2523. Closest Prime Numbers in Range

Given two positive integers `left` and `right`, find the two integers `num1` and `num2` such that:

- `left <= num1 < num2 <= right` .
- Both `num1` and `num2` are .
- `num2 - num1` is the **minimum** amongst all other pairs satisfying the above conditions.

Return the positive integer array `ans = [num1, num2]`. If there are multiple pairs satisfying these conditions, return the one with the **smallest** `num1` value. If no such numbers exist, return `[-1, -1]`.

Example 1:

```
Input: left = 10, right = 19
Output: [11,13]
Explanation: The prime numbers between 10 and 19 are 11, 13, 17, and 19.
The closest gap between any pair is 2, which can be achieved by [11,13] or [17,19].
Since 11 is smaller than 17, we return the first pair.
```

Example 2:

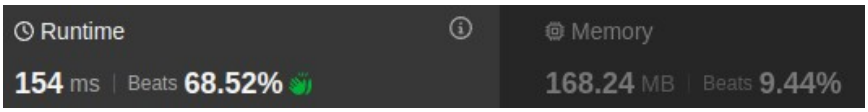
```
Input: left = 4, right = 6
Output: [-1,-1]
Explanation: There exists only one prime number in the given range, so the
conditions cannot be satisfied.
```

Constraints:

- $1 \leq \text{left} \leq \text{right} \leq 10^6$

2523. Closest Prime Numbers in Range

```
/*
   Sieve of Eratosthenes: store all primes numbers
   Time complexity: O(right . log(log(√right)) + (right − left) + m)
   Space complexity: O(right + m)
*/
```

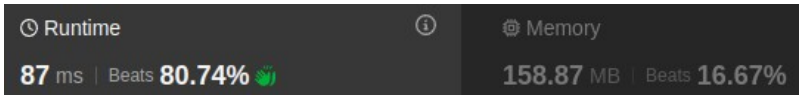



```
class Solution {
   public:
      std::vector<int> closestPrimes(int left, int right) {
         // Build Sieve of Eratosthenes's array
         // TC: O(right . log(log(√right)))
         // SC: O(right)
         std::vector<int> is_prime(right+1,true);
         is_prime[0]=is_prime[1]=false;
         for(int i=2;i*i<=right;++i){
            if(is_prime[i]){
               for(int j=i*i;j<=right;j+=i) is_prime[j]=false;
            }
         }

         // Store all primes numbers in range[left,right]
         // TC: O(right-left)
         // SC: O(m)
         std::vector<int> primes;
         for(int i=left;i<=right;++i){
            if(is_prime[i]) primes.push_back(i);
         }
```

```
        // Get the two closest prime numbers in range [left,right]
        // TC: O(m)
        // SC: O(1)

        // Get the primes size
        int m=primes.size();

        // if size<2, means there is no pair
        if(m<2) return {-1,-1};

        int diff=INT_MAX; // Track the minimum difference
        int prime1,prime2; // Track pair of the two closest primes numbers
        // Iterate all over primes numbers found in range[left,right]
        for(int i=0;i<m-1;++i){
            // If their difference is less than the minimum difference found so far
            if(primes[i+1]-primes[i]<diff){
                diff=primes[i+1]-primes[i]; // Minimize the difference

                // Update the answer
                prime1=primes[i];
                prime2=primes[i+1];
            }
        }

        return{prime1,prime2};
    }
};
```

2523. Closest Prime Numbers in Range

```
/*
   Sieve of Eratosthenes: Track pair of the two adjacent prime numbers
   Time complexity: O(right.log(log(√right))+(right−left))
   Space complexity: O(right)
*/
```




```
class Solution {
   public:
      std::vector<int> closestPrimes(int left, int right) {
         // Build Sieve of Eratosthenes's array
         // TC: O(right.log(log(√right)))
         // SC: O(right)
         std::vector<int> is_prime(right+1,true);
         is_prime[0]=is_prime[1]=false;
         for(int i=2;i*i<=right;++i){
            if(is_prime[i]){
               for(int j=i*i;j<=right;j+=i) is_prime[j]=false;
            }
         }
```

```
    // Get the two first smallest prime number
    // TC: O(right))
    // SC: O(1)

    int diff=INT_MAX; // To minimize the difference
    int prev_prime=-1,next_prime=-1; // Track the two adjacent primes numbers
    int prime1=-1,prime2=-1; // Track the two primes numbers with minimum difference
    for(int cur_num=left;cur_num<=right;++cur_num){
        // If current number is not prime, pass the next number
        if(!is_prime[cur_num]) continue;

        // If current number is prime:

        // If we still don't have the previous prime number and the
        // the current number will be the previous prime number
        if(prev_prime==-1) prev_prime=cur_num;

        // If we still don't have the next prime number that immediately
        // follows the previous prime number
        // the current number will be the next prime number
        else if(next_prime==-1) next_prime=cur_num;

        // Once we have both the two adjacent prime numbers
        if(next_prime!=-1 && prev_prime!=-1){

            // If difference between the two adjacent prime numbers is less or equal than 2
            // This is the minimum difference that we can have
            int d=next_prime-prev_prime;
            if(d<=2) return {prev_prime,next_prime};

            // If their difference is less than minimal difference found so far
            if(d<diff){
                diff=d; // Minimize the difference

                // Update the answer
                prime1=prev_prime;
                prime2=next_prime;
            }

            // Pass to the two next adjacent prime numbers
            prev_prime=next_prime;
            next_prime=-1;
        }
    }
    return {prime1,prime2};
}};
```