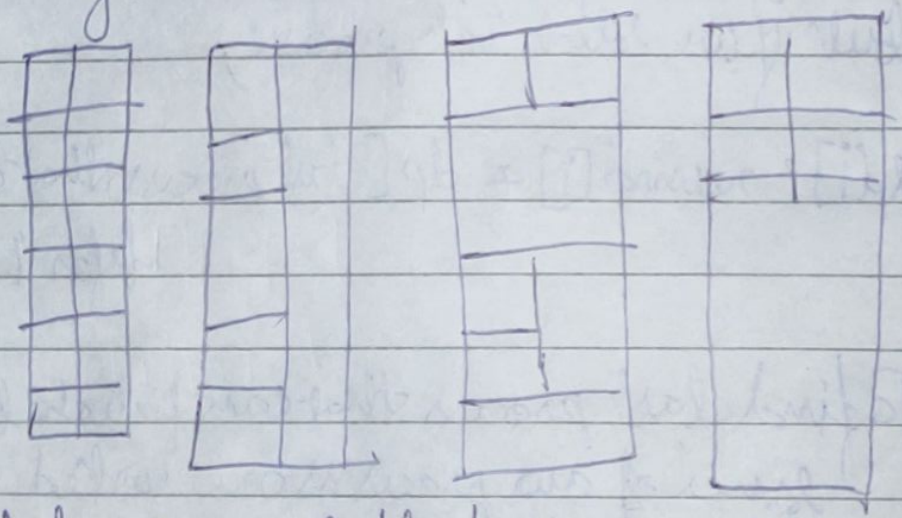
★ Counting Towers

DP CSES

Looks hard but simple.

Q: Build a tower of width 2 and height n. For a given n.

For example n=6.

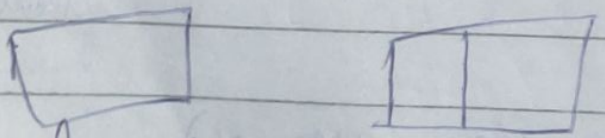


Find how many diff. towers can be made.

Approach:-

Lets try n build tower for n=1.

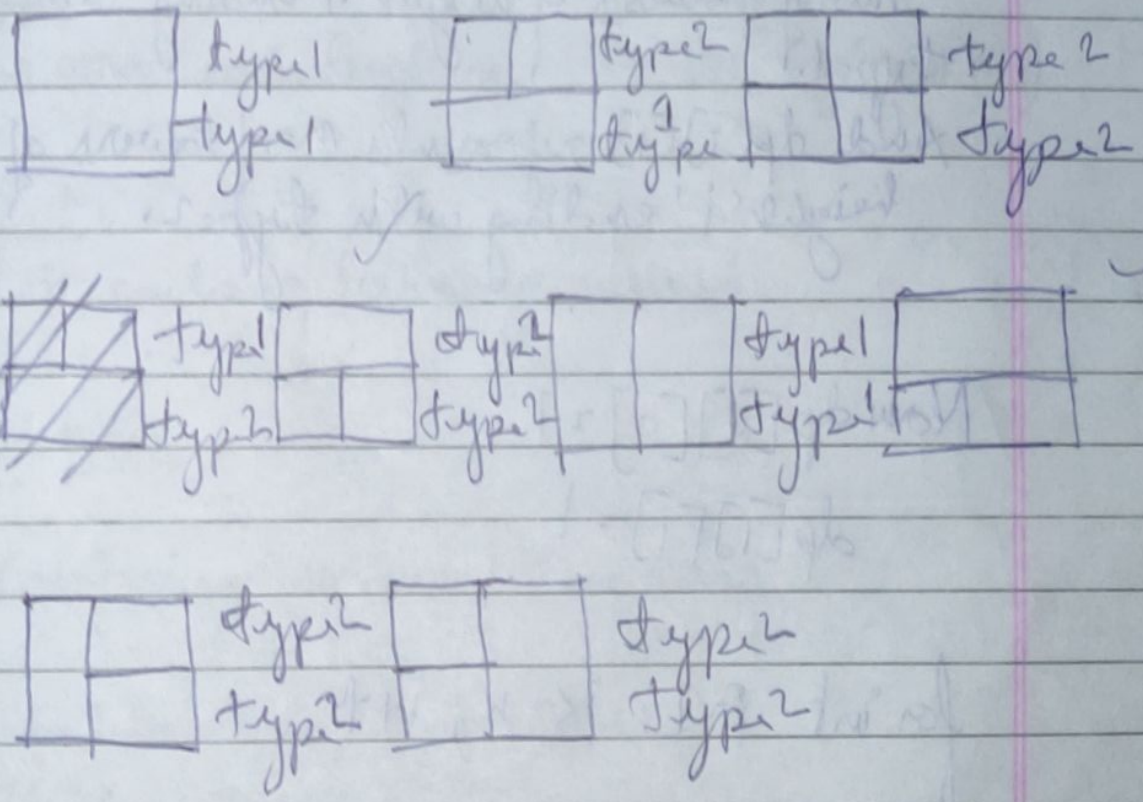
Output = 2



If we call them type 1 and type 2.

Let's check for n = 2.

Output = 8.



No. of ways to go from type1 to type1 = 2.

No. of ways to go from type1 to type2 = 1

No. of ways to go from type2 to type1 = 1

No. of ways to go from type2 to type2 = 4

Taking this induction forward,

dp[n+1][2], where dp[i][0] represents no. of towers of height 'i' ending with type 1.

And dp[i][1] represents no. of towers of height 'i' ending with type 2.

Now dp[1][0] = 1

dp[1][1] = 1

for int i=2; i<=n; i++

dp[i][0] += 2 * dp[i-1][0]

dp[i][0] += dp[i-1][1]

dp[i][1] += dp[i-1][0]

dp[i][1] += dp[i-1][1] * 4

print(dp[n][0] + dp[n][1]);

Median of two sorted array of equal size

Expected complexity

Given
Two array each length n. $O(\log(n))$

Output
Print median of both arrays combined.

Approach

Comparing medians of two arrays.

i) Calculate medians m_1 and m_2.
ii) If $(m_1 == m_2)$ return m_1.
iii) If $(m_1 > m_2)$
Then median is present in arr1[0] ... arr1[n/2]
either or arr2[n/2] ... arr2[n-1]

iv) If $m_2 > m_1$
Then median is present in arr1[n/2] ... arr[n-1]
either or arr2[0] ... arr2[n/2]

v) Repeat until size of both array becomes 2.
vi) median = $\frac{(\max(arr1[0], arr2[0]) + \min(arr1[1], arr2[1]))}{2}$

Median of two sorted arrays of different sizes

Given
2 Arrays size m and n. Both sorted.
Find median of both combined.

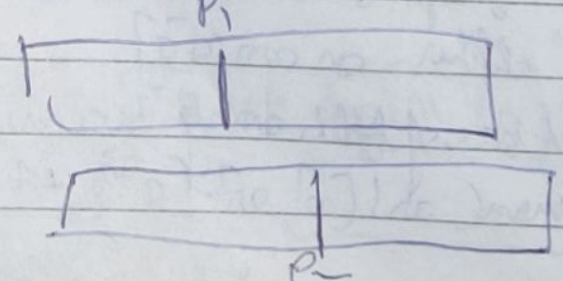
★ Best explanation → Tushar Roy Video.

Approach

Median in (m+n) combined array
= (m+n+1)/2 th elements

(+1 because it works well for both odd and even)

Idea is to find partitions in both arrays such that total no. of elements in left taken together is equal to the total number of elements in right.



There are corner cases that need to get handled

Also, our target is to achieve partitions such that all the elements to the left are less than or equal to all the elements to the right.

Now, if we do a binary search on one of the array, we know we want $(m+n+1)/2$ elements to the left of both partitions combined. Hence, index of partition in other array can be mathematically calculated as:

$$j = (m+n+1)/2 - (mid+1) - 1.$$

Now the only thing we're left with is checking elements to the left <= elements to the right.

while (low <= high)

```
if (nums1[i] <= nums2[j+1] && nums2[j] <= nums1[i+1])
    we have found the answer
else if (nums1[i] > nums2[i+1])
    we are too much to the right in first array
    high = i-1
else if (only else)
    low = i+1
```

1st Choice

Page No.
Date

* To handle corner cases, either take them separately or use INT_MIN and INT_MAX on the overflowing positions.

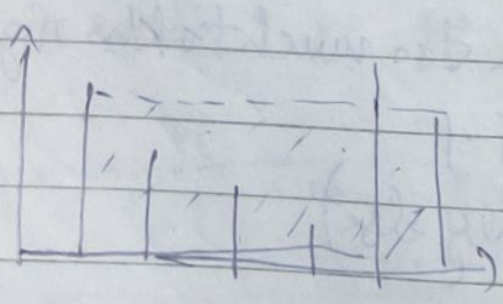
Answer will be max(nums1[i], nums2[j]) and min(nums1[i+1], nums2[j+1])

Container with most water

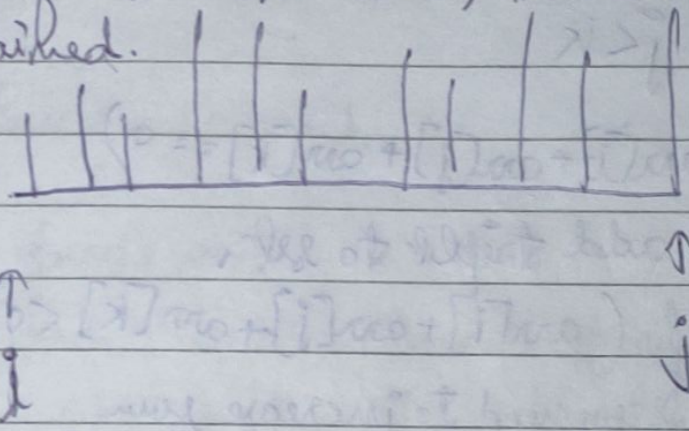
EC

Given

n non-negative integers $a_1, \ldots, a_n$ where each represents a point at co-ordinate (i, a_i). n vertical lines are drawn such that endpoints of the line i is at (i, a_i) and $(i, 0)$. Find two lines which together with the x-axis contains maximum area of water.



Approach:- Area formed between the lines is limited by the shorter line.
The farther the line, the more will be the area obtained.



Take two pointers one at start and other at the end. Move the pointer pointing to shorter line closer.
Compute max at each step.

3 Sum Problem

Given an array, find distinct sets such that arr[i] + arr[j] + arr[k] = 0.

Expected complexity $O(n^2)$ strictly less than $O(n^2 \log n)$.

Approach:- To find sets, we can change order of elements in array.
Exploiting that we can sort the array.
Now for each 'i' while traversing the array, we need to find 'j' and k such that $i < j < k$.

1st Choice

Page No.
Date

We can do this by using two pointer approach. Take $j = i+1$ and $k = n-1$.

while $j < k$

if (arr[i] + arr[j] + arr[k] == 0)
    add triplet to set.

else if (arr[i] + arr[j] + arr[k] < 0)
    need to increase sum
    hence j++.

else
    decrement sum
    k--;

To avoid duplicate triplets skip previous used for i and k.

S = 1 1 0 0 0

S+=S ⇒ 1 1 0 0 1 1 0 0 0

S1 ⇒ 0 0 1 0 1 0 1 0 1 0

S2 ⇒ 0 1 0 1 0 1 0 1 0 1

Slide window and compare the counts and answers at each step.

Minimum number of flips to make the Binary String alternating.

Given

Input: Binary string s.

Two operations:-

Type 1: Remove character from start of string and append it to the end of the string.

Type 2: Pick any character in s & flip its value.

Output: Minimum number of type-2 operation to make string alternating.

Example: s = 111000, answer = 2.

Approach: There are only two types of alternating strings possible 0101..., 1010...

Create both type of string S_1, S_2 of length $2*n$. (Because of this)

For the first operation we can simply do $S+=S$.

finally use sliding window of size n and compare with both s_1 and s_2.