Interview Problems on Arrays

Vinay Neekhra

Senior Instructor & Mentor

Reachable in Scaler Lounge

"Evolution forged the entirety of sentient life on this planet using only one tool... making mistakes" - Robert Ford

Q. Given an array of size N and Q queries of start (s) and end (e) index. for every query return the sum of all even indexed elements from s to e. Google Direct-i

↘ Range-sum

ex: A = [2, 3, 1, 6, 4, 5]
0 1 2 3 4 5

Query s	e	ans
1	3	1
2	5	5
0	4	7
3	3	0

A[2]
A[2] + A[4]
A[0] + A[2] + A[4]
~

Brute force: T.C. = O(N*Q) , SC = O(1)

obsⁿ: A = [2, 3, 1, 6, 4, 5]
0 1 2 3 4 5

PS = [2, 5, 6, 12, 16, 21]
0 1 2 3 4 5

PSe = [2, 2, 3, 3, 7, 7]

[s = 1, e = 3
psum[3]
- psum[1-1]]

$pS_e[i]$ = Sum of even indexed elements till i^{th} index

$$= \begin{cases} pS_e[i-1] & \text{if } i \text{ is odd} \\ pS_e[i-1] + A[i] & \text{if } i \text{ is even} \end{cases}$$

A = [2, 3, 1, 6, 4, 5]
(indices 0 1 2 3 4 5)

PS_o = [0, 3, 3, 9, 9, 14]
(indices 0 1 2 3 4 5)

$PS_o[4]$ = 3+6 = 9

T.C. = O(N+Q)
S.C. = O(N)

Q: Given an array of size N, count the number of special indices in the array.

- Directi
- Facebook
- Google
- JP morgan

Special index: an index is special if we remove that index-element, sum of all even indexed elements is equal to sum of all odd indexed elements.

ex. A = [4, 3, 2, 7, 6, -2] , ans = 2
(indices 0 1 2 3 4 5)

i	Array	Se	So	
0	[3, 2, 7, 6, -2] (0 1 2 3 4)	8	8	✓
1	[4, 2, 7, 6, -2] (0 1 2 3 4)	9	8	✗
2	[4, 3, 7, 6, -2]	9	9	✓
3	[4, 3, 2, 6, -2]	4	9	✗
4	[4, 3, 2, 7, -2]	4	10	✗
5	[4, 3, 2, 7, 6]	12	0	✗

Quiz A = [4, 1, 5, 3, 7, 10] (0 1 2 3 4 5)

A* = [4, 1, 3, 7, 10]

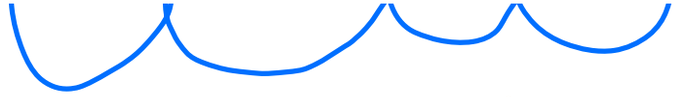
oddSum = 8

Quiz A = [2, 3, 1, 4, 0, -1, 2, -2, 10, 8] (0 1 2 3 4 5 6 7 8 9)

odd sum = 15 ✗

even sum = 8

Quiz A = [2, 3, 1, 4, 0, -1, 2, -2, 10, 8] (0 1 2 3 4 5 6 7 8 9)



Brute force : for every index i check it it's special.

A = [2, 3, 1, 4, 0, -1, 2, -2, 10, 8]
0 1 2 3 4 5 6 7 8 9

T.C. = $O(N^2)$
S.C. = $O(1)$

Obsⁿ:

$i = 3$

Not special

sum of odd indexed elements after removing index-3: = 15

sum of odd indexed elements from 0 to 2 ⇒ 3

+

sum of even indexed elements from 4 to 9 ⇒ 4, 6, 8

0 + 2 + 10 ⇒ 12

sum of even indexed elements after removing index-3: = 3 + 5 = 8

sum of even indexed elements from 0 to 2 ⇒ (2 + 1) = 3

sum of odd indexed elements from 4 to 9 (5, 7, 9)

$(-1) + (-2) + 8 = 5$

Afer removing index i

$$S_E = S_e[0, i-1] + S_o[i+1, N-1]$$

$$S_O = S_o[0, i-1] + S_e[i+1, N-1]$$

$PS_e \rightarrow$ even indexed sum

$PS_o \rightarrow$ odd indexd sum

for every index ($S_e == S_o$) $\Rightarrow$ O(1) time (using prefix sum)

$S_e[0, i-1] = PS_e[i-1]$ — time O(1)

$So[0, i-1] = PS_o[i-1]$ O(1)

$So[i+1, N-1] = PS_o[N-1] - PS_o[i]$ O(1)

$Se[i+1, N-1] = PS_e[N-1] - PS_e[i]$ O(1)

T.C. = O(N)
S.C. = O(2*N) = O(N)

Q Given an array of N integers, find the majority element.
Majority element is the element that occurs more than N/2 times.

ex. (I) A = [2, 1, 4] Ans = no majority element

(II) A = [3, 4, 3, 2, 4, 4, 4, 4] Ans = 4

$5 > \frac{8}{2}$

Quiz: A = [3, 4, 3, 6, 1, 3, 2, 5, 3, 3, 3] ans = 3

N	A	Ans

$N = 11$, count_3 = 6

0	[ ]	-1
1	[a]	a
2	[a, b]	-1

Quiz = [4, 6, 5, 3, 4, 5, 6, 4, 4, 4]

$N = 10$

count_4 = 5 ~~>~~ 10/2 ✗

count_6 = 2 ~~>~~ 10/2 ✗

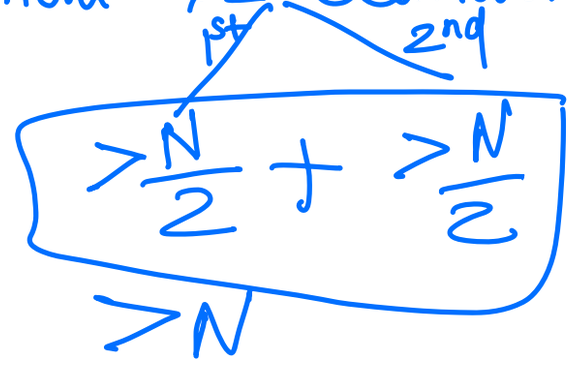
count_5 = 2 ~~>~~ 10/2 ✗

count_3 = 1 ~~>~~ 10/2 ✗

Obs^n: there can be only 1 majority element.

$\Rightarrow$ assume that there is more than one majority element, 2 elements ✗

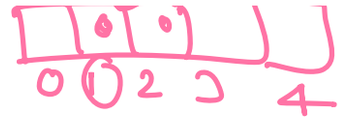
~~total elements~~ N



10:47

A = [2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4]

Brute force: = count[2] = 3
count (3)



for every element, count its appearence
if count > N/2 return elemnt.

Steps: (I) for every element from 0 to N-1
→ count how many times it appears in the array
→ if count > N/2 return

```
void findMajority (A) {
1.      for (int i=0; i<N; i++) {
2.          int count=0;
3.          for (int j=0; j<N; j++) {
4.              if (A[i] == A[j])
5.                  count++;
            }
6.          if (count > N/2)
7.              print (A[i]);
        }
```

T.C = O(N²)
S.C = O(1)

```
        }
    }
    print ("No majority found");
}
```

8.

A = [3,4,4]

A = [3, 4, 3, 2, 4, 4, 5, 6]
    0 1 2 3 4 5 6 7

T.C. = $O(N^2)$
S.C. = $O(1)$

N = 8

i

Count

A[i]

j

⇒ Your code should be simple enough for a 10 year old to understand

⇒ your function shouldn't be so long that you need to scroll.

(II)nd soln (Better than Brute force):

Obs^{n}: Sorting may help in counting

T.C. for sorting O(NlogN)

(I) After sorting, visit elements from left to right,

(a) If current element is equal to previous element, count++

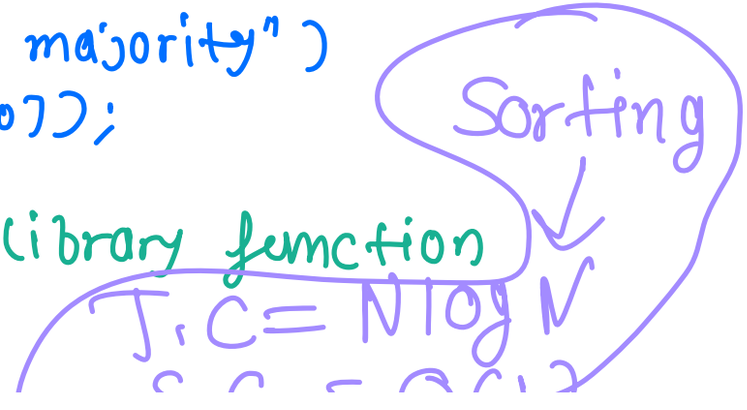
(b) If (count > N/2) return ans
else count = 1

```
void findMajority (A) {
    int N = A.size();

    if (N == 0)   print ("No majority")
    if (N == 1)   print (A[0]);

    sort (A);     // calling library function

    int count = 1;
```



```
int currElement = A[0]
for (i=1; i<N; i++){
        if (A[i] == currElement)
            count++;
        else{
            if(count > N/2) {
                Print (currElemant);
                return;
            }
            else{
                count=1;
                currElement= A[i];
            }
        }
}
if (count > N/2) Print (currElement)
else Print ("No majority found");
}
```

A = [3,4,4]
0 1 2

T.C. = O(NlogN)

S.C := O(1)

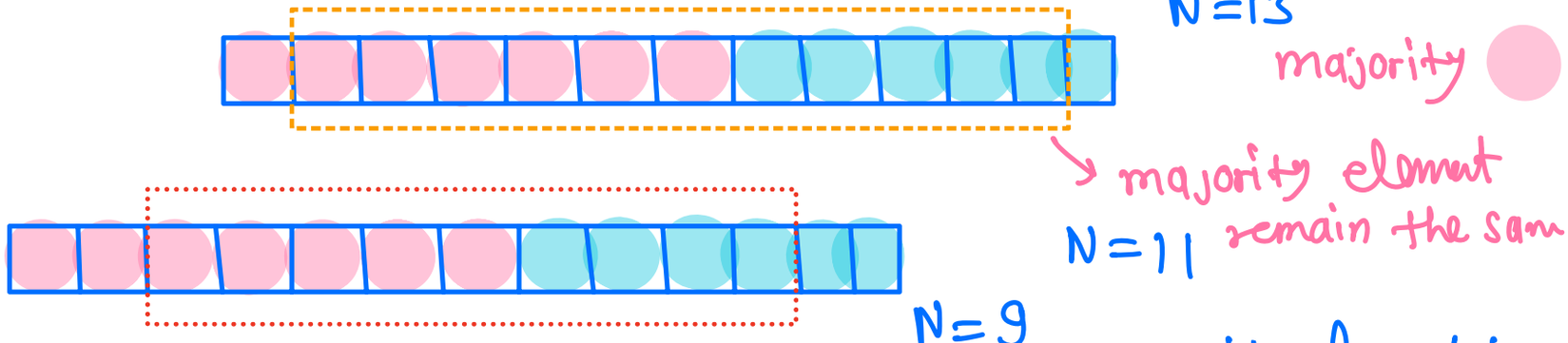
III Optimal soln:

Obsn (I) we can't sort the array

(II) Only one majority element

5. Moore's Voting algorithm

⇒ If we remove any two distinct elements the majority elements remain the same



election:

OP : ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

N = 17

PP: ✓ ✓ ✓

RP: ✓ ✓

GP: ✓ ✓ ✓

9 > 17/2

Remove	OP	PP	RP	GP	winner	N
initial stage	9	3	2	3	OP	17
1OP, 1PP	8	2	2	3	OP	15
1OP, 1RP	7	2	1	3	OP	13
1OP, 1GP	6	2	1	2	OP	11
1OP, 1PP	5	1	1	2	OP	9
1RP, 1GP	5	1	0	1	OP	7

Cond^n: can not remove same element in a round

Approach:

A = [3, 4, 3, 2, 4, 4, 8, 4]

0 1 2 3 4 5 6 7

traversing from i=0 to 9.

Count = 1 0 1 0 1 2 1 0 1

count how many times "survivor" occur in the original array

if count > N/2, ⇒ return num.

A = [3, 4, 3, 6, 1, 3, 2, 5, 3, 3, 3]
0 1 2 3 4 5 6 7 8 9 10

i = 0

major = 3 3 3 1 2 3 3 3

count = 1 0 1 0 1 0 1 0 1 2 3

last step: count, how many time 3 has come in the original array (should be > N/2)

T.C. = O(N), S.C. = O(1).

H.W. majority element > N/3