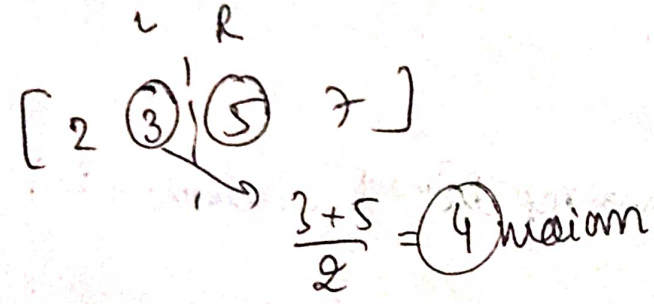
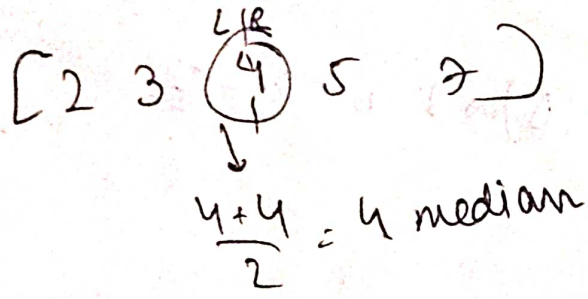
Median of 2 sorted arrays (VI)

Median → If we cut the sorted array to 2 halves of equal lengths, median = $\frac{\max(\text{lower half}) + \min(\text{upper half})}{2}$

i.e., the average of the 2 nos. immediately next to the cut.





N (length of array)	Index of L/R
1	0/0
2	0/1
3	1/1
4	1/2
5	2/2
6	2/3
7	3/3
8	3/4

∴ index of $L = \frac{N-1}{2}$ $R = \frac{N}{2}$

∴ median $\Rightarrow \frac{arr[L] + arr[R]}{2}$

$= \frac{arr\left[\frac{N-1}{2}\right] + arr\left[N/2\right]}{2}$ ← sorted array median.

Now for 2 sorted arrays:

N = 4

[6 9 13 18] → [# 6 # 9 # 13 # 18 #] → N' = 9

N = 5

[6 9 11 13 17] → [# 6 # 9 # 11 # 13 # 18 #] → N' = 11

#→ imaginary positions

$N' = 2N+1$ irrespective of value of N

∴ middle cut in such a case should always be placed on the nth position (0 based).

∴ $L = \frac{N-1}{2}$, $R = \frac{N}{2}$

∴ index $L = \frac{\text{cut position} - 1}{2}$

$R = \frac{\text{cutposition}}{2}$

∴ cutposition = N

We need to find a cut in this array which divides the 2 arrays into 2-halves such that

any no. in both arrays left halves combined $<=$ any no. in both arrays right halves combined.

Observations

① Now, N_1 = len of arr 1

N_2 = len of arr 2

with imaginary extra positions,

there are in total ⇒ $2N_1 + 1 + 2N_2 + 1$

⇒ $(2N_1 + 2N_2 + 2)$ positions

∴ There must be exactly $(N_1 + N_2)$ positions on each side of the cut.

② ∴ If we we cut at a position C2 in arr 2, (i)

then C1 in arr1 (cut position must be = $N1 + N2 - C2$)

③ When 2 cuts C1 & C2 are made, we have 2 sets of L & R :-

$L1 = arr1\left[\frac{\text{cut position} - 1}{2}\right]$ $R1 = arr1\left[\frac{\text{cut position}}{2}\right]$

$= arr1\left[\frac{C1 - 1}{2}\right]$ $R1 = arr1\left[\frac{C1}{2}\right]$

$L2 = arr2\left[\frac{c2-1}{2}\right]$ $\quad R2 = arr2\left[\frac{c2}{2}\right]$

L_1 or R_1

[1 2 (3)/(4)]

L_2 C_2 R_2

[1 (2)/(3) 4 5]

How to find if this is the required cut?

condition to be satisfied : L_1 and L_2 should be largest no. of arr1 & arr2 2's left halves respectively.

and R_1, R_2 should be smallest nos. in their right halves respectively.

∴ 2 conditions to be satisfied :

(i) condition ⇒ $L1 <= R1$ and $L2 <= R2$

(always satisfied ∵ array is sorted)

(ii) last final condition ⇒ $L1 <= R2$ and $L2 <= R1$ (*)

(criss-cross condition)

↙

use binary search to find out the perfect cut.

After we find both the c1 & c2, we also get L1, L2, R1, R2.

∴ median = $\frac{(\max(L1, L2) + \min(R1, R2))}{2}$ → 1

⇒

Binary search code :-

```
int n1 = arr1.size();
int n2 = arr2.size();
if(n1<n2) return fn(arr2, arr1);
int lo=0, hi = n2*2; // taking lo & hi w.r.t length of arr2 because we will be determining the cut portion in R2 primarily using these parameters.

while(lo <=hi)
{
  int c2 = (lo+hi)/2;
  int c1 = n1+n2 - c2; //eqn. stated earlier, check!

double L1 = (c1==0) ? INT_MIN : arr1[(c1-1)/2];
"  L2 = c2==0 ? INT_MIN : arr2[(c2-1)/2];
"  R1 = (c1==n1*2) ? INT_MAX : arr1[c1/2];
"  R2 = (c2==n2*2) ? INT_MAX : arr2[c2/2];

if (L1>R2) lo = c2+1; ←
else if (L2>R1) hi = c2-1;

} return ((max(L1,L2) + min(R1,R2))/2);
```

return -1;

}

BS conditions:

if (L1 > R2) → denotes that arr1's lower half is too big, it contains a big element that needs to be eliminated hence the left half and put into right half of arr1

∴ We need to shift cut1 in arr1 to ~~right~~ left while means to shift cut2 in arr2 to right.

else if (L2 > R1) → denotes arr2's lower half is too big

∴ We need to shift cut 2 in arr2 to left, while shifts cut1 in arr1 to right in turn.

otherwise → perfect cuts found c1 & c2

IMP

1) we can move either C1 or C2 using binary search. But it's less time taking to find the cut in the shorter array using binary search.

2) Edge case: when cut falls on 0th or 2(N*2)th or last position. This exceeds the boundary of the array. To handle this, we imagine arr1 and arr2 have 2 extra elements. Both arrays have [INT_MIN (extra at -1), 1, 2, 3, 4, INT_MAX (extra at Nth)]

If L falls out of bounds, L = INT_MIN

R " " " " , R = INT_MAX