Binary Search

Criteria

1. **Search Space:** Range in which we are performing our search
2. **Target:** Key of search
3. **Condition:** For answer; To eliminate left; To eliminate right

1.Problem Statement – Search in rotated array

Given a sorted array of integers **A** of size **N** and an integer **B**, where array **A** is rotated at **some pivot** unknown beforehand.
For example, the array [0, 1, 2, 4, 5, 6, 7] might become [4, 5, 6, 7, 0, 1, 2].
Your task is to search for the target value B in the array. If **found**, return its **index**; **otherwise**, return **-1**.
You can assume that **no duplicates** exist in the array.

NOTE: You are expected to solve this problem with a time complexity of **O(log(N))**.

Problem Constraints

1 <= N <= 1000000
1 <= A[i] <= 10^9
All elements in A are **Distinct**.

Output Format: Return index of B in array A, otherwise return **-1**

Example Input

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

B = 4

Example Output: 0

Example Explanation: Target 4 is found at index 0 in A.

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

Rotated Array

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

Brute force Idea

Do a linear search

TC = O(N)

SC =O(1)

Idea -2
Observation 1:

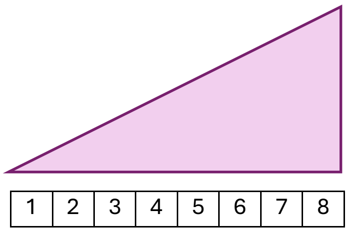
A=[4, 5, 6, 7, 8, 1, 2, 3]

In rotated sorted array, we have 2 sorted subarray

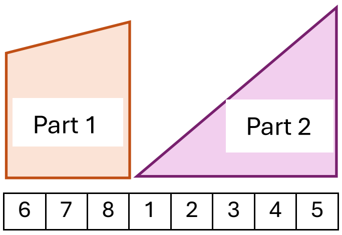
Observation 2:

1.How can we identify if A[i] is part of P1 or P2?

Compare with A[0], if (A[i] >=A[0]) then it belongs to P1 else It belongs to P2.



2.If any element = 8, if A[0] <8, then it will surely come in P1 and not in P2.



Binary Search

1. Search Space: The entire array
2. Target: Key
3. Condition: mid = (l+r)/2

Check if A[mid] and key are in same part → If, Yes: Apply binary search

➔ If No: Move to the actual part by changing mid value

Actual Code:

```
public class Solution {
    public int search(final int[] A, int B) {
        boolean targetinP1 = true;
        if(B < A[0]) targetinP1 = false;
        int l =0, r = A.length-1;
        while(l <=r){
            int mid = (l+r)/2;
  //identify part of mid
            boolean midinP1 = true;
            if(A[mid] < A[0]) midinP1 = false;
  //if both are in same part
            if(midinP1 == targetinP1){
                if(A[mid] == B) return mid;
                else if(A[mid] <B) l = mid+1;
                else r = mid-1;
            }
            else if(midinP1){                //if mid is in P1 but target is in P2 → move my mid to P2.
                l = mid+1;
            }else{
```

```
                    r = mid-1;
                }
            }
            return -1;
        }
}
```

Dry Run:

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

K = 11

targetinFirstHalf = True

L	R	Mid	A[mid]	A[mid] part	Both k and A[mid] -- Same part	Condition
0	7	(0+7)/2 = 3	A[3] =1	1<6; Part 2	No since 11>6	R = mid-1
0	2	(0+2)/2 =1	A[1] = 7	7>=6; Part 1	Yes	L = mid +1
2	2	(2+2)/2 = 2	A[2] = 8	8>6; Part 1	Yes	L = mid +1
3	2	(3+2)/2 = 2	--------------	-----------------	-----------------	**Return -1**

Stop: since l > r → return -1

k = 5

targetinFirstHalf = False as 5 < A[0]

L	R	Mid	A[mid]	A[mid] part	Same part	Condition
0	7	(0+7)/2 = 3	A[3] =1	1<6; Part 2	Yes	So basic condition will be binary search L = mid+1
4	7	(4+7)/2 =5	A[5] = 3	3<6; Part 2	Yes	Since 3< 5 eliminate left side; L = mid +1
6	7	(6+7)/2 = 6	A[6] = 4	4<6; Part 2	Yes	L = mid +1
7	7	(7+7)/2 = 7	--------------	-----------------	--------------	**Return -1**

2.Problem Statement – Square Root of Integer

Given an integer A. Compute and return the square root of A. If A is not a perfect square, return floor(sqrt(A)).

NOTE:
The value of A*A can cross the range of Integer.
Do not use the sqrt function from the standard library.
Users are expected to solve this in O(log(A)) time.

Output Format: Return floor(sqrt(A))

Example Input: 11

Example Output: 3

Example Explanation

When A = 11 , square root of A = 3.316. It is not a perfect square so we return the floor which is 3.

N	Floor(sqrt(N))
9	sqrt(3) = 3
12	sqrt(12) = 3
16	sqrt(16) = 4
24	sqrt(24) = 4

Brute Force(n =50)

i	i * i	ans
1	1*1 = 1	1
2	2*2 = 4	2
3	3*3 = 9 (9<50)	3 → potential answer
4	4*4 = 16	4
..	..	..
..	..	..
7	7*7 = 49	7 → ans
8	8*8 = 64	since (64 > 50) → stop

Brute force code:

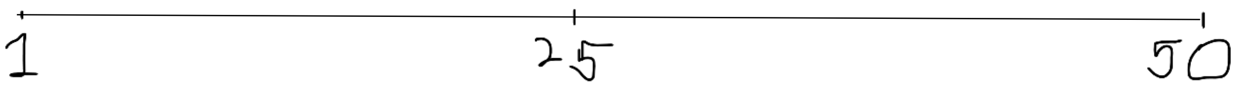
```
int Floorsqrt(int N){
        int ans = 1;
         for(int i =1; i*i <=N; i++){
              ans = i;
         }
         return ans;
}
```

TC = O(sqrt(N))

SC = O(1)

Binary Search:

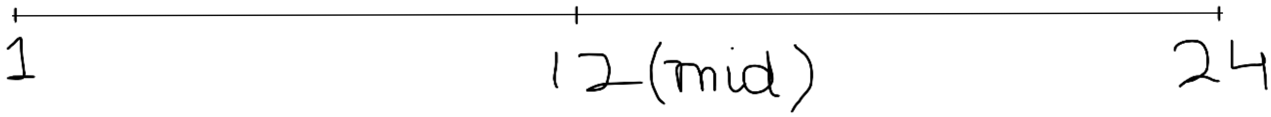
1. **Search Space:** smallest ans =1; largest (possible) ans = N
2. **Target:** floor(sqrt(N))
3. **Condition:** if(mid * mid <=N) L = mid +1; *//potential ans*
   else { r = mid -1}



for n =50

mid =25, 25*25 = 625

Since 625 > 50, discard the elements on the right.



mid = 12, 12*12 = 144

Since 144 > 50, discard the elements on the right.

mid = 5, 5 *5 = 25

Since, 25< 50, so ‘5’ could be my potential ans right now but will look for better answer so l = mid+1.

Actual Code

```
public class Solution {
    public int sqrt(int A) {
        long l =1;
        long r = A, ans =0;
        while(l <=r){
            long mid = l+(r-l)/2;
            if(mid * mid <= A){ //potential ans
                ans = mid;
                l = mid+1;
            }
            else{
                r = mid-1;
            }
        }
        return (int)ans;
    }
}
```

Dry Run

n = 50, ans = ~~6~~ 7

L	R	Mid	mid*mid	where next?
1	50	25	25*25 = 625	625 >50; r = mid-1
1	24	12	12*12 = 144	144 > 50; r = mid-1
1	11	6	6*6 = 36	36<50; l = mid +1; one of my potential ans
7	11	9	9*9 = 81	81>50; r = mid-1
7	8	7	7*7 = 49	49<=50; next potential ans
8	8	8	8*8 = 64	64>50; r = mid-1
8	7	---------------	----------------------	**return ans =7**

3.Problem Statement – Median of two sorted array

Given two sorted arrays A and B of size M and N respectively, return the median of the two sorted arrays. Round of the value to the floor integer [2.6=2, 2.2=2]

Output Format: Return an integer.

Example Input:
A = [1, 3]
B = [2]

Example Output: 3

Median → is middle element in a sorted array

A =

10	20	30	40	45	46	50	60	70	80
0	1	2	3	4	5	6	7	8	9

Median = (45+46)/2

Median of two sorted arrays→ Odd elements

A =

1	4	5

C=

1	2	3	4	5

B =

2	3

Brute Force

1. Merge into a single sorted array
2. Find the median array
3. return median

TC = O(N+M)

SC = O(N+M)

Idea -2 : Binary Search

1. **Search Space:** Apply binary search in a combined array – 2 arrays A & B
2. **Target:** Median of A & B
3. **Condition:** if(L1 <= R2 && L2 <= R1) → ans
   else if (L2 > R1) l = mid +1 → eliminate left
   else r = mid -1 → eliminate right

A =

10	30	45	46	60	80

B =

20	40	50	70

C =

10	20	30	40	45	46	50	60	70	80

Observation1: Both parts have some elements from Array A & some elements from Array B

Observation2: All elements in Part1 (P1) < All elements in Part2(P2)

TRIAL & ERROR

A =

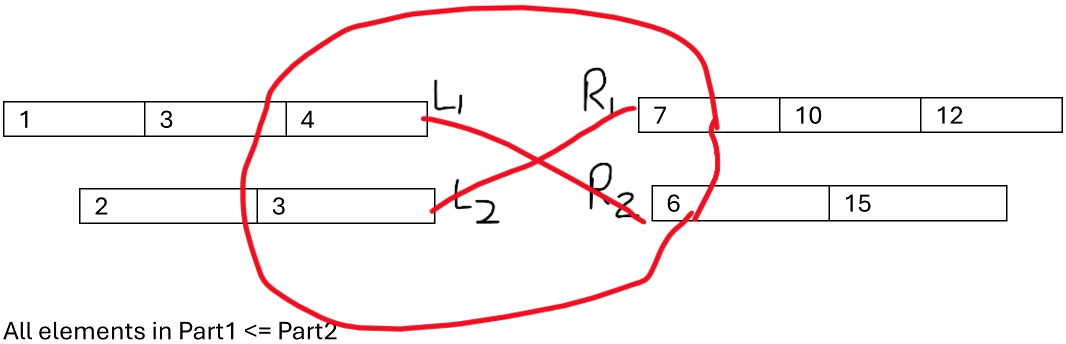
1	3	4	7	10	12

B =

2	3	6	15

Try 1:

Let me pick 3 elements from A in Part1



All elements in Part1 <= Part2

Here L1 <= R2 and L2 <= R1

Part1 :

1	2	3	3	4

Part2:

6	7	10	12	15

Median = (Max(L1, L2) + Min(R1, R2))/2 in case of even elements.

Total elements in Part1 = (N+M)/2

Dry Run:

A =

1	2	3	4	9	11
0	1	2	3	4	5

B =

7	12	14	15
0	1	2	3

Total elements in Part1 = (N+M)/2 = (6+4)/2 = 5

L	**R**	**mid**	**L1**	**R1**	**L2**	**R2**	**is valid?**
0	5	0+5/2 = 2 (this is the total #of elements taken in P1)	A[1] = 2	3	14	15	Since L2>R1; L = mid1+1
3	5	3+5/2 = 4	4	9 since L2 < R1 & L1<R2 so this is a good split	7	12	4 <=12; 7<=9

Elements taken from A in Part1 = 2 (mid element)

L1 = A[mid1-1]
R1 = A[mid1]
L2 = B[mid2-1]
R2 = B[mid2]

mid2 = total elements in PartA – mid → 5 -2 = 3

ans = (Max(L1, L2) + Min(R1, R2))/2

Max(4,7) + Min(9, 12)/2 = 7+9 /2 = 8

If L2 < R1 → select more elements from A in Part1

Note: Time Complexity depends on length of array A. So among both array, whosoever would have less length, I could take lesser length array as array A.

How to handle odd length array

Total elements in P1 = (n+m+1)/2 → Take one extra element in Part 1

Median = Max{L1, L2)

Edge Cases

if (mid1 ==0) → L1 will go out of bound L1 = - infinity	if(mid1 == n) → R1 will go out of bound R1 = + infinity
if(mid2 ==0) L2 = -infinity	if(mid2 == m) R2 = + infinity

Actual Code

```
public class Solution {
  public int solve(int[] A, int[] B) {
    //always consider smaller array as A
    if(A.length > B.length){
      int [] temp = A;
        A = B;
        B = temp;
    }
    //initialize all values
    int n = A.length, m =B.length;
    int totalinPartA = (m+n+1)/2;
    int l =0, r = n;
    double median =0;
    //Binary Search
    while(l <= r){
      int mid1 = (l+r)/2;
      int mid2 = totalinPartA - mid1;
      //initialize L1, L2, R1,R2

      int L1 = (mid1 >0) ? A[mid1 -1]:Integer.MIN_VALUE;
      int R1 = (mid1 <n) ? A[mid1]: Integer.MAX_VALUE;
      int L2 = (mid2 > 0)? B[mid2 -1]:Integer.MIN_VALUE;
      int R2 = (mid2 < m)? B[mid2]:Integer.MAX_VALUE;

      //condition for ans
      if(L1 <= R2 && L2 <= R1){
        if((n+m) % 2 ==0){
          median = (Math.max(L1, L2) + Math.min(R1, R2))/2;
          return (int) median;
        }else{
          median = Math.max(L1, L2);
          return (int) median;
```

```
            }
          }
          else if (L2 > R1){
            l = mid1 +1;
          }else{
            r = mid1 -1;
          }
        }
        return (int) median;
    }
}
```

Problem Statement 4 – Matrix Median

Problem Description

Given a matrix of integers **A** of size N x M in which each row is sorted.

Find and return the overall median of matrix A.

NOTE: No extra memory is allowed.

NOTE: Rows are numbered from top to bottom and columns are numbered from left to right.

Problem Constraints

1 <= N, M <= 10^5

1 <= N*M <= 10^6

1 <= A[i] <= 10^9

N*M is odd

Input Format

The first and only argument given is the integer matrix A.

Output Format

Return the overall median of matrix A.

Example Input

Input 1:

```
A = [   [1, 3, 5],
        [2, 6, 9],
        [3, 6, 9]   ]
```

Input 2:

```
A = [   [5, 17, 100]   ]
```

Example Output

Output 1:

5

Output 2:

17

Example Explanation

Explanation 1:

A = [1, 2, 3, 3, 5, 6, 6, 9, 9]

Median is 5. So, we return 5.

Explanation 2:

Median is 17.

Actual Code

```
public class Solution {
    public int findMedian(ArrayList<ArrayList<Integer>> A) {
        int l =Integer.MAX_VALUE, r = Integer.MIN_VALUE, ans =0;
        int n = A.size();
        int m = A.get(0).size();
        //int count =0;
        for(int i = 0; i<n; i++){
            l = Math.min(l, A.get(i).get(0));
            r = Math.max(r, A.get(i).get(m-1));
        }
        int req_cnt = (n*m +1)/2; // required count of numbers less than the median // median would
be req_cnt+1th number
        while(l<r){
            int mid = l+(r-l)/2;
            int count = 0;
            for (int i = 0; i<n; i++){
                count += countlessthanorequalto(A.get(i), mid);
            }
            if(count < req_cnt){
                l = mid+1; // Median is larger
            }else{
                r = mid; // Median is smaller or equal
            }
        }
        return l;
    }
    private int countlessthanorequalto(ArrayList<Integer> row, int mid){
        int l =0, r = row.size() -1;
        while(l<=r){
            int mid2 = l+(r-l)/2;
            if(row.get(mid2) <= mid){
                l = mid2 +1;
            }else{
                r = mid2 -1;
            }
        }
        return l;
    }
}
```

Problem Statement 5 - Ath Magical Number

Problem Description

You are given three positive integers, **A**, **B,** and **C**.

Any positive integer is magical if divisible by either **B** or **C**.

Return the **A**th smallest magical number. Since the answer may be very large, return modulo **10^9 + 7**.

Note: Ensure to prevent **integer overflow** while calculating.

Problem Constraints

1 <= A <= 10^9

2 <= B, C <= 40000

Input Format

The first argument given is an integer **A**.

The second argument given is an integer **B**.

The third argument given is an integer **C**.

Output Format

Return the **A**th smallest magical number. Since the answer may be very large, return modulo **10^9 + 7**.

Example Input

Input 1:

A = 1

B = 2

C = 3

Input 2:

A = 4

B = 2

C = 3

Example Output

Output 1: 2

Output 2: 6

Example Explanation

Explanation 1:

1st magical number is 2.

Explanation 2:

First four magical numbers are 2, 3, 4, 6 so the 4th magical number is 6.

Actual Code

```
public class Solution {
   public int solve(int A, int B, int C) {
      long l = 1, ans =0;
      long r = (long) A * Math.min(B, C);
      long mod = 1000000007L;
```

```
    long gcd = findgcd(B, C);
    long lcm = (long) B * (long)C/gcd;
    while(l <= r){
      long mid = l + (r-l)/2;
      long count = isPossible(mid, B, C, lcm);
      if(count >= A){
        ans = mid;
        r = mid-1;
      }else{
        l = mid+1;
      }
    }
    return (int) (ans % mod);
  }
    private long isPossible(long mid, long B, long C, long lcm){    //count magical numbers
      return mid/B + mid/C - mid/lcm;
    }

    private long findgcd(long a, long b){
      if (b==0) return a;
      return findgcd(b, a % b);
    }
}
```

T.C. :- $O(\log(\min(B,C) \times A))$

S.C. :- $O(1)$

Problem Statement 6 - Find Smallest Again

Problem Description

Given an integer array **A** of size **N**.

If we store the sum of each triplet of the array **A** in a new list, then find the **Bth** smallest element among the list.

NOTE: A triplet consists of three elements from the array. Let's say if **A[i], A[j], A[k]** are the elements of the triplet then **i < j < k**.

Problem Constraints

3 <= N <= 500

1 <= A[i] <= 10^8

1 <= B <= (N*(N-1)*(N-2))/6

Input Format

The first argument is an integer array A.

The second argument is an integer B.

Output Format

Return an integer denoting the B^{th} element of the list.

Example Input

Input 1:

A = [2, 4, 3, 2]

B = 3

Input 2:

A = [1, 5, 7, 3, 2]

B = 9

Example Output

Output 1:

9

Output 2:

14

Example Explanation

Explanation 1:

All the triplets of the array A are:

(2, 4, 3) = 9

(2, 4, 2) = 8

(2, 3, 2) = 7

(4, 3, 2) = 9

[7,8,9,9]

So the 3^{rd} smallest element is 9.

Actual Code

```
import java.util.*;

public class Solution {
    public int solve(ArrayList<Integer> A, int B) {
        Collections.sort(A); // Step 1: Sort the array
        int n = A.size();
        long low = (long) A.get(0) + A.get(1) + A.get(2);
        long high = (long) A.get(n - 1) + A.get(n - 2) + A.get(n - 3);
        int ans = 0;

        while (low <= high) {
```

```
        long mid = low + (high - low) / 2;
        long count = countTriplets(A, mid); // Step 3: Count triplets

        if (count >= B) {
          ans = (int) mid; // Update answer
          high = mid - 1; // Search for smaller values
        } else {
          low = mid + 1; // Search for larger values
        }
      }

      return ans;
    }

    private long countTriplets(ArrayList<Integer> A, long mid) {
      int n = A.size();
      long count = 0;

      for (int i = 0; i < n - 2; i++) {
        int j = i + 1, k = n - 1;

        while (j < k) {
          long sum = (long) A.get(i) + A.get(j) + A.get(k);
          if (sum <= mid) {
            count += (k - j); // All pairs from j to k are valid
            j++;
          } else {
            k--; // Decrease the upper pointer
          }
        }
      }

      return count;
    }
  }
```

Problem Statement 7 - ADD OR NOT - Unsolved

Problem Description

Given an array of integers **A** of size **N** and an integer **B**.

In a single operation, any one element of the array can be increased by 1. You are allowed to do at most **B** such operations.

Find the number with the **maximum** number of occurrences and return an array **C** of size 2, where **C[0]** is the number of occurrences, and **C[1]** is the number with maximum occurrence. If there are several such numbers, your task is to find the **minimum** one.

Problem Constraints

$1 <= N <= 10^5$

$-10^9 <= A[i] <= 10^9$

$0 <= B <= 10^9$

Input Format

The first argument given is the integer array A.

The second argument given is the integer B.

Output Format

Return an array C of size 2, where C[0] is number of occurence and C[1] is the number with maximum occurence.

Example Input

Input 1:

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

B = 3

Input 2:

A = [5, 5, 5]

B = 3

Example Output

Output 1: [4, 2]

Output 2: [3, 5]

Example Explanation

Explanation 1:

Apply operations on A[2] and A[4]

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

Maximum occurence = 4

Minimum value of element with maximum occurence = 2

Explanation 2:

A = [5, 5, 5]

Maximum occurence = 3

Minimum value of element with maximum occurence = 5

Actual Code

```
import java.util.*;

public class Solution {
    public ArrayList<Integer> solve(ArrayList<Integer> A, int B) {
        Collections.sort(A); // Sort the array
        int n = A.size();
        int maxOccurrences = 1;
        int minValue = A.get(0);

        int left = 0, right = 0, totalCost = 0;
```

```java
        // Sliding window to calculate max occurrences
        while (right < n) {
            // Calculate the cost of making A[left...right] equal to A[right]
            totalCost += (right - left) * (A.get(right) - A.get(right - 1));

            // If cost exceeds B, move the left pointer
            while (totalCost > B && left < right) {
                totalCost -= (A.get(right) - A.get(left));
                left++;
            }

            // Update the maximum occurrences and the minimum value
            int currentOccurrences = right - left + 1;
            if (currentOccurrences > maxOccurrences) {
                maxOccurrences = currentOccurrences;
                minValue = A.get(right);
            } else if (currentOccurrences == maxOccurrences) {
                minValue = Math.min(minValue, A.get(right));
            }

            right++;
        }

        ArrayList<Integer> result = new ArrayList<>();
        result.add(maxOccurrences);
        result.add(minValue);
        return result;
    }
}
```