

1. Majority Element - I

1. Problem Discussion :-

You will be provided with an array of size 'n' and n elements. You have to find the majority element in the array if it exists. Otherwise print 'No Majority Element Exists'. 'Majority Element' is an element that occurs more than $n/2$ times in the array. We have to solve this problem in linear time with $O(1)$ space complexity.

For example: Given array is as follows:



And the size of the array is 10. So the majority element is the element that occurs more than 5 (i.e. $10/2$) times.



So the answer is 1 here. Because it occurs 6 times. So I think that the question is clear, now let's move to approach.

2. Approach :-

Naive approach :-

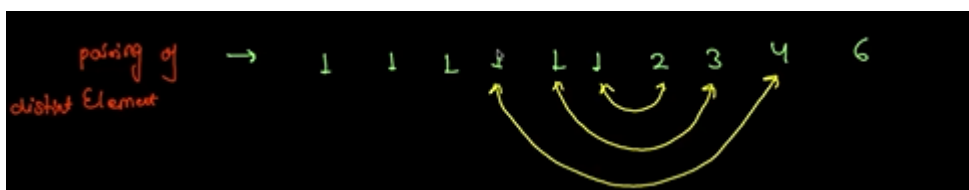
1) It says to traverse each element and count its frequency. It seems simple but let's discuss the cons of this approach. First of all it takes $O(n^2)$ time and then if the element occurs more than one time then we calculate its frequency every time it occurs which is a waste of time.

2) Use a **FREQUENCY HashMap** where the key is the element and value is the frequency then we update the frequency of every element. But this approach also **use $O(d)$ space** where 'd' is the number of distinct elements, which is not allowed.

So let's discuss an optimised approach.

Optimised approach :-

We call this approach 'pairing of distinct elements' because we will be pairing distinct elements here.



In the above image you can see that the '1' is paired with '4' then next 1 is paired with '3' then next '1' is paired with '4'.



So the last element remaining unpaired can be a valid candidate to be a majority element because it remains unpaired so there can be possibility that the other remaining elements are also equal to that element so we can not pair the same element. because all the distinct elements are paired only same elements will be unpaired. So we check if the last remaining element is a valid candidate to be

a majority element or not; it means we check it's frequency if it is greater than $n/2$ then `isValidCandidate()` will return true else it will return false. So here we will be keeping two variables 'val' and 'count' (i.e. The frequency of the element). 'val' is initially equal to `arr[0]` where 'arr' is our array and `count=1`.

Value = 1
Count: 1

And we also keep a variable 'i' which is 1 initially (i.e. $i=1$), which points to `arr[1]` initially.

Index: 0 1 2 3 4 5 6 7 8 9
Value: 1 1 3 4 5 6 6 7 9 9
↑
i

Now we check if the next element that is at `arr[i+1]` is equal to 'val' or not if it is equal to 'val' then increase its frequency that is stored in count to `count+1`.

Index: 0 1 2 3 4 5 6 7 8 9
Value: 1 1 3 4 5 6 6 7 9 9
↑
i

Value = 1
Count: 2

if (val == arr[i])?
count++;

Here `arr[i]==value`, so we increase the count to 2. If it is not equal to 'val' then pair it with that element and decrease its frequency that is `count=count-1`. In both cases we will increase the value of 'i' to `i++` till 'i' is less than the length of the array.

Index: 0 1 2 3 4 5 6 7 8 9
Value: 1 1 3 4 5 6 6 7 9 9
↑
i

Value = 1
Count: 2

if (val == arr[i])?
count++;

After applying this approach on the whole array we get this array.

Index: 0 1 2 3 4 5 6 7 8 9
Value: 1 1 3 4 5 6 6 7 9 9
↑
i

So now the remaining element that is unpaired is 9. Now we will check if 9 is a valid candidate or not. So for that purpose we will write a method `isValidCandidate()` that returns true if the frequency of the number is more than $n/2$ and returns false otherwise. Since the frequency of 9 is 2 which is less than $n/2$ ($10/2 = 5$), so it is not a valid candidate so we print 'No majority Element Exist', If the frequency of 9 were more than 5 then we would print 9.

3. CODE -:

```
// ~~~~~User Section~~~~~

public static int validCandidate(int[] arr) {
    int val = arr[0];
    int count = 1;
    for(int i = 1; i < arr.length; i++) {
        if(val == arr[i]) {
            count++;
        } else {
            count--;
        }
        if(count == 0) {
            val = arr[i];
            count = 1;
        }
    }
    return val;
}

public static void printMajorityElement(int[] arr) {
    // write your code here
    int val = validCandidate(arr);
    int count = 0;
    for(int ele : arr) {
        if(ele == val)
            count++;
    }

    if(count > arr.length / 2) {
        System.out.println(val);
    } else {
        System.out.println("No Majority Element exist");
    }
}
```

4. Code Explanation -:

1.) We first of all make two variables 'val' and 'count'.

```
int val = arr[0];
int count = 1;
```

2.) This is the code for the pairing part. In this part of the code we pair distinct elements.

```
for(int i = 1; i < arr.length; i++) {
    if(val == arr[i]) {
        count++;
    } else {
        count--;
    }
    if(count == 0) {
        val = arr[i];
        count = 1;
    }
}
```

Then here's the complete code for validCandidate() method that returns a valid candidate.

```

public static int validCandidate(int[] arr) {
    int val = arr[0];
    int count = 1;
    for(int i = 1; i < arr.length; i++) {
        if(val == arr[i]) {
            count++;
        } else {
            count--;
        }
        if(count == 0) {
            val = arr[i];
            count = 1;
        }
    }
    return val;
}

```

Here is the code for printmajorityelement() that first check if the frequency of the valid candidate is more than $n/2$ or not. If it is then it print it also print 'No majority element exists'.

```

public static void printMajorityElement(int[] arr) {
    // write your code here
    int val = validCandidate(arr);
    int count = 0;
    for(int ele : arr) {
        if(ele == val)
            count++;
    }

    if(count > arr.length / 2) {
        System.out.println(val);
    } else {
        System.out.println("No Majority Element exist");
    }
}

```

5. Analysis :-

Time Complexity: $O(n)$

Here the time complexity is $O(n)$ because we traverse the array only once.

Space complexity : Constant

Here No extra space is used so space complexity is constant.

That was easy. Wasn't it? Our desire to make you learn will remain unsatisfactory if you still have doubts. We strongly recommend you to watch our video lecture on Majority Element 1 for clearing any type of doubts.

Suggestions and feedback are always welcomed. You can contact us via our website. All the best for a bright future! Happy Coding!

Author: Parth Mathur

2. Majority Elements - I I

1. Problem Discussion:

You will be provided with an array 'arr' of size 'n' and you have to return those elements whose frequency is more than $n/3$ times in an arraylist. You have to solve this problem in $O(1)$ space Complexity. For e.g.

1	2	1	3	2	2	1
0	1	2	3	4	5	6

Majority Element=[1, 2]

2. Approach:

Let's discuss the approach As this problem is an extended part of 'Majority Element - I' So we will try to relate both the problems . So in 'Majority Element - I' there could be only one element in the array that occurs more than $n/2$ times ,it means in that problem only one majority element was possible .So we have to search for only one element. But here the number of majority elements can be 0,1 or 2. And also in the 'pairing of distinct elements' we were pairing only two elements but now there is a slight difference. Here we will be pairing three distinct elements. The last remaining elements that are unpaired will be those elements that are either same or can't from a pair of three i.e. number of remaining elements are 1 or 2.



In the above image you can see that we pair 1,2,3 i.e.three distinct elements. Now let's start talking about the code part. Here we will be taking four variables 'val1'=arr[0] (initially) , 'count1'=1(holds the initial frequency of val1) , 'val2'=any random value initially or say arr[0] and 'count2' =0(holds initial frequency of val2). And we also have an 'i' variable that is i=1 (initially). Now we will check if arr[i]==val1 then increase 'count1' to count1 + 1 i.e. increase its frequency , else if arr[i]==val2 to then increase 'count2' to count2 + 1, else if arr[i] != val1 && arr[i] !=val2 then check, if the count1==0 if it is then it means val1 does not hold any number then initialize it with arr[i] and count1 =1,else check if count2==0 then it means val2 does not hold any number so initialize val2 with arr[i] and count2 =1 .Else if both the count1 and count2 are not equal zero then it means val1 != val2 != arr[i] so pair all the three i.e. count1=count1-1 & count2=count2 -1. Let's discuss the approach with an example. Suppose this is the array initially.

1	2	1	3	2	2	1
0	1	2	3	4	5	6

↑
i

'val1' is arr[0] i.e. 1 and count1=1 && val2 is also equal to arr[0] or we can assign any random value initially and count2=0 and 'i'=1.

```

val1 = 1
count1 = 1

val2 = 1
count2 = 0

```

Now we check the condition if arr[1]==1 else if arr[1]== 1.

```

if(arr[i] == val1) {
    count1++;
} else if(arr[i] == val2) {
    count2++;
}

```

So here arr[1] i.e. 2 != 1 so we go to the else part and check if count1==0 or count2==0 .

```

else {
    if(count1 == 0) {
        val1 = arr[i];
        count1 = 1;
    } else if(count2 == 0) {
        val2 = arr[i];
        count2 = 1;
    } else {
        count1--; count2--;
    }
}

```

So here count2 is 0 ,so we initialize the val2 = 2 and count2= 1.and 'i' to 'i+1'.

```

val2 = 2
count2 = 1

```

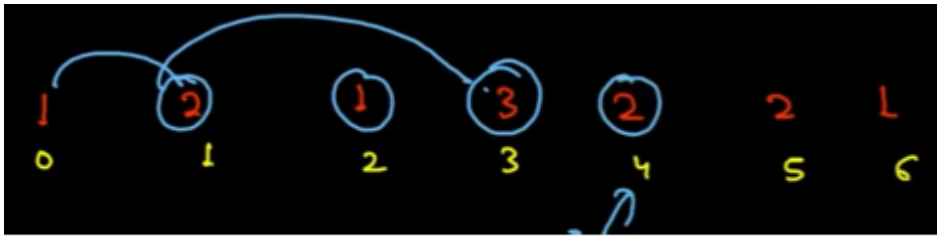
Now again we will compare arr[2] with val1 and val2. So here arr[2] i.e. 1 == val1 so we increase the count1 to count1+1, So count1 is 2 now and 'i' to 'i+1' , 'i' is 3 now.

```

val1 = 1
count1 = 2

```

Again we check the same conditions here for arr[3] , Here arr[3] != val1 & arr[3] != 2 & count1 != 0 & count2 != 0 so we pair the three elements i.e. val1 , val2 and arr[3] So we decrease count1 to count1 -1 and count2 to count2 - 1. So we pair 1, 2 and 3 here.



At last we get two values $val1=1$ & $val2=2$, these two numbers are valid candidate to be a majority element. So we check the frequency of both the numbers if it is greater than $n/3$ or not if it is then we add the elements to the resultant arraylist. Then we return the arraylist.

3. Code

```

// ~~~~~User Section~~~~~
public static boolean isGreaterThanNb3(int[] arr, int val) {
    int count = 0;

    for(int ele : arr) {
        if(ele == val)
            count++;
    }

    return count > arr.length / 3;
}

public static ArrayList<Integer> majorityElement2(int[] arr) {
    if(arr.length == 0) return new ArrayList<>();
    int count1 = 1;
    int val1 = arr[0];

    int count2 = 0;
    int val2 = arr[0];

    int i = 1;
    while(i < arr.length) {
        if(arr[i] == val1) {
            count1++;
        } else if(arr[i] == val2) {
            count2++;
        } else {
            if(count1 == 0) {
                val1 = arr[i];
                count1 = 1;
            } else if(count2 == 0) {
                val2 = arr[i];
                count2 = 1;
            } else {
                count1--;
                count2--;
            }
        }
        i++;
    }

    ArrayList<Integer> res = new ArrayList<>();
    if(isGreaterThanNb3(arr, val1) == true)
        res.add(val1);

    if(val1 != val2 && isGreaterThanNb3(arr, val2) == true)
        res.add(val2);

    return res;
}

```

4. Code Explanation:

Here is the code for the method that checks if the frequency of the valid candidate is more than $n/3$ or not.


```
// MAJORITY ELEMENT - USER SOLUTION
public static boolean isGreaterThanNb3(int[] arr, int val) {
    int count = 0;

    for(int ele : arr) {
        if(ele == val)
            count++;
    }

    return count > arr.length / 3;
}
```

In the below code we check if the arr.length is 0 or not if it is then there would be no majority element so we return an empty arraylist. Else we initialize 'val1', 'val2', 'count1' and 'count2'.

```
if(arr.length == 0) return new ArrayList<>();
int count1 = 1;
int val1 = arr[0];

int count2 = 0;
int val2 = arr[0];
```

Below is the code for pairing three distinct elements.

```
int i = 1;
while(i < arr.length) {
    if(arr[i] == val1) {
        count1++;
    } else if(arr[i] == val2) {
        count2++;
    } else {
        if(count1 == 0) {
            val1 = arr[i];
            count1 = 1;
        } else if(count2 == 0) {
            val2 = arr[i];
            count2 = 1;
        } else {
            count1--;
            count2--;
        }
    }
    i++;
}
```

5. Analysis:

Time Complexity: $O(n)$

Here we traverse the whole array only once so it takes $O(n)$ time.

Space complexity: Constant

No extra space is used so space complexity is constant.

Author: Damini Patel

3. Majority Element General

1. Problem Discussion :

You are given an array of size 'N' and an element K. Your task is to find all elements that appear more than N/K times in the array and return these elements in an ArrayList in sorted order.

Example) For $N = 8$, $arr[] = \{3, 1, 2, 2, 1, 2, 3, 3\}$, $K = 4$

Explanation:

If we see the frequency of all elements in array:

Element	Frequency
1	2
2	3
3	3

We can see the only frequency of 2 and 3 is more than $N/K(2)$.

So, the output is [2,3].

2. Approach :

Our approach will be based on the basis of previous problems “Majority Element I” and “Majority Element II”.

Original Array

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

Approach: We just need to somehow map the frequency of elements. If we know the frequency of elements we could see which elements have frequency greater than N/K times and then add them to our result arraylist.

But unlike in previous problems if we try to use k distinct values and count it would be inefficient. So, to map the frequency of elements we would use a hashmap.

For creating hashmap we will use element as key in hashmap and frequency as value in hashmap. For filling our frequency hashmap we will traverse over our array and check if the Hashmap already contains the traversed element or not. If it is present, then increase its frequency by using `get()` and `put()` function in Hashmap and if it occurs first time then we put this element with 1 frequency.

Element(Key)	Frequency(Value)
1	2
2	3
3	3

Once we have a frequency hashmap we just need to traverse the hashmap and add elements which have frequency greater than N/K times to our result arraylist.

N = 8, K = 4

N/K = 2

Arraylist = [3,2]

We need to sort our arraylist because it is not necessary that our hashmap is filled in sorted because of which the arraylist will be filled in unsorted order and we are told to return a sorted arraylist.

Arraylist = [2,3]

3. Code :

```
//User Section
public static ArrayList<Integer> majorityElement(int[] arr, int k) {
    HashMap<Integer, Integer> map = new HashMap<>();
    int n = arr.length;
    for(int i = 0; i < n; i++) {
        int key = arr[i];
        if(map.containsKey(key) == true) {
            map.put(key, map.get(key) + 1);
        } else {
            map.put(key, 1);
        }
    }
    ArrayList<Integer> res = new ArrayList<>();
    for(int key : map.keySet()) {
        if(map.get(key) > n / k)
            res.add(key);
    }
    Collections.sort(res);
    return res;
}
```

4. Code Discussion :

1. First we have to create our frequency hashmap.

```
HashMap<Integer, Integer> map = new HashMap<>();
int n = arr.length;
for(int i = 0; i < n; i++) {
    int key = arr[i];
    if(map.containsKey(key) == true) {
        map.put(key, map.get(key) + 1);
    } else {
        map.put(key, 1);
    }
}
```

2. Now, we just need to add all elements whose frequency in the array is more than N/K times.

```
ArrayList<Integer> res = new ArrayList<>();
for(int key : map.keySet()) {
    if(map.get(key) > n / k)
        res.add(key);
}
```

3. Here, we sorted our arraylist because we were told to return a sorted arraylist.

```
Collections.sort(res);
```

4. Now, we just have to return the result arraylist.

```
return res;
```

5. Analysis :

Time Complexity : $O(n \log(n))$

Overall time complexity of the function will be $O(n \log(n))$ because we used sorting for our arraylist.

Space Complexity : $O(n)$

Since, we created a frequency hashmap to store the frequency of all array elements, the space complexity becomes $O(n)$.

Author: Kunal Singh Bhandari

4.Next Greater Element I I I

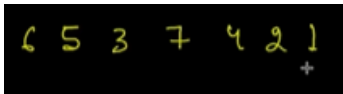
1. Problem Discussion:

You will be given a positive number 'n' in the form of String.You have to find the smallest number which has exactly the same digits existing in the number 'n' and is greater in value than 'n'. Return -1 If no such positive number exists. For example: Input : s= "123" Output: "132" I think the problem is clear now,Let's move to approach.....

2. Approach:

We have to make a number which is **just greater(i.e. Smallest from all the greater numbers)** than the given number using the same digits. So let's see how we can solve this.

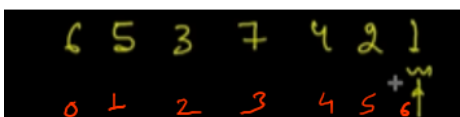
Let's take an example ,suppose this is the given num.



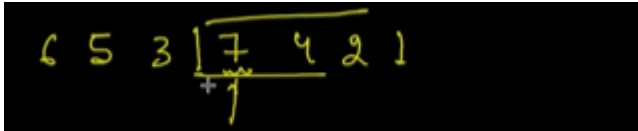
Step 1 Find first Dip: We have to find out a just greater number than '6537421' which is '6541237' . So here we will be making changes in the digits having the least weightage because if we work with the digits having the highest weightage , so it might form a very big number but we have to find a smaller number in all numbers greater than the given number.

So we will be working with digits having the least weightage. So one thing is clear now that we have to start from the end of the string.

Let's take a variable 'i' such that $i = n.length - 2$.Where 'n' is the string. Here 'n'='6537421' , So 'i' is pointing to 1 i.e. 'i'=5th index.



Here $arr[i+1] \leq arr[i]$,it means numbers are in increasing order,so even if we swap 2 &1 it will be 12 which is even less than 21.So we do nothing we just decrease i to i-1 . Now we again check if $arr[i] \geq arr[i+1]$ So again $4 \geq 2$, we do nothing.we again decrease i to i-1 . We continue to check and decrease i to i-1 , until $arr[i+1]$ becomes greater than $arr[i]$.



Till this point all the numbers are already in their **maximum state**. So, at $i=3$ th index we find that $arr[i+1] > arr[i]$, e.g. $7 > 3$. It means here we get a dip and at this point we get a number with the least weightage that can contribute in making a number just greater than the current number.

Step 2 Replace first dip with just Greater number: We can swap 3 with a number **just greater** than 3 in the array traversed so far i.e. from $i+1$ to $arr.length-1$.

Because swapping 3 with 7 will give us '73421' much more greater than 37421

and if we swap 3 with 1 it gives us '17423' which is even less than 37421

and if we swap 3 with 2 it gives '27431' which is also less than 37421

and if we swap 3 with 4 it gives us '47321' which is a bit greater than 37421 not more nor less.

So here we will be swapping 3 with 4. So the total number becomes '6547321'.

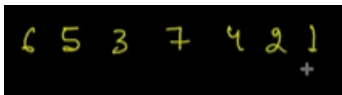
Step 3 reverse the number after dip index : After doing this we want to minimize the numbers after 4 also in 47321 so here we reverse all the numbers after 4. So the number becomes 41237 and the original number is.



So this is the number smaller in all the numbers greater than 6537421.

3. Code

4. Code Explanation



```

// User Section
public static String nextGreaterElement(String str) {
    char[] num = str.toCharArray();

    int i = num.length - 2;
    while(i >= 0 && num[i] >= num[i + 1]) {
        i--;
    }

    if(i >= 0) {
        int k = num.length - 1;
        while(num[k] <= num[i]) {
            k--;
        }
        char temp = num[i];
        num[i] = num[k];
        num[k] = temp;
    } else {
        return "-1";
    }

    String res = "";
    for(int j = 0; j <= i; j++) {
        res += num[j];
    }

    for(int j = num.length - 1; j > i; j--) {
        res += num[j];
    }

    return res;
}

```

Handwritten Annotations:

- Step 1:** Find first dip. (An arrow points to the '3' in the sequence 6 5 3 7 4 2 L, where '3' is the first element that is greater than the next element '7').
- Step 2:** Replace dip with just greater Number. (An arrow points from '3' to '7' in the sequence 6 5 3 7 4 2 L, resulting in 6 5 7 3 2 L).
- Step 3:** Reverse Number After dip index. (An arrow points from the sequence 6 5 7 3 2 L to 6 5 4 7 3 2 L, where the sequence after the dip is reversed).
- if No dip Found:** (An arrow points from the 'else' block to the return "-1" statement).
- ascending:** (An arrow points from the sequence 6 5 3 7 4 2 L to the sequence 6 5 4 7 3 2 L, indicating the final result is in ascending order).

5. Analysis

Time Complexity : $O(n)$

Here we traverse the whole string once so it takes $O(n)$ time.

Space Complexity: Constant

No extra space is used here so space complexity is constant.

Author: Damini Patel

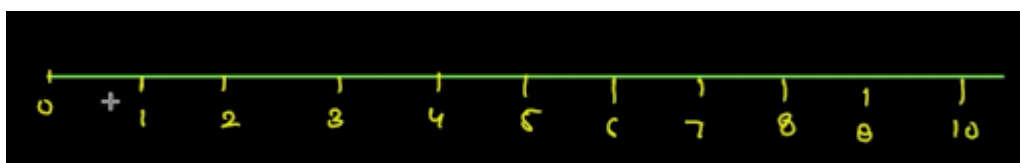
5.Min Jumps With +i -i Moves

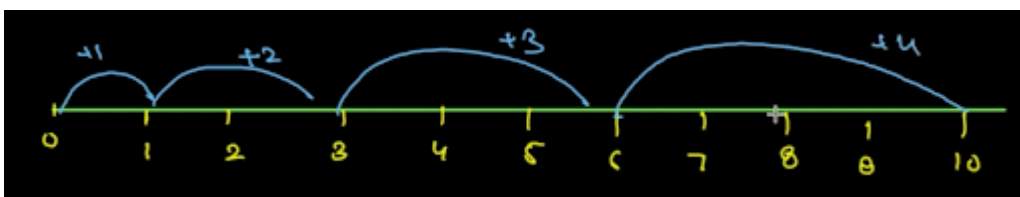
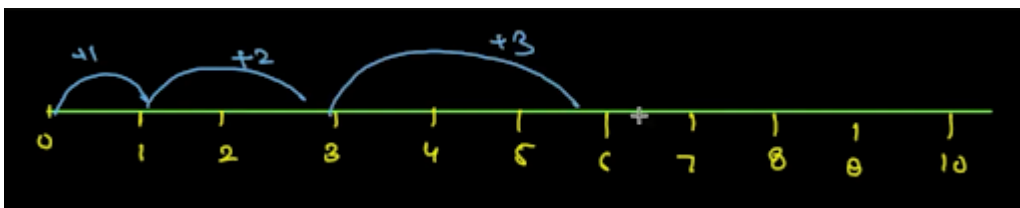
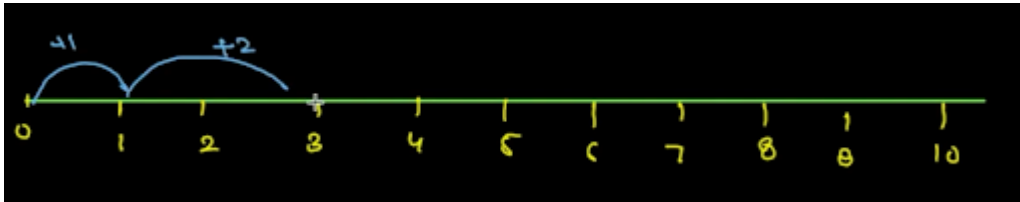
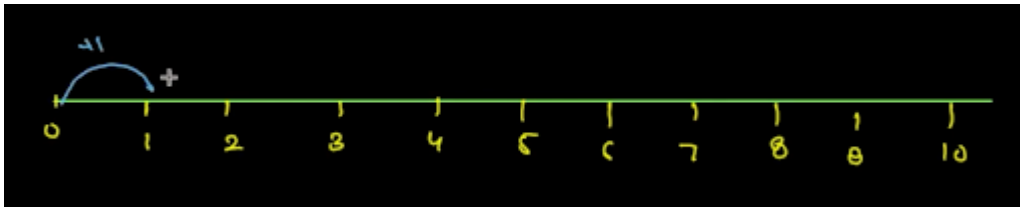
1. Problem Discussion:

1. Given an integer X. 2. The task is to find the minimum number of jumps to reach a point X in the number line starting from zero. 3. The first jump made can be of length one unit and each successive jump will be exactly one unit longer than the previous jump in length. 4. It is allowed to go either left or right in each jump. Example: $X = 9$ Ans = 5

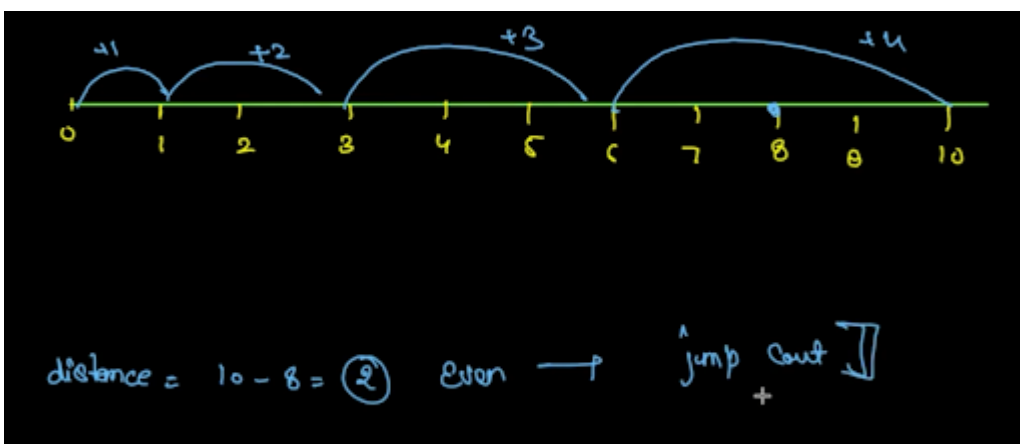
2. Approach:

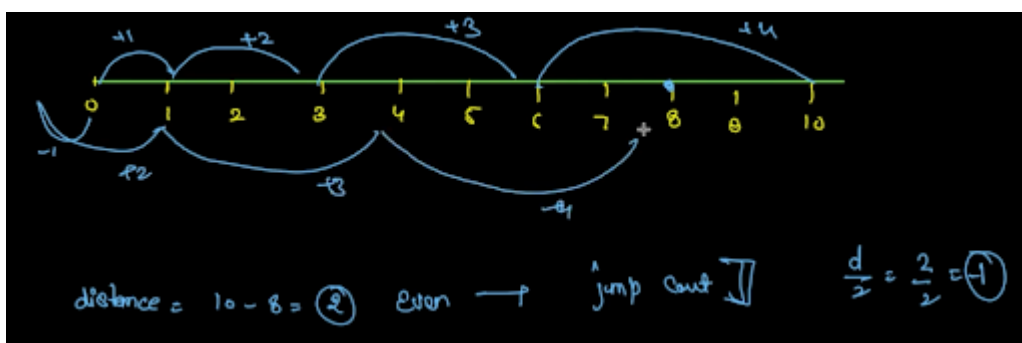
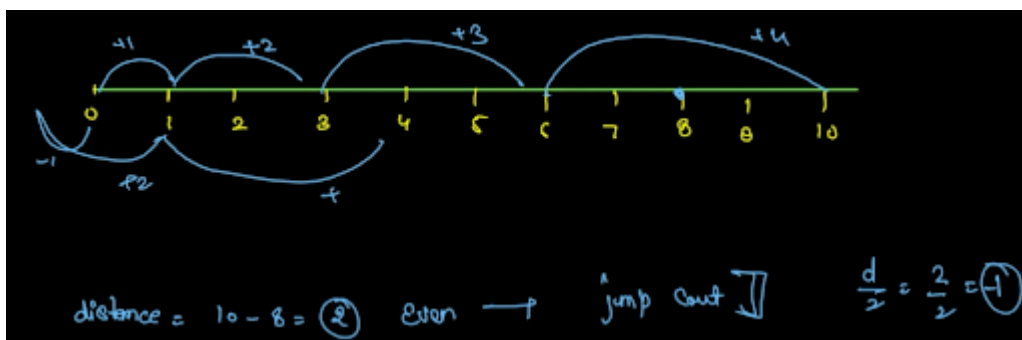
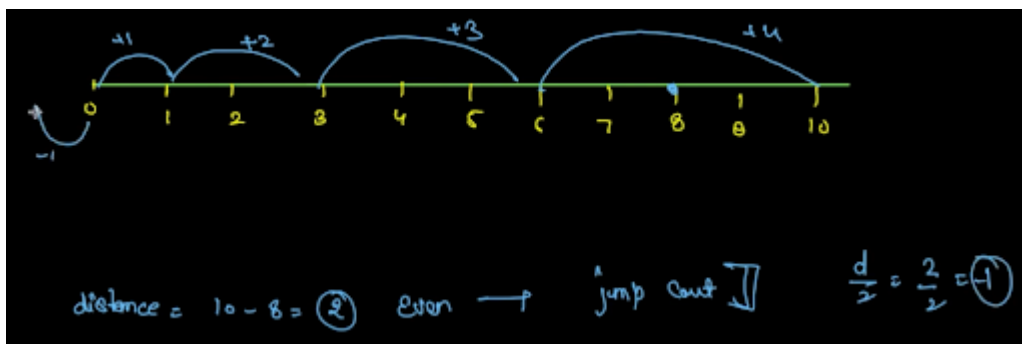
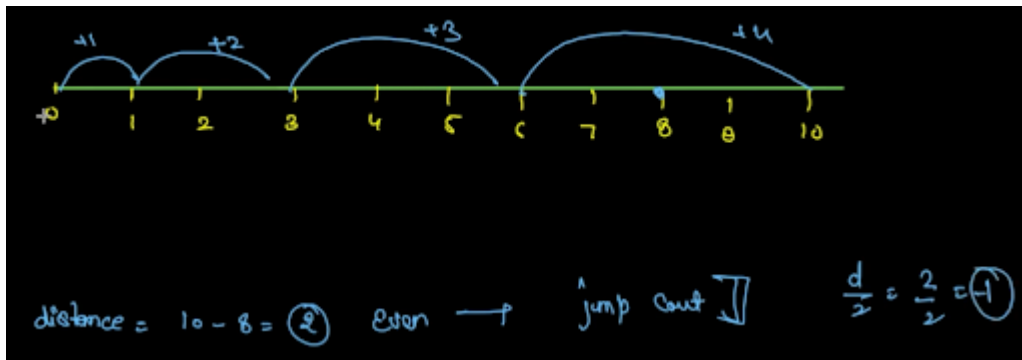
We have to reach a destination which is X. The jump will start from 1 and we can either take this jump in the negative or positive direction of the number line to reach X.





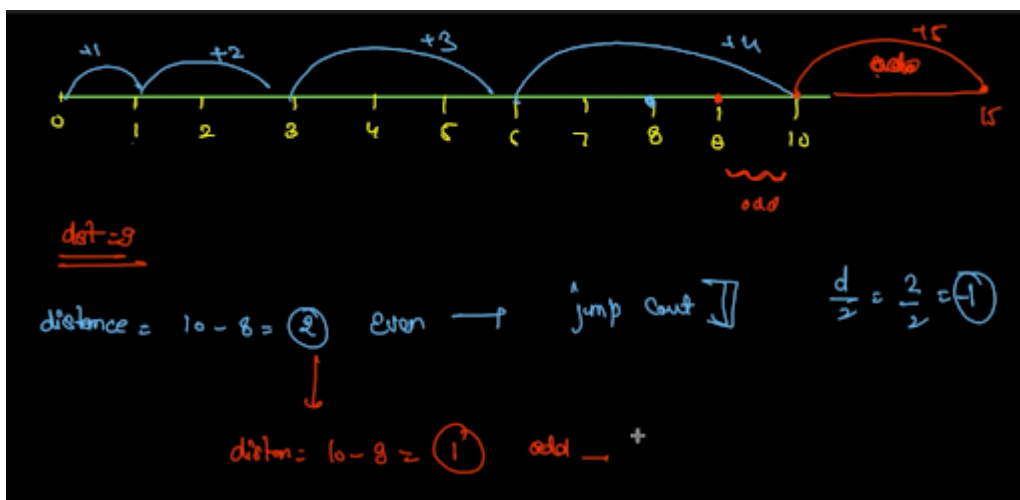
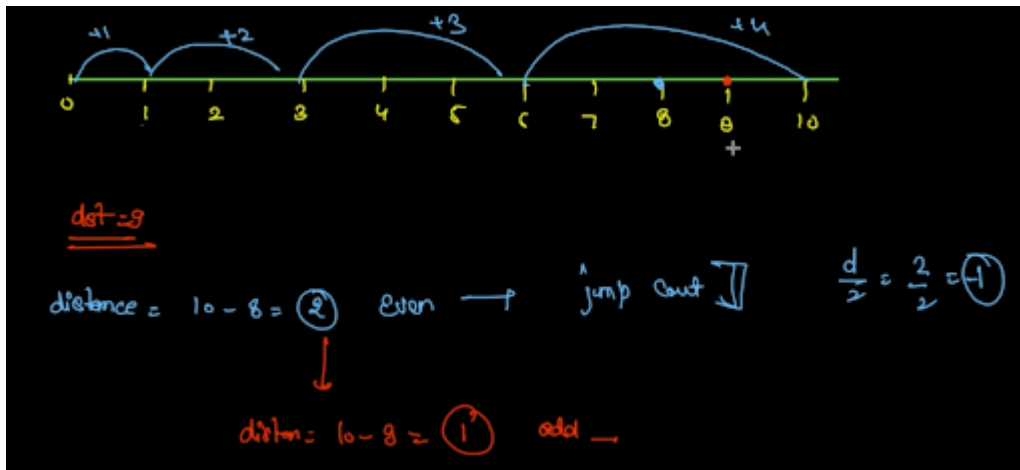
So, the approach is of following steps: 1.) To reach the destination we have to minimize the distance between the current position and the destination. For that take a variable j start from 1 and till the sum of all jumps is less than the destination increase the j by one. When the sum of all jumps is greater than or equal to the destination, then we have the minimum jumps to reach X (which is j). For calculating the sum of the jumps, take a variable sum to maintain the distance cover till j jumps.

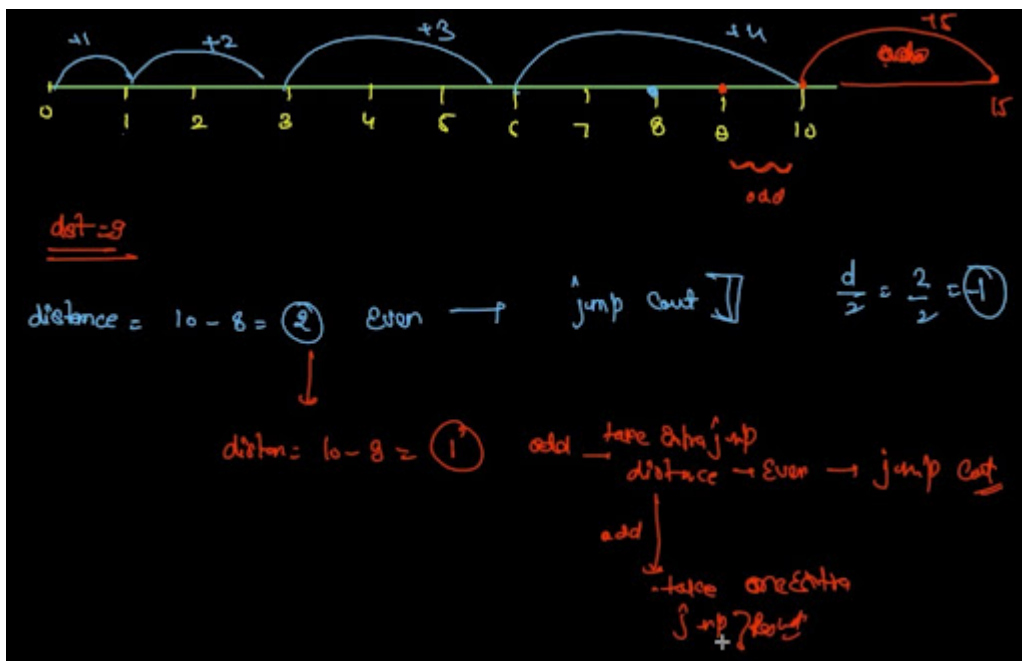
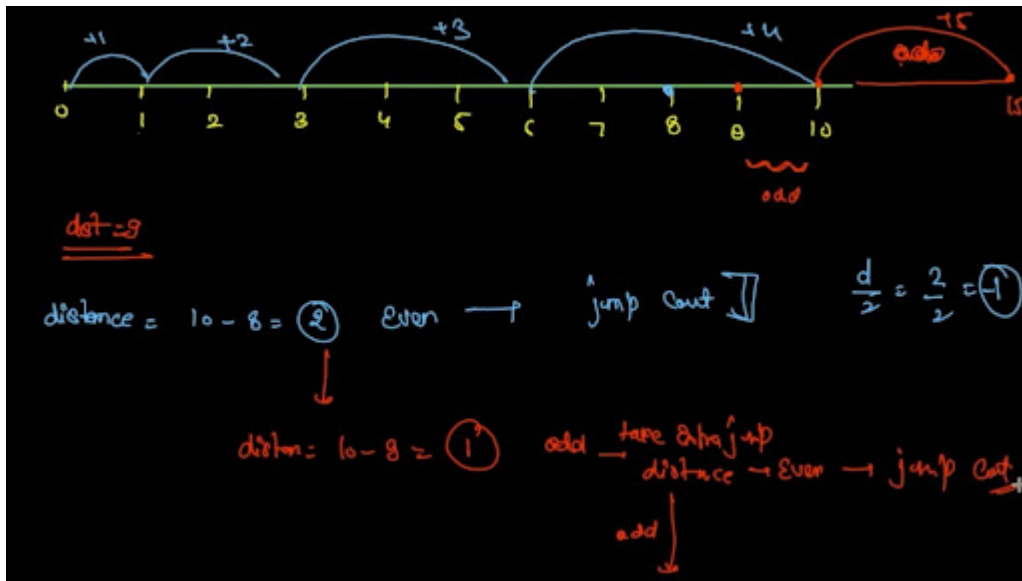




2.) Now note the distance between current position and the destination. There are two cases after measuring the distance these are: a) if the distance is even \rightarrow for this case the answer for the minimum number of jumps will be j . Why? If we find the distance between the current position and the destination and take a jump in the negative direction of the difference by 2 then we will reach the exact destination by taking j jumps. b) if the distance is odd \rightarrow in this case take one more jump in the forward direction. Now after taking one more jump again we have two cases: 1) if the distance between the current position and the destination is even then this time the answer is j . 2) if the distance between current position and the destination is odd here we have to take one more jump

again and this time the distance will be even between current position and the destination and the answer will be j . This time distance will be even because first time the distance was odd if the jump is even then new distance will be odd + even = odd, which makes the distance odd again but the next jump will increase by one i.e. previous jump was even, now it is increased by 1, so this time the jump is odd. Now, if we find the distance this time, previous odd distance + new odd jump which is equal to even again. Hence, at this moment the jump j will be the answer.





```
int jump = 1;
int sum = 0;

while(sum < x) {
    sum += jump;
    jump++;
}
```

Sum = 1 + 2 + 3 + 4 + ... \times time $< x$

no. of iterations

$$\text{sum} = 1 + 2 + 3 + 4 + \dots + \text{th time} < x$$

$$\frac{n(n+1)}{2} < x$$

```
int jump = 1;
int sum = 0;

while(sum < x) {
    sum += jump;
    jump++;
}
```

no. of iterations → complexity

$$\text{sum} = 1 + 2 + 3 + 4 + \dots + \text{th time} < x$$

$$\frac{n(n+1)}{2} < x$$

$$n^2 + n - 2x = 0$$

$$n = \frac{-1 \pm \sqrt{1 - 4(1)(-2x)}}{2}$$

$$n = \frac{-1 + \sqrt{1 + 8x}}{2}$$

$$n \approx \sqrt{x}$$

$ax^2 + bx + c = 0$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
int jump = 1;
int sum = 0;

while(sum < x) {
    sum += jump;
    jump++;
}
```

no. of iterations → complexity

$$\text{sum} = 1 + 2 + 3 + 4 + \dots + \text{th time} < x$$

$$\frac{n(n+1)}{2} < x$$

no. of iterations

$$n = \sqrt{x}$$

complexity = $O(\sqrt{x})$

$$n^2 + n - 2x = 0$$

$$n = \frac{-1 \pm \sqrt{1 - 4(1)(-2x)}}{2}$$

$$n = \frac{-1 + \sqrt{1 + 8x}}{2}$$

$$n \approx \sqrt{x}$$

$ax^2 + bx + c = 0$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

3. Code:

```

public class Main {
    // User Section
    public static int minJumps(int x) {
        int sum = 0;
        int i = 1;
        while(sum < x) {
            sum += i;
            i++;
        }

        if(sum == x) {
            return i - 1;
        } else if((sum - x) % 2 == 0) {
            return i - 1;
        } else if((sum + i - x) % 2 == 0) {
            return i;
        } else {
            return i + 1;
        }
    }
}

```

4. Analysis:

Time Complexity: $O(\sqrt{x})$, where x is the given number. Space Complexity: We have used constant space so space complexity is $O(1)$.

Author: Rohit Vishwakarma

6. Max Product Of Three Numbers

1. Problem Discussion :

You are given an array of size 'N'. Your task is to find three numbers whose product is maximum and return the maximum product.

Example) For $N = 6$, $arr[] = \{3, 2, -4, -6, 5, 1\}$

Explanation:

If we take the numbers -4, -6 and 5 we will get the maximum product as 120.

So, the output is 120.

2. Approach :

Naive Approach :

One basic approach to solve this problem is to generate all possible subsets of 3 for the given array and calculate the product of subset to find the maximum product. But this approach has a time complexity of $O(n^3)$.

Optimized Approach :

Our optimized approach will not be based on our naive approach instead our approach will be based on the intuition that to create the maximum product of 3 numbers we have to pick the largest 3 numbers from the array.

0	1	2	3	4	5	6
5	10	2	3	7	6	8

Maximum product = $7 * 8 * 10 = 560$

But this approach will only work when all elements in the array are positive integers.

0	1	2	3	4	5
3	2	-4	-6	5	1

Product of 3 largest elements = $2 * 3 * 5 = 30$

Maximum product = $-4 * -6 * 5 = 120$

So, now we have to improve over our original method such that it can overcome the problem of negative integers. Before improving the original method we have to analyze why our method failed for negative integers.

If we observe carefully we will find our method failed because we didn't account that if we multiply two negative integers we get a positive integer. So, if we pick 2 smallest numbers from the array and multiply them with the largest integer in the array we will have a second contender for creating the maximum product of 3 elements along with the product of largest 3 numbers.

Now, first we need the maximum, second maximum, third maximum, minimum and second minimum from the array.

0	1	2	3	4	5
3	2	-4	-6	5	1

Maximum = 5

Second Maximum = 3

Third Maximum = 2

Minimum = -6

Second Minimum = -4

After this work is simple we calculate the product of 2 smallest with the largest integer in the array and the product of largest 3 numbers, and see which is larger. Now we just need to return the larger product.

0	1	2	3	4	5
3	2	-4	-6	5	1

Maximum = 5

Second Maximum = 3

Third Maximum = 2

Minimum = -6

Second Minimum = -4

Product of 3 largest elements = $2 * 3 * 5 = 30$

Product of 2 smallest with largest element = $-4 * -6 * 5 = 120$

$120 > 30$, 120 will be returned

3. Code :

```
// ~~~~~User Section~~~~~
public static int maximumProduct(int[] arr) {
    int min = (int)1e9;
    int smin = (int)1e9;

    int max = -(int)1e9;
    int smax = -(int)1e9;
    int tmax = -(int)1e9;

    for(int i = 0; i < arr.length; i++) {
        if(arr[i] > max) {
            tmax = smax;
            smax = max;
            max = arr[i];
        } else if(arr[i] > smax) {
            tmax = smax;
            smax = arr[i];
        } else if(arr[i] > tmax) {
            tmax = arr[i];
        }

        if(arr[i] < min) {
            smin = min;
            min = arr[i];
        } else if(arr[i] < smin) {
            smin = arr[i];
        }
    }

    return Math.max(max * smax * tmax, max * min * smin);
}
```

4. Code Discussion :

1· First we have to declare and initialize maximum, second maximum, third maximum, minimum and second minimum variables.

```
int min = (int)1e9;
int smin = (int)1e9;

int max = -(int)1e9;
int smax = -(int)1e9;
int tmax = -(int)1e9;
```

2· Now, we just need to find the maximum, second maximum, third maximum, minimum and second minimum for our approach to work.

```

for(int i = 0; i < arr.length; i++) {
    if(arr[i] > max) {
        tmax = smax;
        smax = max;
        max = arr[i];
    } else if(arr[i] > smax) {
        tmax = smax;
        smax = arr[i];
    } else if(arr[i] > tmax) {
        tmax = arr[i];
    }

    if(arr[i] < min) {
        smin = min;
        min = arr[i];
    } else if(arr[i] < smin) {
        smin = arr[i];
    }
}

```

3. Now, we just have to return the max between the product of maximum, second maximum and third maximum, and the product of maximum, minimum and second minimum.

```

return Math.max(max * smax * tmax, max * min * smin);

```

5. Analysis :

Time Complexity : $O(n)$

Overall time complexity of the function will be $O(n)$ because we used only one loop for calculating maximum, second maximum, third maximum, minimum and second minimum.

Space Complexity : $O(1)$

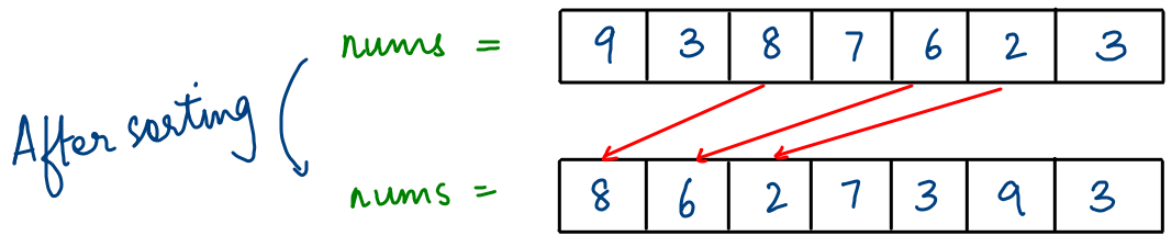
Since, no extra spaces are used to store numbers, therefore the space complexity is constant.

Author: Kunal Singh Bhandari

7.Sort Array by Parity

1. Problem Discussion

You will be given an array `nums` of non-negative integers. You will be required to arrange the elements of the array in a specific order, such that all even elements will be followed by all odd elements. [even→odd] Let's take a look at this example: `nums = {9, 3, 8, 7, 6, 2, 3}` After sorting, our output should be: `nums = {8, 6, 2, 7, 3, 9, 3}`



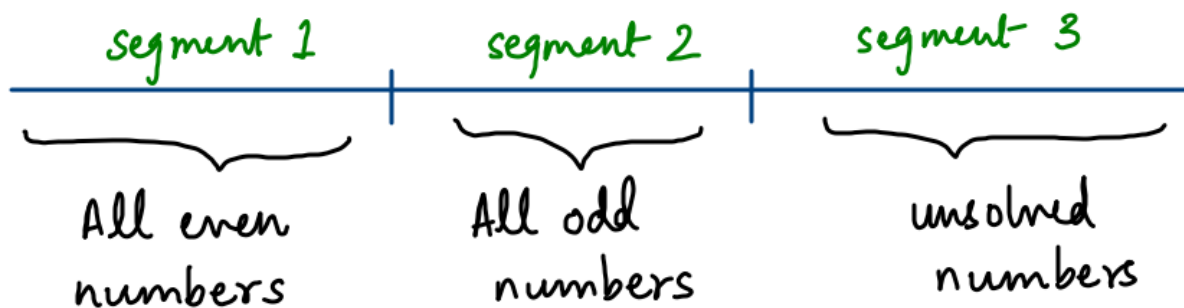
Order of even numbers is preserved

Note: You have to preserve the order of even elements without using extra space. I would strongly recommend you to check out the Sort 01 question (from level 1/Time and space complexity module), As the concepts used in that question are quite similar to the one we're going to use in this question.

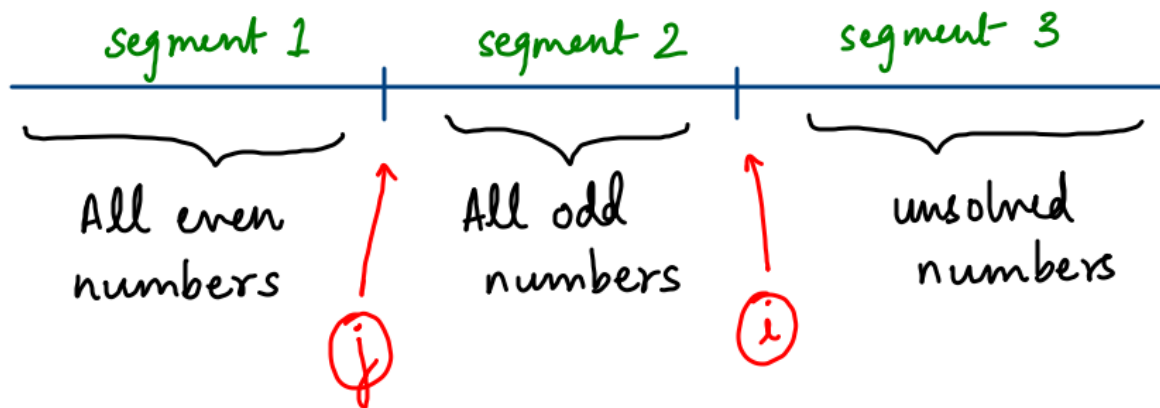
Sort 01

2. Approach

Assuming you already know how to solve the Sort-01 question from level 1/Time and space complexity module. This approach will be fairly straightforward. We will use the same range/segment approach that we used in sort-01, that is we are going to divide the array into 3 segments. Segment 1, for all even numbers. Segment 2, for all odd numbers. Segment 3, for all unsolved numbers so far.



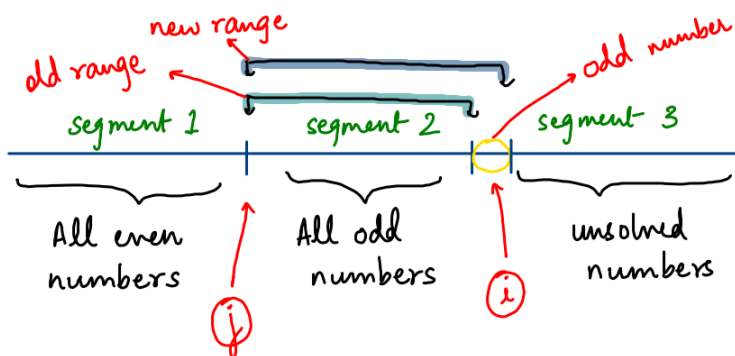
Let's take two pointers, i and j, which represent the first odd number and the first unsolved number respectively.



j = First odd number

i = First unsolved number

Now, as we traverse the array, our number can either be even or odd, which means, it will either belong to the first segment or the second segment. So, if the number is, let's say, odd. Then, we will just increase the range of the second segment, i.e., $i++$.



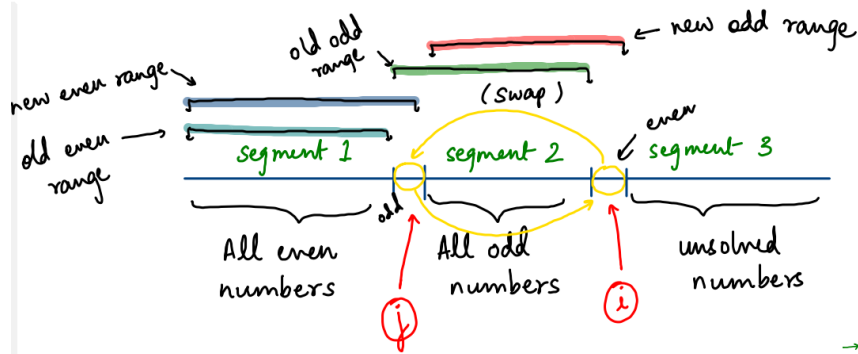
j = First odd number

i = First unsolved number

$arr[i] \rightarrow \text{odd}$

- Include that number in segment 2 .
- So, segment 2 will increase .
- $i++$

But if the number is even. We will swap the first element of the second segment with the current number, so that our current number becomes the last element of the first segment and both the segments increase by one, i.e., $i++$ and $j++$. Let's look at the example:



j = First odd number
 i = First unsolved number

$arr[i] \rightarrow odd$
 \rightarrow Include that number in segment 2
 \rightarrow So, segment 2 will increase.
 $\rightarrow i++$

$arr[i] \rightarrow even$

- Include that number in segment 1.
- So, swap the first odd value with $arr[i]$
- Hence, both segments will increase
- $i++$, $j++$

So, after our complete iteration, we would have the right answer.

nums =

9	3	8	7	6	2	3
---	---	---	---	---	---	---

$i=0, j=0$, $arr[i] == odd$, $i++$

$i \rightarrow 1$

nums =

9	3	8	7	6	2	3
---	---	---	---	---	---	---

$i=1, j=0$, $arr[i] == odd$, $i++$

$i \rightarrow 2$

nums =

9	3	8	7	6	2	3
---	---	---	---	---	---	---

$i=2, j=0$, $arr[i] == even$
 swap $arr[i]$ & $arr[j]$, $i++$ & $j++$

$i \rightarrow 1$, $j \rightarrow 1$

nums =

8	3	9	7	6	2	3
---	---	---	---	---	---	---

$i=3, j=1$, $arr[i] == odd$, $i++$

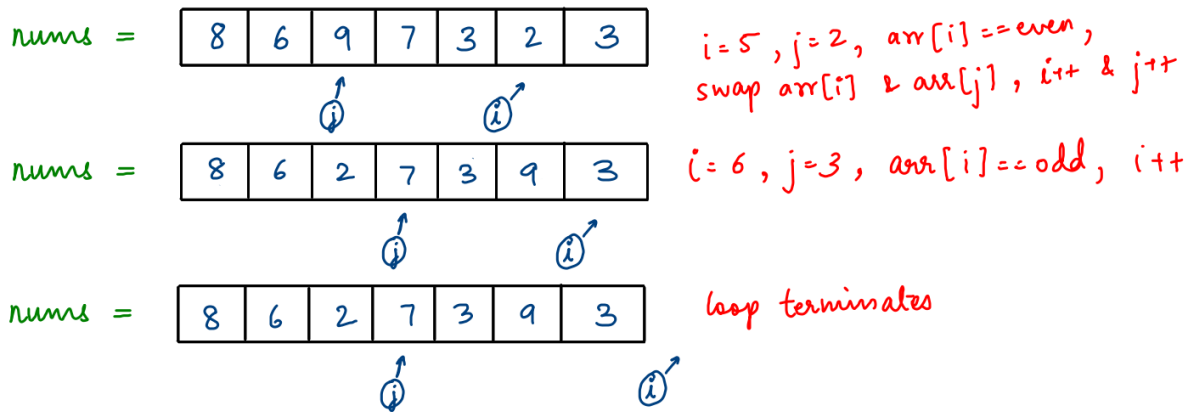
$i \rightarrow 4$, $j \rightarrow 2$

nums =

8	3	9	7	6	2	3
---	---	---	---	---	---	---

$i=4, j=1$, $arr[i] == even$,
 swap $arr[i]$ & $arr[j]$, $i++$ & $j++$

$i \rightarrow 5$, $j \rightarrow 3$



3. Code

```
public class Main {

    // ~~~~~User Section~~~~~

    private static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    public static void sortByParity(int[] nums) {
        int i = 0;
        int j = 0;

        while(i < nums.length) {
            if(nums[i] % 2 == 0) {
                swap(nums, i, j);
                i++;
                j++;
            } else {
                i++;
            }
        }
    }
}
```

4. Analysis

Time Complexity

$O(N)$, as we're only traversing the array once to completely sort the array, assuming there are N elements in the array.

Space Complexity

$O(1)$ auxiliary space is being used

Author: Varun Kumar

8.Best Meeting Point

1. Problem Discussion :

In this problem a group of two or more people wants to meet. You are given a 2D grid of values 0 or 1, where each 1 marks the home of someone in the group. Your task is to find and minimize the total travel distance.

You have to return min distance where distance is calculated using 'Manhattan Distance', where $\text{distance}(p1, p2) = |p2.x - p1.x| + |p2.y - p1.y|$.

Example) For $N = 3$ and $M = 5$, $\text{grid}[] = \{\{1,0,0,0,1\}, \{0,0,0,0,0\}, \{0,0,1,0,0\}\}$

The point (0,2) is an ideal meeting point, as the total travel distance of $2 + 2 + 2 = 6$ is minimal.

So, the output is 6.

2. Approach :

Naive Approach :

One basic approach to solve this problem is to calculate the distance between all possible meeting points from every house and find the minimum distance. But this approach will take too much time because of multiple comparisons.

Optimized Approach :

Our optimized approach will be based on first finding the ideal meeting point because after finding the ideal meeting point calculating the minimum distance will be very easy.

But how can we find the ideal meeting point ?

Original Grid					
	0	1	2	3	4
0	1	0	0	0	1
1	0	0	0	0	0
2	0	0	1	0	0

Currently, we don't know how to find the ideal point but we know that we have to store coordinates of all the houses because in the end we have to calculate the distance between all the houses and the ideal meeting point.

So, first we need to store coordinates of all the houses. For that we need to traverse all over our grid to store the x and y coordinates of all houses or the row and column of elements with value equal to one.

X-Coordinate (Row)	Y-Coordinate (Column)
0	0
0	4
2	2

Now, we have coordinates of all houses and we only need to find an ideal meeting point.

If we carefully think about our current situation, then we will find that we are in a 2d plane with some points and we need to find a point which is at the minimum distance from all the points. Isn't it same as finding the centroid of a two-dimensional region? So, If we can find the centroid of all the points on the grid with individuals, then that will be the place with the shortest distance travelled. We can easily find the centroid but most of the time we will get 8 "closest points" because the value of the centroid will be in decimal and among these closest points only one will be an ideal meeting point.

So, to find the closest point to the centroid we will take the median of all the x-coordinates and y-coordinates of all houses to get x and y coordinates of the ideal meeting point.

Ideal Meeting Point


X-Coordinate = 0


Y-Coordinate = 2

Now, we have to just calculate the Manhattan distance for every house from the ideal meeting point and add them.

Original Grid

	0	1	2	3	4
0	1	0	0	0	1
1	0	0	0	0	0
2	0	0	1	0	0

Houses : 

Ideal Meeting Point : 

	X-Coordinate (Row)	Y-Coordinate (Column)	Distance	
	0	0	2	
	0	4	2	
	2	2	2	

Minimum distance = 6

3. Code :

```
// *****User Section*****
public static int minTotalDistance(int[][] grid) {

    ArrayList<Integer> xcord = new ArrayList<>();
    ArrayList<Integer> ycord = new ArrayList<>();
    for(int r = 0; r < grid.length; r++) {
        for(int c = 0; c < grid[0].length; c++) {
            if(grid[r][c] == 1) {
                xcord.add(r);
                ycord.add(c);
            }
        }
    }

    int x = xcord.get(xcord.size() / 2);
    int y = ycord.get(ycord.size() / 2);

    int dist = 0;
    for(int i = 0; i < xcord.size(); i++) {
        dist += Math.abs(xcord.get(i) - x) + Math.abs(ycord.get(i) - y);
    }

    return dist;
}
```

4. Code Discussion :

1. First we have to store the x and y coordinates of all houses of the people who are going to meet.

```
ArrayList<Integer> xcord = new ArrayList<>();
ArrayList<Integer> ycord = new ArrayList<>();
for(int r = 0; r < grid.length; r++) {
    for(int c = 0; c < grid[0].length; c++) {
        if(grid[r][c] == 1) {
            xcord.add(r);
            ycord.add(c);
        }
    }
}
```

2. Here, we need to sort the x coordinates and y coordinates so that the median of x coordinates and y coordinates can be accessed easily.

```
Collections.sort(xcord);
Collections.sort(ycord);
```

3. Now, we just need to find the coordinates of the ideal meeting point which will be the median of x coordinates and y coordinates.


```
int x = xcord.get(xcord.size() / 2);
int y = ycord.get(ycord.size() / 2);
```

4. Now, we just have to calculate Manhattan distance from all coordinates to the ideal meeting point and add them together to get the minimum total distance.

```
int dist = 0;
for(int i = 0; i < xcord.size(); i++) {
    dist += Math.abs(xcord.get(i) - x) + Math.abs(ycord.get(i) - y);
}
```

5. Now, we just need to return the minimum distance.

```
return dist;
```

5. Analysis :

Time Complexity : $O(n*m)$

Overall time complexity of the function will be $O(n*m)$ because we traverse all over our grid to store the x and y coordinates of all houses.

Space Complexity : $O(n)$

Since, we used arrays to store the x and y coordinates of all houses, the space complexity of the function becomes $O(n)$.

Author: Kunal Singh Bhandari

9.Sieve Of Eratosthenes

1. Problem Discussion:

You will be given an Integer 'n'. You have to print all primes from 2 to 'n'. The time complexity should be lesser than $(n \sqrt{n})$. For e.g. input:

```
n = 10
```

Output:

```
2 3 5 7
```

I hope the problem is clear. Let's discuss the approach now.....

2. Approach:

We know that to check if a number 'i' is prime or not we run a loop till (\sqrt{i}) , So it takes (\sqrt{i}) time. So to check if 'n' numbers are prime or not it will take $n * \sqrt{n}$ time. Which is not less than $(n \sqrt{n})$. So here we are going to discuss another approach to solve this problem. In this approach we are going to take some extra space. Here we will be taking a boolean array of size $n+1$. Where each index represents a prime number under range 2 to 'n'. Initially we mark all the numbers true initially which means all numbers are marked as prime initially. (true \rightarrow prime & false \rightarrow non-prime). It means we suppose initially all the numbers are prime numbers. Suppose this is the boolean array, say 'primes' and $n=37$.

X 0	7 → T	14 → T	22 → T	30 → T
X 1	8 → T	15 → T	23 → T	31 → T
2 → T	9 → T	16 → T	24 → T	32 → T
3 → T	10 → T	17 → T	25 → T	33 → T
4 → T	11 → T	18 → T	26 → T	34 → T
5 → T	12 → T	19 → T	27 → T	35 → T
6 → T	13 → T	20 → T	28 → T	36 → T
		21 → T	29 → T	37 → T

And we also take an iterator say 'i' which points to prime numbers in the array which is initialised to 2, i.e. i=2.

$$i = 2$$

Now we know that all the multiples of that prime number will definitely not be prime numbers because they can be divided by that prime number itself. So next we are going to unmark all the multiples of that particular prime number as false in the 'primes' array. Then we increase our 'i' to 'i+1' i.e. we move to the next prime number. So first of all we check if that number is a prime number or not i.e. `primes[i] == true`, if it is then mark all of its multiples as false. We continue to do so till 'i' is less than equal to (\sqrt{n}) . because after (\sqrt{n}) all the multiples of all the prime numbers less than equal to (\sqrt{n}) i.e. $a*b$ start to repeat, i.e. After (\sqrt{n}) it becomes $b*a$. For example :

<u>36</u>	1x 36	36x 1
	2x 18	18x 2
	3x 12	12x 3
	4x 9	9x 4
	6x 6	

Here $n=36$ so after (\sqrt{n}) i.e. 6, all the pairs start to repeat in the reverse order. i.e. $3*12$ is $12*3$, $2*18$ is $18*2$, but we have already marked 18 & 12 as multiple of 2. So there is no sense of marking 2 as a multiple of 12 and 18. So we will be marking the multiples of all the prime numbers less than (\sqrt{n}) as false only. We don't need to run a loop till 'n'. So here we marked all the multiples of 2 as false, i.e. those numbers are non-prime numbers.

0	7 → T	14 → F	22 → F	30 → F
1	8 → F	15 → T	23 → T	31 → T
2 → T	9 → T	16 → F	24 → F	32 → F
3 → T	10 → F	17 → T	25 → T	33 → T
4 → F	11 → T	18 → F	26 → F	34 → F
5 → T	12 → F	19 → T	27 → T	35 → T
6 → F	13 → T	20 → F	28 → F	36 → F
		21 → T	29 → T	37 → T

After that we mark the multiples of 3 as non-prime.

i = 2 3

So the 'primes' array after marking the multiples of 3 as false:

0	7 → T	14 → F	22 → F	30 → F
1	8 → F	15 → F	23 → T	31 → T
2 → T	9 → F	16 → F	24 → F	32 → F
3 → T	10 → F	17 → T	25 → T	33 → F
4 → F	11 → T	18 → F	26 → F	34 → F
5 → T	12 → F	19 → T	27 → F	35 → T
6 → F	13 → T	20 → F	28 → F	36 → F
		21 → F	29 → T	37 → T

Similarly we will mark the multiples of 4,5 and 6 as false. We mark the multiples of all the prime numbers as false till 6 only because $(\sqrt{36}) = 6$. So the array after marking all the multiples of all the prime numbers less than equal to 6 is:

0	7 → T	14 → F	22 → F	30 → F
1	8 → F	15 → F	23 → T	31 → T
2 → T	9 → F	16 → F	24 → F	32 → F
3 → T	10 → F	17 → T	25 → F	33 → F
4 → F	11 → T	18 → F	26 → F	34 → F
5 → T	12 → F	19 → T	27 → F	35 → F
6 → F	13 → T	20 → F	28 → F	36 → F
		21 → F	29 → T	37 → T

So all the numbers that are marked as 'true' will be our prime numbers. And also with the help of the 'primes' array we can answer if a number is prime or not in $O(1)$ time just by checking true or false.

3. Code:

```
// ~~~~~User Section~~~~~
public static void printPrimeUsingSieve(int n) {
    // write your code here
    boolean[] isPrime = new boolean[n+1];
    Arrays.fill(isPrime, true);

    //precalculation part
    for(int i=2; i*i<=n; i++){
        if(isPrime[i] == true){
            //unmarking all the multiples of i;
            for(int j= i+i; j<=n; j+=i){
                isPrime[j] = false;
            }
        }
    }
    for(int i = 2; i<=n; i++){
        if(isPrime[i] == true){
            System.out.print(i+" ");
        }
    }
}
}
```

4. Code explanation

First of all we take a boolean array isPrime and fill it with true.

```
for(int i = 2; i < isPrime.length; i++) {
    if(isPrime[i] == true) {
        System.out.print(i + " ");
    }
}
System.out.println();
```

Then we mark the multiples of all the prime numbers till (root n) as false.

```
for(int i = 2; i * i <= n; i++) {
    if(isPrime[i] == true) {
        for(int j = i + i; j <= n; j += i) {
            isPrime[j] = false;
        }
    }
}
```

Here is the code to print all the numbers that are marked as true i.e. all the prime numbers.

```
for(int i = 2; i < isPrime.length; i++) {
    if(isPrime[i] == true) {
        System.out.print(i + " ");
    }
}
System.out.println();
```

5. Analysis

Time Complexity: $O(n \log(\log n))$

Here we have an array of size 'n' So here for $i=2$ we are taking a jump of 2 every time till n so it takes $n/2$ time ,for $i=3$ we are taking a jump of 3 every time so it takes $n/3$ time,similarly it will continue till

'i' less than equal to (\sqrt{n}) . So total time will be: $n/2 + n/3 + n/4 + \dots + n/(\sqrt{n}) = n(1/2 + 1/3 + 1/4 + \dots + 1/(\sqrt{n})) = n(\log(\log(n)))$.

Space Complexity: $O(n)$

Here we take an array of 'n' size so extra space used here $O(n)$.

Author: Damini Patel

10. Segmented Sieve

1. Problem Discussion:

1. Generate all primes between 'a' and 'b' (both are included). 2. Print every number in new line. 3. Allowed time Complexity: $O(n \log(\log n))$, where $n = b - a$. 4. Allowed Space Complexity: $O(n)$, where $n = b - a$; Note: Please focus on constraints. Constraints 1. $1 \leq a \leq b \leq 10^9$ 2. $b - a \leq 10^5$ Example: $A = 22$ $B = 51$ Ans = 23, 29, 31, 37, 41, 43, 47

2. Approach:



We have the range from a to b. We can find all the numbers isprime or not one by one. For checking isprime of a number k it will take \sqrt{k} . And we have n numbers where $n = b - a$. So, for finding n numbers isprime or not it will take $O(n \cdot \sqrt{k})$ times. We have allowed time complexity which is $O(n \log(\log n))$ but this approach will take $O(n \cdot \sqrt{k})$. This means this approach is not that efficient.

Naïve approach 2:

Generate prime array using sieve algorithm. Generate the prime from 2 to the upper range we have given i.e. till b. We have given two more constraints where 1) $1 \leq a \leq b \leq 10^9$, and 2) $b - a \leq 10^5$. Here in the first constraint we have a range of b till 10^9 , now let's say the value of $a = 10^8$ and $b = 10^9$. So for precalculation of the prime no. the boolean array will be of range $10^9 + 1$, but we just have to calculate $b - a$ no.s ($10^9 - 10^8$). In this approach we are using extra space which is not that efficient approach. And also it will increase time complexity as well.

Trivial approach:

In this approach, for precalculation of the no.s for prime i.e. using the sieve algorithm, we will use the boolean array of size $b - a + 1$. Now, generate the prime from 2 to \sqrt{b} using sieve algorithm (this problem is a follow up problem of the sieve of eratosthenes). Now, the next step is marking using the prime array for not prime numbers. For this step we need the starting index for every prime number and then mark all the multiples from 2 to \sqrt{b} in the prime array, where each index is mapped as $\text{index} + a$.

What- lower Range = 22, upper Range = 51
(boolean) arr $\rightarrow b - a + 1 = 51 - 22 + 1 = 30$
False \rightarrow prime
True \rightarrow Not prime

What:- lower Range = 22, upper Range = 51
 (boolean) $arr \rightarrow b-a+1 = 51-22+1 = 30$

False \rightarrow prime
 True \rightarrow Not prime

0 \rightarrow 22	10 \rightarrow 32	20 \rightarrow 42
1 \rightarrow 23	11 \rightarrow 33	21 \rightarrow 43
2 \rightarrow 24	12 \rightarrow 34	22 \rightarrow 44
3 \rightarrow 25	13 \rightarrow 35	23 \rightarrow 45
4 \rightarrow 26	14 \rightarrow 36	24 \rightarrow 46
5 \rightarrow 27	15 \rightarrow 37	25 \rightarrow 47
6 \rightarrow 28	16 \rightarrow 38	26 \rightarrow 48
7 \rightarrow 29	17 \rightarrow 39	27 \rightarrow 49
8 \rightarrow 30	18 \rightarrow 40	28 \rightarrow 50
9 \rightarrow 31	19 \rightarrow 41	29 \rightarrow 51

What:- ① lower Range = 22, upper Range = 51
 (boolean) $arr \rightarrow b-a+1 = 51-22+1 = 30$

False \rightarrow prime
 True \rightarrow Not prime

② Generate prime from 2 to \sqrt{b} (using sieve)
 2 to $\sqrt{51} \approx 7$
 2, 3, 5, 7

0 \rightarrow 22	10 \rightarrow 32 +	20 \rightarrow 42
1 \rightarrow 23	11 \rightarrow 33	21 \rightarrow 43
2 \rightarrow 24	12 \rightarrow 34	22 \rightarrow 44
3 \rightarrow 25	13 \rightarrow 35	23 \rightarrow 45
4 \rightarrow 26	14 \rightarrow 36	24 \rightarrow 46
5 \rightarrow 27	15 \rightarrow 37	25 \rightarrow 47
6 \rightarrow 28	16 \rightarrow 38	26 \rightarrow 48
7 \rightarrow 29	17 \rightarrow 39	27 \rightarrow 49
8 \rightarrow 30	18 \rightarrow 40	28 \rightarrow 50
9 \rightarrow 31	19 \rightarrow 41	29 \rightarrow 51

What: ① lower Range = 22, upper Range = 51
(boolean) arr → $b - a + 1 = 51 - 22 + 1 = 30$

② Generate prime from 2 to \sqrt{b} . (using sieve)
2 to $\sqrt{51} \approx 7$

③ marking using primes.
→ Starting index for every prime Number.

0 → 22 → T	10 → 32 → T	20 → 42 → T
1 → 23	11 → 33 → T	21 → 43
2 → 24 → T	12 → 34 → T	22 → 44 → T
3 → 25 → T	13 → 35 → T	23 → 45 → T
4 → 26 → T	14 → 36 → T	24 → 46 → T
5 → 27 → T	15 → 37	25 → 47
6 → 28 → T	16 → 38 → T	26 → 48 → T
7 → 29	17 → 39 → T	27 → 49 → T
8 → 30 → T	18 → 40 → T	28 → 50 → T
9 → 31	19 → 41	29 → 51 → T

What: ① lower Range = 22, upper Range = 51
(boolean) arr → $b - a + 1 = 51 - 22 + 1 = 30$

② Generate prime from 2 to \sqrt{b} . (using sieve)
2 to $\sqrt{51} \approx 7$

③ marking using primes.
→ Starting index for every prime Number.

0 → 22 → T	10 → 32 → T	20 → 42 → T
1 → 23	11 → 33 → T	21 → 43
2 → 24 → T	12 → 34 → T	22 → 44 → T
3 → 25 → T	13 → 35 → T	23 → 45 → T
4 → 26 → T	14 → 36 → T	24 → 46 → T
5 → 27 → T	15 → 37	25 → 47
6 → 28 → T	16 → 38 → T	26 → 48 → T
7 → 29	17 → 39 → T	27 → 49 → T
8 → 30 → T	18 → 40 → T	28 → 50 → T
9 → 31	19 → 41	29 → 51 → T

Now the question is how will we find the starting index of prime multiple? For finding the starting index of any no. first we need to find the multiple which will be equal to $(\text{int})\text{ceil}((a * 1.0) / \text{prime}[i])$, this will give us the multiple. Now we will use this to find starting index which will be $(\text{multiple} * \text{prime}[i] - a)$, where a is the base value. If we use this to find multiple there will be a case come that we will miss the case is if $a = 2$ and $b = 100$. Now the primes for this will be 2, 3, 5, 7, Multiple = $(2 * 1.0) / 2 = 1$ Starting index = $1 * 2 - 2 = 0$. Here at 0 index of isprime array is map to 2 and we can't mark it as not prime because 2 is a prime no. so, to avoid this if the multiple = 1 than we will increase multiple by 1. If we increase the multiple by 1, now multiple is 2 so starting index = $2 * 2 - 2 = 2$, where 2 is mapped with value 4 which is not a prime no.

Starting Index of prime Multiple-

prime \rightarrow 2 3 5 7

$$a = 22$$

$$b = 51$$

$$\text{multiple} = \left(\frac{a * 1.0}{\text{prime}[i]} \right)_{\text{ceil}} \quad m = \left(\frac{22 * 1.0}{7} \right)_{\text{ceil}} = \left(\frac{22}{7} \right)_{\text{ceil}} = (3.14...) \downarrow 4$$

$$\text{Starting Index} = \underbrace{\text{multiple} * \text{prime}[i]}_{\text{multiple}} - \underbrace{a}_{\text{base value}}$$

$$\text{S.I} = 4 * 7 - 22 = 28 - 22 = 6$$

Starting Index of prime Multiple-

prime \rightarrow 2 3 5 7

$$\text{multiple} = \left(\frac{a * 1.0}{\text{prime}[i]} \right)_{\text{ceil}}$$

$$\text{multiple} = 1$$

$$\text{Starting Index} = \underbrace{\text{multiple} * \text{prime}[i]}_{\text{multiple}} - \underbrace{a}_{\text{base value}}$$

$$\text{S.I} = 1 * 2 - 2 = 0$$

$$\text{multiple} = \left(\frac{2 * 1.0}{2} \right) = 1$$

$$\text{S.I} = 1 * 2 - 2 = 0$$

$$a = 22$$

$$b = 51$$

$$m = \left(\frac{22 * 1.0}{7} \right)_{\text{ceil}} = \left(\frac{22}{7} \right)_{\text{ceil}} = (3.14...) \downarrow 4$$

$$a = 2$$

$$b = 100$$

$$\text{prime} = 2, 3, 5, 7, \dots$$

Starting Index of prime Multiple-

prime $\rightarrow 2, 3, 5, 7$

$$\text{multiple} = \left(\frac{a * 1.0}{\text{prime}[i]} \right)_{\text{ceil}}$$

multiple = 1, multiple++

$$\text{Starting Index} = \underbrace{\text{multiple} * \text{prime}[i]}_{\text{multiple}} - \underbrace{a}_{\text{base value}}$$

$a = 2$
 $b = 51$
 $m = \left(\frac{22 * 1.0}{7} \right)_{\text{ceil}} = \left(\frac{22}{7} \right)_{\text{ceil}} = (3.14...)_{\text{ceil}} = 4$

$a = 2$
 $b = 100$
 $\text{prime} = (2) 3 \leq 7 \dots$

$21 = 4 * 7 - 22 = 28 - 22$
 $\text{multiple} = \left(\frac{2 * 1.0}{2} \right) = 1$
 $SI = 2 * 2 - 2 = 2$

complexity \rightarrow n Elements $O(n \log \log n)$

$$T(n) = \left[\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \frac{n}{7} + \dots + \frac{n}{n} \right]$$

$\frac{n}{\text{prime}}$
 $T(n) = n \left[\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \dots + \frac{1}{n} \right]$

$= n \log(\log n)$
 Convergence and divergence-

$T(n) = n \log \log n$
 $= 5n \equiv O(n)$

$n = 10^9 = 2^{32}$
 $\log_2 n = \log_2 2^{32}$
 $\log_2 (32) = 2^5$
 $\log_2 (2^5) = 5$

Why is this approach working? We have used the sieve algorithm where we have saved all the isprime of the no. and the indexes are mapped from a to b. If the no. is prime which starts from 2 then all the multiples of these primes should be marked as not prime. For that we are marking all multiples as not prime.

3. Code Discussion:

Step1: Make a root of b Step2: get primes till root b by using sieve algorithm.

```
// ~~~~~User Section~~~~~
public static ArrayList<Integer> sieveOfEratosthenes(int N) {
    boolean[] arr = new boolean[N + 1]; // false - prime || true - non prime
    for (int i = 2; i * i <= N; i++) {
        if (arr[i] == false) {
            for (int j = i * 2; j <= N; j += i) {
                arr[j] = true;
            }
        }
    }
    ArrayList<Integer> ans = new ArrayList<>();
    for (int i = 2; i <= N; i++) {
        if (arr[i] == false) {
            ans.add(i);
        }
    }
    return ans;
}
```

Step3: marking isprimes using prime. Step4: Find starting index for marking by finding multiple and the starting index.

```
for (int i : primes) {
    int multiple = (int) Math.ceil(a * 1.0 / i);
    if (multiple == 1) {
        multiple++;
    }
    int firstidx = (multiple * i) - a;
    while (firstidx < arr.length) {
        arr[firstidx] = true;
        firstidx += i;
    }
}
```

Step5: Print all the primes from a to b where we need to use condition if isprime[i] = false && i + a != 1 then the no. is prime, where i+a is the val.

```
for (int i = 0; i < arr.length; i++) {
    if (arr[i] == false && (a + i) != 1) {
        sb.append(a + i + "\n");
    }
}
sb.append("\n");
System.out.println(sb);
```

4. Code:

```
// ~~~~~User Section~~~~~
public static ArrayList<Integer> sieveOfEratosthenes(int N) {
    boolean[] arr = new boolean[N + 1]; // false - prime || true - non prime
    for (int i = 2; i * i <= N; i++) {
        if (arr[i] == false) {
            for (int j = i * 2; j <= N; j += i) {
                arr[j] = true;
            }
        }
    }
    ArrayList<Integer> ans = new ArrayList<>();
    for (int i = 2; i <= N; i++) {
        if (arr[i] == false) {
            ans.add(i);
        }
    }
    return ans;
}

public static void segmentedSieveAlgo(int a, int b) {
    StringBuilder sb = new StringBuilder();
    int rootb = (int) Math.sqrt(b);
    ArrayList<Integer> primes = sieveOfEratosthenes(rootb);
    boolean[] arr = new boolean[b - a + 1];
    for (int i : primes) {
        int multiple = (int) Math.ceil(a * 1.0 / i);
        if (multiple == 1) {
            multiple++;
        }
        int firstidx = (multiple * i) - a;
        while (firstidx < arr.length) {
            arr[firstidx] = true;
            firstidx += i;
        }
    }
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == false && (a + i) != 1) {
            sb.append(a + i + "\n");
        }
    }
    sb.append("\n");
    System.out.println(sb);
}
```

5. Analysis:

Time complexity: $O(n \log(\log(n)))$, where $n = b - a$. Space complexity: $O(n)$, where $n = b - a$.

Author: Rohit Vishwakarma