binary search.

Q60. N^{th} root of any integers

→ given n and m, find $\sqrt[n]{m}$

we will use binary search, ~~it can~~

→ binary search can also be used when relation is monotonically increasing

→ search space: wherever answer lies.

[1 14 27]

low mid high

$mid = \frac{1+27}{2} = 14 \rightarrow 14 \times 14 \times 14 > 27$ $mid^3 >$ ans.

high = mid.

$mid = \frac{1+14}{2} = 7.5^3 \geq 27 \rightarrow$ high = mid = 7.5

$\frac{1+7.5}{2} = 4.25^3 > 27 \rightarrow$ high = mid = 4.25

$\frac{1+4.25}{2} = 2.625^3 < 27 \rightarrow$ low = mid = 2.625

$\frac{2.625+4.25}{2} = 3.4375^3 < 27 \rightarrow$ high = mid = 3.437

⋮ ⋮

this will be done until high − low > $1 \times e^{-6}$.

$\boxed{TC: \log_2(m \times 10^9) \times N}$

Q61. Matrix median.

→ given n×m matrix with all rows sorted and n×m = odd.

→ all the elements are from 1 - 10^9.

Approach 1.

naive approach.

→ iterate & put each element in ds, sort it and then return ans.

Date: Page:

TC: $O(N \times m) \log(N \times m)$

SC: $O(N \times m)$ ☹ optimize!!

Approach 2. Binary search.

→ this is also monotonic in nature.

→ search space ~~[1 - 10^9]~~ the $[1 - 10^9]$

$[1 \qquad\qquad\qquad\qquad 10^9]$

5×10^8
mid.

~~mid~~ = 1

→ go at each element and count number of elements <= mid

if count < mid ∴ high = mid - 1

else low = mid + 1

$mid = \frac{low + high}{2}$

Step 1. low = 1 high = 15

$mid = \frac{1+15}{2} = 8$. ... how many elements <= 8 ⇒ 7

need to move the mid in order to get median.

high = 7.

step 2. low = 1 high = 7

$mid = \frac{1+7}{2} = 4$. ... how many elements <= 4 ⇒ 4.

including 4, he has 4 numbers smaller/equal to it but this can't be median coz it says it has 4 elements lesser than equal to 4 and greater than equal to 4.

low = 4+1 = 5

Step 3. low = 5 high = 7.
mid = 6. how many no of lesser equal to 6 → 7.
~~low = 5~~ high = 5.

Step 4. low = 5 high 5
mid = 5. how many elements <= 5 → 4
low = 6.
low > high ? : breakpoint = low (Ans).

** binary search method to find elements in array <= mid.
→ apply binary search on every row.
how may elements <= mid, point to index just greater than the mid.
$mid1 = \frac{0+6}{2} = 3$. mid1 = 3 - a[mid] = 5

low = mid1+1, if mid1<= mid.
high = mid1-1, if mid1 > mid
if low > high, break
ans. low elements are lesser than equal to 5.

Q62. Single element in sorted array.
→ except one, all elements appears twice
→ only one element appears once.
[1, 1, 2, 3, 3, 4, 4, 8, 8] Ans 2

Approach 1
~~Appearing~~ Linear Iterating and doing XOR operations
Ans: element that appears once.

TC → O(N)
SC → O(1)

Approach 2 Binary search
→ find the break point such that all elements at left appears left and right element occurs once.
eles → all elements except one have one even index & one odd index.
low = 0, high = n-2
mid = (low + high)/2

better condition
if nums[mid] = nums[mid ± 1]

left half
1st inst: even
2nd inst: odd

right half.
1st instance: odd
2nd instance: even.

check for left half, if it doesnt, high = mid-1
else low = mid + 1
do this untill high >= low.

TC → O(log N) SC → O(1)

Q63. search in rotated sorted array.
→ ~~array~~ sorted array is being ~~for~~ rotated and now you need to find element
→ if you can't find return -1

Approach 1 Linear search
TC → O(N) SC → O(1)

Approach 2. Binary search
low = 0, high = n-1 mid = (low + high)/2
if mid != target, find which half is sorted and check whether target lies in range.
if it doesn't lies, low = mid + 1.
if it does then high = mid.
TC → O(log N)
SC → O(1)

Q64. Median of 2 sorted Array.
→ given couple of array (sorted)
→ find median.

Approach 1 Naive solution
→ merge both the arrays using merge() from merge sort
→ find length and get median.

TC → O(N1+N2)
SC → O(N1+N2) ☹

Approach 2. (slight optimized). Naive solution
→ find length of both arrays, find its median position
→ iterate until you don't reach median position through merge

TC → O(N1+N2)
SC → O(1) 😐

Approach 3. Binary search (optimized)
→ Divide both arrays in 2 parts as well as in equal parts such that $min(l_{21}) \geq max(l_{12})$ and $min(l_{22}) \geq max(l_{11})$
→ decide where to do partition
→ shift partition so as to get correct partition

Dry Run.

arr 1 → | 7 | 12 | 14 | 15 |

arr 2 → | 1 | 2 | 3 | 4 | 9 | 11 |

total elements = 4+6 = 10
partition = 5 − 5

elements picked → 0 (min) − 4 (max)

cut 1 = $\frac{0+4}{2}$ = 2.

pick 2 elements from arr 1.

cut 2 = 5 − 2 = 3.

pick 3 elements from arr 2

$l_1 = 12$ $r_1 = 14$ $l_1 \leq r_1$ ✓ $l_1 \leq r_2$ X

$l_2 = 3$ $r_2 = 4$ $l_2 \leq r_2$ ✓ $l_2 \leq r_1$ ✓

reduce l_1 and inc. r_2.

high = mid − 1 ⇒ 1.

cut 1 = $\frac{0+1}{2}$ = 0

INT_MAX

$l_1 = X$ $r_1 = 7$.

cut 2 = $\frac{5-0}{2}$ = 5

$l_2 = 9$ $r_2 = 11$

$l_1 \leq r_1$ ✓ $l_1 \leq r_2$ ✓

$l_2 \leq r_2$ ✓ $l_2 \leq r_1$ X

decrease l_2 and increase l_1.

low = mid + 1. = 1

cut1 = 1

cut2 = 4.

7 | 12 14 15

1 2 3 4 | 9 11

all condition satisfies.

median ⇒ max (7, 4) + min (12, 9)

/INT/MIN

l_1 = arr1[cut1 - 1] r_1 = arr1[cut1] /INT=MAX

l_2 = arr2[cut2 - 1] /INT/MIN r_2 = arr2[cut2] /INT=MAX

#2.

total elements = odd.

$\frac{n_1+n_2+1}{2}$ = 6. (length of left half.

median = max (l_1, l_2)

TC ⇒ $O(\log_2(\min(n_1, n_2)))$

SC ⇒ $O(1)$

Q. 65. Find k^{th} element of 2 sorted arrays

Approach 1 Naive solution.

2 pointer approach, take counter ~~arr~~ variable and 2 pointers at beginning of array, now move pointer which is smaller and increase counter variable by 1

→ moment counter = k, return the smaller element.

TC ⇒ O(K) SC ⇒ O(1)

Approach 2. (binary search)

partition must be done such that k elements on left, largest of them will be answer

7	12	14	15

K = 5

1	2	3	4	9	11

min = 0, max = 4.
low → high →

cut1 = $\frac{min + max}{2} = \frac{0+4}{2} = 2$. cut2 = 5 − 2 = 3

7 12	14 15
1 2 3	4 9 11

is this valid partition ✗
$l_1 \leq r_2$ ✗ → 12 ≤ 4 ✗.

thus high = mid − 1. = 1.
low = 0, high = 1
cut 1 = 0 cut 2 = 5 − 0 = 5.

*	7 12 14 15
1 2 3	11
4 9	

is this valid partition ✗
$l_2 \leq r_1$ ✗.

thus low = mid + 1 = 1.
low = 1 high = 1
cut 1 = 1. cut 2 = 4.

7	12 14 15
1 2 3 4	9 11

is this valid partition ✓
yes.

kth = max(l_1, l_2) = 7.

TC ⇒ O(log(min(n, m))
SC ⇒ O(1)

166. Allocate Books.

→ given n book & every book has no. of pages
→ given students,
→ allocate books to student such that maximum no. of pages allocated to student is minimum

condition
① a book will be allocated to one student
② each student must get one book minimum
③ allotment must be in contiguous order

Approach → Binary search

stu = 2
arr = [12 34 67 90]

[12 203]
low high

mid = $\frac{12+203}{2}$ = 107

1 → 12 + 34
2 → 67
3 → 90

if (students < exp.division)
low = mid + 1
else ~~high = mid - 1~~ if (students == exp.division)
res = min (res, mid)
high = mid - 1

```
while (low <= high) {
  mid = (low + high) >> 1
  if (allocation possible (mid)) {
    res = mid;
    high = mid - 1; }
  else
    low = mid + 1;
}
```

```
bool isPossible (int barrier) {
    allocated = 1, pages = 0
    for (i = 0 ; i < n ; i++) {
        if (arr[i] > barrier) return false;
        if (pages + arr[i] > barrier)
        {
            allocated ++;
            pages = a[i]
        }
        else pages += arr[i];
    if (allocated > k)
        return false.
    return true.
```

TC → $\log N \times N$

SC → $O(1)$

Q6. Aggressive cows

→ given n stalls and array containing location of n stalls, not necessarily sorted.

→ given c cows → {2, n}

Task: Place ~~c~~ ~~counts such that~~ cows in n stall such that we get largest minimal distance.

Approach 1. Recursive solution

→ try out all possible solution

→ exponential time complexity

Approach 2. Brute force.

arrangement

minimal dist [1 2 4 8 9]

(1) ✓ $C_1 \xleftarrow{1} C_2$ C_4

(2) ✓ $C_1 \leftarrow 3 \rightarrow C_2 \xleftarrow{4} C_3$

(3) ✓ ||

(4) ✗ $C_1 \longleftarrow 7 \longrightarrow C_2 \xleftarrow{1} C_3$

Ans 3.

TC = $O(N^2)$

* monotonic solution.

Make sure to sort arrays

Approach 3 Binary search.

search space => [1 - 8]

[1 8]

low high.

Step 1 low

mid = $\frac{1+8}{2}$ = 4

if we can place cows at dist 4

C_1 x x C_2 x

[1 2 4 8 9] C_3 is left not possible

high = mid - 1.

Step 2.

mid = $\frac{1+3}{2}$ = 2.

C_1 x C_2 C_3

[1 2 4 8 9] we are able to place cows, thus we place answer in result.

res = INT_MIN

res = max(res, mid)

low = mid + 1

Step 3.

mid = $\frac{3+3}{2}$ = 3

C_1 x C_2 C_3

[1 2 4 8 9] we are able to place

res = 3.

low = mid + 1. = [4 3]

low high

Terminate (low > high)

Ans res.

code

```
low = 1 , high = a[n-1] - a[0]
while (low <= high)
{
    mid = (low + high) >> 1
    if (canplacecows (mid))
        res = mid
        low = mid + 1.
    else
        high = mid - 1.
}.
```

c_1

1 2 4 8 9

```
bool canplacecows (arr, n, cows, dist) {
  {
    cnt = 1 , coord = a[0]
    for (i = 1 ; i < n ; i++) {
        if (a[i] - coord >= dist) {
            cnt ++ ;
            coord = a[i] ;
        }
        if (cows == cnt) {
            return true ; }.
    }
    return false ;
  }
```

TC => O(N log N)
SC => O(1)

Bit Manipulation

Q68. check if number is power of 2 or not.

Approach 1 general solution.

→ if n == odd, return false.

→ if n == even, divide n by 2 and repeat process untill result is 1 or reminder = 0

TC = O(log N)

Date : Page :

Approach 2 Bit manipulation

Numbers	~~00001000~~ BR	
8	00001000	$\rightarrow 2^3$
32	00100000	$\rightarrow 2^5$
9	00001001	
64	01000000	$\rightarrow 2^6$
23	00010111	
65	01000001	
128	10000000	$\rightarrow 2^7$

only one bit set in all powers of 2.
find the count of ~~each number having~~ set bits in number, if it is one, the number is power of 2.

TC => O(log n)
SC => O(1)

Approach 3. Optimized.
if n is power of 2, then (n-1) have (logn-1) bits set at that ~~end~~ too all least significant bit and thus.

(n) & (n-1) == 0 if n is power of 2.
else its' not power of 2

TC => O(1) SC => O(1)

069. Count total set bits

Approach 1.: Find total no. of bits in binary representation by dividing number by 2 and ~~count~~ ~~if~~ incrementing count untill n != 0.

```
while (n>=1){
  cnt += (n%2);
  n/=2;
}
```

TC = O(log N) SC = O(1)

Q70. divide 2 integers without / operation
→ no use of divide operation.

Approach 1 repetitive subtraction (large TC)

Approach 2.
→ while we can subtract divisor from dividend
① yes, double divisor (left shift)
② increment the count
③ check again
→ add count in result
→ subtract temporary variable from dividend

Q71 Power set
→ generate all possible ~~subsequence of any string~~ subarrays of given array

no	2	1	0	ans
0	0	0	0	""
1	0	0	1	"a"
2	0	1	0	"b"
3	0	1	1	"ab"
4	1	0	0	"c"
5	1	0	1	"ac"
6	1	1	0	"bc"
7	1	1	1	"abc"

0 → not picking
1 → picking

0 - $2^n - 1$
n = length

```
for (num = (0 → 2^n - 1))
  for (i = (0 - (n-1)))
    if (num & (1<<i))
      sub += S[i]
print (sub)
```

TC ⇒ $O(2^n) \times O(n)$

Date: Page:

Q72. Find MSB ~~in O(1)~~

Approach 1.

dividing number by 2 untill it doesn't becomes 0, side-by-side incrementing count. counter value will tell the position of MSB.

TC ⇒ $O(\log_2 n)$

Approach 2

→ make all bits set, increment n by 1, and do left shift operator in order to get no. whose bit is set at the position

```
n |= n>>1
n |= n>>2
n |= n>>4
n |= n>>8
n |= n>>16
n+1.
return (n>>1)
```

TC = O(1)

Approach 3.

→ take log of the number and convert into int.

TC = O(1)

Q73. Find square of number without using '/', '*', ...

Approach 1. repeatitive addition

TC ⇒ O(n)

Approach 2.

```
square (n) {
    if (n==0) return 0;
    if (n<0) n = -n;
    int x = n>>1
    if (n&1) return (square(x)<<2 + (x<<2) + 1);
    else return (square(x) <<2)
```

TC => O(log n)