Rotated Binary Search

Revision:

arr = [2, 4, 5, 7, 8, 9, 10, 12]

After 1 rotation

[12 ..... 9, 10]

After 2 rotations

[10, 12 ..... 8, 9]

2 Approach :-

1) FIND (PIVOT) →

2)

↓

largest number in the array from where your next numbers are asc ↑

[3, 4, 5, 0, 1, 2]

↓ pivot

Sorted array | Sorted array

Thought process

* find pivot
* search in first half (0, pivot)
* else
  search in second half (pivot+1, end)

How to think in interview

Q: FIND PIVOT

0 1 2 3 4 5 6 7

4, 3, 4, 5, 6, 7, 0, 1, 2, 4

s m l e

asc desc asc

* Apply binary search.

Ans when ? condition for pivot → 2 cases.

CASE 1: mid > mid + 1 element

ans = pivot

↑ final algo BS.)

CASE 2: mid < mid - 1 element

ans = pivot

CASE 3: start >= mid

3, 4, 5, 6, 7, 0, 1, 2

s

you can also ~~take~~ take about end and mid

(S >= M)

0 1 2 3 4 5 6

4, 5, 6, 3, 2, 1, 0.

s m e

Smaller than S.

(4 > 3) ignore right portion

start > mid

since we are looking for peak.

e = m - 1;

case 4: start < mid one of the case should be seen.

3, 4, 5, 6, 2

(s) (m)

(s < m)

start = mid + 1

if this was pivot it would have been returned in case

hence between that bigger no.s lie ahead. Hence, ignore mid &

(RBS)

static int pivot (int ~~nums~~ arr)

start = 0

end = arr.length - 1

while (start <= end)

mid = (start + end)/2;

&& (mid<end) if arr(mid) > arr[mid+1]

~~else~~ if arr[mid-1] < arr[mid] < arr[mid+1]

return arr[mid]

case 5 if (mid <= start)

end = mid - 1

if middle was pivot it would have been caught previous

- therefore pivot lies ~~else~~ after mid

else

start = mid + 1

return -1;

~~in binary search~~

```
int search(int[] nums, int target) {
    int start = 0, end = nums.length - 1;
    int pivot = find pivot(nums[]);
```

if you don't find pivot then arr is not rotated

```
if (pivot == -1)
    binarysearch(start, end, nums, target);
```

If if pivot is found then we have two ASC Arrays to search Target.

```
if nums[pivot == target]
    return pivot;
```

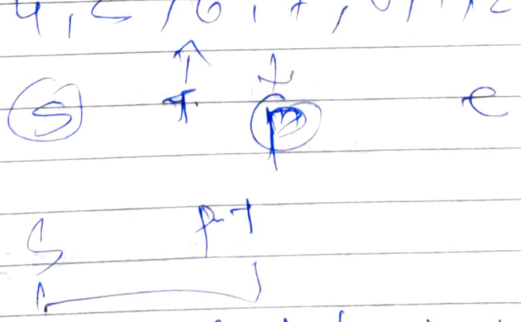
Till now we saw how to find pivot now we will see how to find target using pivot?

3 cases; →

CASE 1: pivot == target

ans = pivot

CASE 2: target element > start element



searchspace = (start, pivot - 1)

if target < start way to search elements is smaller than start

so end = pivot - 1

target < start element

we know all element's pivot should be greater than target

search space
(pivot + 1, end)

3:06:18

FIND NUMBER OF COUNT OF TIMES ARRAY IS ROTATED.

arr = [4,5,6,7,0,1,2]

pivot times array is rotated

0,1,2,4,5,6,7

4,5,6,7,0,1,2,4,5,6,7

rotated 4 times from the sorted array.

pivot = index of

every time we rotate the array pivot

rotation number = number of elements we have from start till pivot + 1

0,1,2,4,5,6,7

7,0,1,2,4,5,6 (1) roller pivot = 0th index

6,7,0,1,2,4,5 (2) roller pivot = 1st index

5,6,7,0,1,2,4 (3) roller pivot = 2nd index

num of rotations = pivot + 1 → index

every time we rotate pivot is coming from elements shifting

[mid end] a[mid] > a[mid+1]

return mid

7,6,0,4,2

mid > start

(arr[mid] < arr[mid+1]) 4 5,6,7

return (m-1) + 1

5,6,7,0,1,2

s m e

a[mid] <= start

end = mid-1

else

start = mid+1;

int pivot = 1

no. of rotation = p+1 return 7

rest

if p return

19:50 29th

Google

410. Split Array Largest Sum.
(LC)

arr = [7, 2, 5, 10, 8] m = 2
↓
no of parts

[7, 2, 5, 10]

[7, 2, 5]

[7, 2]

subarray → continuous

[7, 2, 5, 10, 8] m = 2

		largest
7, 2, 5, 10	8	24 ✓ 8
7, 2, 5	10, 8	14 18
7, 2	5, 10, 8	9 23
7	2, 5, 10, 8	7 25

min

Minimize the largest sum

take sum of every subarray
take max
do it for every case

① min number of partitions
max = 1

② max = (n) [3, 4, 1, 2]
[3], [4], [1], [2]

→ ans
subarray sum of
entire array

max value of any of question = case 1

[10, 32]
↓ ↓
UL LL

SEARCHING IN 2D ARRAYS.

BINARY SEARCH.

```
for( r=0; r<n; r++)
    for( c=0; c<n; c++)
        if a[r][c] = target.
            return r[i][j]
            (1,2)
return -1.
```

worst case (N*N) time complexity. O(N²) → O(M*N) → C.

Q: Matrix is sorted in a row wise and column wise order.

10, 20, 30, 40
15, 25, 35, 45 → sorted
28, 29, 37, 49
33, 34, 38, 50

sorted

search for target = 37)

not efficient.

How to minimize search space ?

If binary data is given

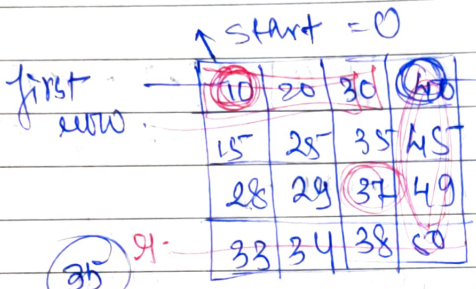
How many cases

CASE 1: if (target == element)
ans found

CASE 2: if (element < target)

CASE 3:
if (target > element)

In matrix you have to try to minimize row and cols to reduce search space.



target > elem
r++
else
c--

if (e > t)

Searching in 2D array

target < elem
row ++
else

if target == element
return {i, j};

if target < a[r][c]
then
c--
else
row++

if target < element
c--;

Target > element
r++

(Angular / React / Node.js)

20 E → MATH . LS, BS RKP, Two pointer
30 M → Greedy
50 H → DP.

JAVA OOPS NIGHT

this keyword → used to call another constructors
① of the same class.

this();
this.age = age
this.name = name

② used to access the parameters of same class.

Static keyword →
allows user to access ~~static~~ variables and methods without creating object of the class

POLYMORPHISM
↓
greek word
↓
many forms.

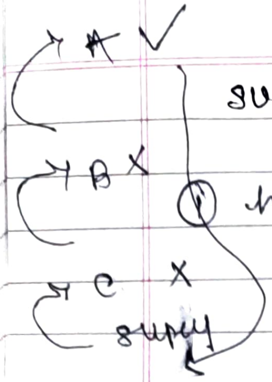
calc(5);
calc

int calc();
int cal(int x);

Compile time polymorphism
function overloading

INHERITANCE

class Person { }
class Developer inherits Person
↓ ↓
child. parent



public Person { }

SUPER KEYWORD

(1) helps to call constructor of the parent class

super("Amy", 24)

this keyword = current property access
super keyword = parent property access

extends → keywords tells which class inherits which class

PRO

ABS → (ON) OFF

MANUAL

DRIVER LANG OFF

Runtime polymorphism

class Parent {
void walk();

class Developer extends Parents
void wait (int age)

Object

calling functions in different

Encapsulation.

Packages & access modifiers
↓
naming convention
↓
encapsulation
↓
lowercase

access → private ↔ value
only in class
package ✗

protected — parent ↔ children
only public for children

ENCAPSULATION.

getters and setters

ABSTRACTION
DATA HIDING
Complexity reduces.

user — ~~Complexity hiding~~ → we don't want to know
↑
remove complexity from users.

ABSTRACT, INTERFACE

class Couch
jute price;
~~int~~
string brand;

Audi ~~extends~~ Car

BMW extend cars

Abstract class Car
abstract void car()
↓
mandatory defined in child class

INTERFACES —

interface Car {

void start();

↓

all methods by default

public abstract

in classes (abstract) we

may define methods

but in interfaces it doesn't allow

interface BMW implements car

@Override → spell check

Multiple

Inner ↓

One class can't have

more than 1

parent

abstract

↓

initialization

not concept

FORZA HORIZON 4

Bumps → DAMPING →

FRONT (Max - Min) x Wo/o + Min = Setting

Rear. (60%.

SPRINGS = 900

800

ANTI ROW - 38

29

analysis.

can we create one

if we say about target by we leave this places. can we do this?

Continuation

of course we can.

Searching in Binary

in 2D Array.

Search in a sorted matrix

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

first element x

convert it to array and apply binary search.

start

end

target == element

target > element start++

else end--;

- take middle row.

reduce rows/

vice versa.

① if element == target

② element > target

ignore rows

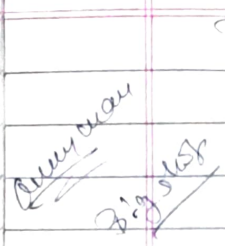
if element < target

ignore above rows

only two rows will be left

check next column has the ans

target = ?



if (target <= matrix[rstart][cmid-1]

if target > matrix[rstart][mid+1])

SORTING

3, 1, 5, 4, 2

BUBBLE SORT

① In every step we compare adjacent elements.

if (1st element > 2nd element)
SWAP!

1, 3, 5, 4, 2

1, 3, 4, 5, 2

1st step done → 1, 3, 4, 2, 5

why Bubble sort

* largest element came in the end

2nd time 1, 3, 4, 2, 5

1, 3, 2, 4, 5

Now compare only till last - i

** 2nd last element comes to 2nd last pos

3rd time 1, 2, 3, 4, 5

3rd last element — " — 3rd last pos

** Also known as syncing sort. AKA. exchange sort

with 1st 2nd 3rd pass we can ignore elements from the (last - 1) index as they have been kept in resp. position

i 0 1 2 3 4

j 1 2 3 4

5, 1, 5, 4, 2

no need to sort

is a[i] > a[j]

first pass i=0 { j comp (n-1) times for

1, 3, 2, 4, 5

i=1

i=2

j will only check

(j = length - i)

j=0; j < length - i - 1

you can go till end but its not helping.

O(1)

f(n)

SPACE AND TIME COMPLEXITY

↓ constant O(1)

→ copy Array not req

→ in place sorting algorithm
↳ no new array req

↓ How many comp

Best case
O(n) → sorted

worst case
$O(n^2)$
↳ sorted in opp site order

number of comparisions

WC (0000 2000

BC (100 50

size of i/p

Time complexity → As the i/p grows how the ~~Input~~ time grows.

Best case

1, 2, 3, 4, 5

i=0

if j never swaps for i

for i=

it means array is sorted

Best case comp (N-1)

In time complexity we ignore constants

because -

* we do not want the exact time
* we want relationship b/w size of
* we can remove non dominating i/p & terms since i/p size will be usually ~~time~~ moreover with large input size

constants really are really small factors negligible.

classmate
Date
Page

(WORST CASE) $O(N^2)$

5,4,3,2,1 — 1st $i=0$

4,5,3,2,1 — $j=0$

4,3,5,2,1 — $=1$

4,3,2,5,1 — $=2$

4,3,2,1,5 → — $=3$ → (length - i) > j

$i=1$

$i=1$ — 4,3,2,1,5 — $j=0$

3,4,2,1,5 — 1

3,2,4,1,5 — 2

3,2,1,4,5 — 3

$i=2$ — 3,2,1,4,5 — $j=0$

2,3,1,4,5 — $j=1$

2,1,3,4,5

$i=3$ — 2,1,3,4,5 — $j=0$

1,2,3,4,5

sorted

$i=3$ $j=1$ ⇒ stop

(n-1) swaps

(n-2) swaps

(n-3) swaps

(n-4) swaps

WITH NORMAL (N) WHEN dealing with computers

$i = (0 \longrightarrow n-1)$

$j = 0 \longrightarrow (n-i-1)$ ×

$(n-1) + (n-2) + (n-3) + (n-4) \cdots (n-n)$

total

$4n - 10$

$O(n)$

$O\left(\frac{4n-n^2}{2}\right)$

* less dominating terms are removed

$4n$

$(n-4)+(n-3)+(n-2)+(n-1)$

$4n - 1 - 2 - 3 - 4$

$4n - (1+2+3+4 \cdots n)$

$4n - \frac{n(n+1)}{2}$

$\frac{8n - n^2 + n}{2}$ $O(n^2)$

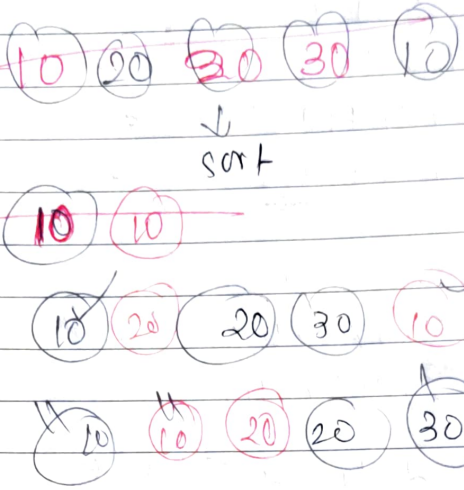
SPECIAL THINGS →

Stable searching algorithms

↓ what is?

order should be same

when value is same

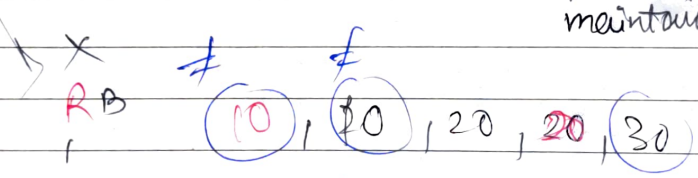


B R

B R

In original array ball of 10 was before red ball

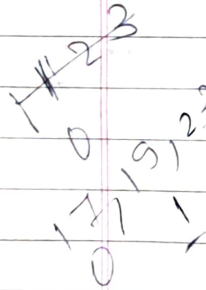
in sorted order is maintained



RB

Stable → original order maintained for equal values

unstable → change in order for equal values



passive optimisation

the loop should break if

× no swap occurs at the first step ie for i=0 if

break

boolean checkSwap = false; no swaps

if(swap) print (already sorted) if swap (swap = true)

SELECTION SORT

select element & put it in right position

0 1 2 3 4

4, 5, 1, 2, 3

2 ways →

select smallest put in correct index

4, 3, 1, 2, 5

2, 3, 1, 4, 5

2, 1, 3, 4, 5

1, 2, 3, 4, 5

you can also select nth element and swap it with corresponding last index

4, 5, 1, 2, 3

1, 5, 4, 2, 3

1, 2, 3, 4, 5

1st step find max element in array

$(n-1)$ comparisions.

$(n-2)$

$(n-3)$

:

1 → second last comp

0 comparision

Total comp. $0 + 1 + 2 + \dots (n-1)$ 1 to n

$(n-1) + \frac{n(n+1)}{2}$

$\frac{n(n-1)}{2}$ $\frac{n^2 - n}{2}$ $O(n^2)$ $n\frac{(n-1)}{2}$

Why constant remove less dominating term

1, 2, 3, 4, 5. (max) = 5

5 4 3 1 2

max, temp

WORST CASE = $O(N)^2$

Best Case = $O(N^2)$

Stability = NOT STABLE

Uses → Perform well on small list

already in correct position

way to

5 4 1 2 3

max, temp

max = 5

arr[i] = 3 arr[arr.length - 1 - i]

arr[arr.length - 1] = max

2147483647

SELECTION SORT Apna College

[4 1 9 2 3 6] n=6

smallest element

1 4 9 2 3 6

1 2 9 4 3 6

1 2 3 4 9 6

1 2 3 4 6 9

O(n-1) of inner loop

1 swap

back to Kunal Kushwaha.

static void selection(int[] arr) {

for(int i=0; i< arr.length; i++)

we need two things.

~~maxIndex~~ = ?

~~last index~~ = ? (arr.length - 1 - i)

maxIndex = getMaxIndex(arr, 0, last)

int maxIndex(int[] arr, int start, int end)

static void swap(int ~~start~~ first, second) {arr

int temp = arr[first];

arr[first] = arr[second]

arr[second] = ~~start~~ temp;

INSERTION SORT

partially sorting the array

parts, parts

i=0 to 1

i=0 to 2

3

:

i=0 to n

5, 3, 4, 1, 2

put elements in left side of correct position

3, 4, 5, 1, 2

pass = 1 step

i=0

5, 3, 4, 1, 2

i j
0 1

i=0
j=i+1

when i = 0 1
it will be sorted 1st pass index 1

when i = 2
it ——— 4 ——— index 2

i will run
=0 to (n-2)

to understand when array is going

5, 3, 4, 1, 2

i<N-2 j>0

if(a[j] > a[j-1])
break; no need to check as its already sorted

if (a[j] < a[j-1])
swap;
else
break; // left side sorted

i=4 i=5
if 1 out of bound
i<N-2

COMPLEXITY ANALYSIS

worst case

desc sorted

5, 4, 3, 2, 1

$1, 2, \ldots, n = \frac{n(n+1)}{2} \quad \frac{n^2+n}{2}$

N → total number of comparisions

$O(N^2)$

Best case

array sorted

1, 2, 3, 4, 5

break $O(N)$

100

100 comp

why use insertion sort

adaptive

if, $j < j-1$

break

WC → 10000

no. of step reduced

internal sort algo stable

works good when array is partial sorted

hybrid sorting algo ← sort inbuilt

these all algorithms sorting doesn't perform well with the large data

(merge sort) (Quick) sort

↓

combined with insert

we try to insert the element in left of the correct position

CODE

```
static void insertionSort(int[] arr)

    for(int i=0; i<arr.length-1; i++
        for(int j=i+1; j>0; j--
            if(arr[j] > arr[j+1])
            swap(arr[j], arr[j+1])
            else
            break;
```

1 step
if i=0
if index 1 it
is sorted
if i=4
if index 2

2 step

```
if arr[j-1] > arr[j]
swap(arr, j, j-1)
else
break;
```

CYCLIC SORT —

* when given numbers from range 1, N use cyclic sort —

1, 2, 3 ... N

0 1 2 3 4
3, 5, 2, 1, 4

1 — N

for single pass sort we have to

After sort

0 1 2 3 4
1, 2, 3, 4, 5

element = index + 1
or
index = element - 1;

0 1 2 3 4

3, 5, 2, 1, 4

if 3 is == index + 1 ?

else

swap with

swap(a[i], arr, a[i], a[i+1])

element

-1

(2), 5, 3, 1, 4

↓

if swapped element is not in place check with corresponding index.

((5), 2, 3, 1, 4)

↓

if 5 == correct index (index + 1)

?

(4, 2, 3, 1, 5) → n steps

↑ ⇒ O(n)

is 4 correct index

(1, 2, 3, 4), 5 → correct

= (ans)

worst case

(5, 4, 3, 2, 1) (n-1) swaps

if (n-1) numbers are at correct index

it implies last element will be also at correct index

4 swaps made

+ 1 last check

n = (5) swaps.

N - 1 → N

O(N)

O(N) → Linear.

(5, 3, 4, 2, 6, 1)

CODE: CYCLE SORT.

void cyclicSort

start moving i put it in correct index only more i when i is in the correct index

(check swap more)

int i = 0

while (i < arr.length)

int correct = arr[i] - 1;

arr[i] != arr[correct] !=

arr[i] != correct)

swap(arr, i, correct

classmate

Date
Page

Questions → cyclic sort (heap array)
Amazon →

LC ~~236~~ 268 missing Number

arr num distinct from [0 to n] return the number that is missing

[1,2] [1,2,3]

(0,1)
for i=0
swapped (1,0)

i=1

arr[i] == 0 && i == n
ret i+1; [i]= (i+1)

(0,1)
if arr[i] == 0

(0,1)
i=0 we swap
(1,0)
↑
i=2

a[i]==0 && n==i
ret i+1.

Kunal approach
if number given 0 - N
then
total number N+1

important
how to start 2
you know.

Here in sorted missing
element == index
0, 1, 2, ... n

0 1 2 3
4, 0, 2, 1
can't put index out of bound

0, 4, 2, 1
ignore correct
0, 1, 2, 4 ignore

0 1 2 3
4, 0, 2, 1
index 4 doesn't exist
can't put 4, 1, 2, 0

0 1 2 3
[0, 1, 2, 2, 4]
arr ↑
3 ans

0 1 2 3 4

0, 1, 2, 4

CASE 2: when n is not in the array

if N=4

1, 0, 3, 2

0 1 2 3

(0, 1, 2, 3)

is index +1 = element

if i == N

return N;

cyclic sort

SC → O(1)

const

no new array

cyclic sort - question

find the missing number

int missing (int[] nums) {

for (int i=0; i<=nums.length ; i++

int correct = i;

sorting. if(nums[correct] == nums[i];

swap(arr , correct, i);

searching for (int i=0; i< num.length; i++

if nums[correct] != i

i

return i;

else

return

num.length

```
int i=0.
while ( i < nums.length)
    int correct = nums[i].
    if ( nums[i] < nums.length && nums[i] != nums[correct])
        swap ( nums, first, correct
    else
        i++;
searching for missing elem
for(int i=0; i< nums.length; i++)
    if (nums[i] != i)
        return i;
    else
}
return nums.length;
```

range 0 to n
index 0 → value

range 1 to n
index 0. value - 1

N = 8

4, 3, 2, 7, 8, 2, 3, 1 cycle sort
correct = num[i];
num[i] != nums[correct]

0 1 2 3 4 5 6 7
4, 3, 2, 7, 8, 2, 3, 1