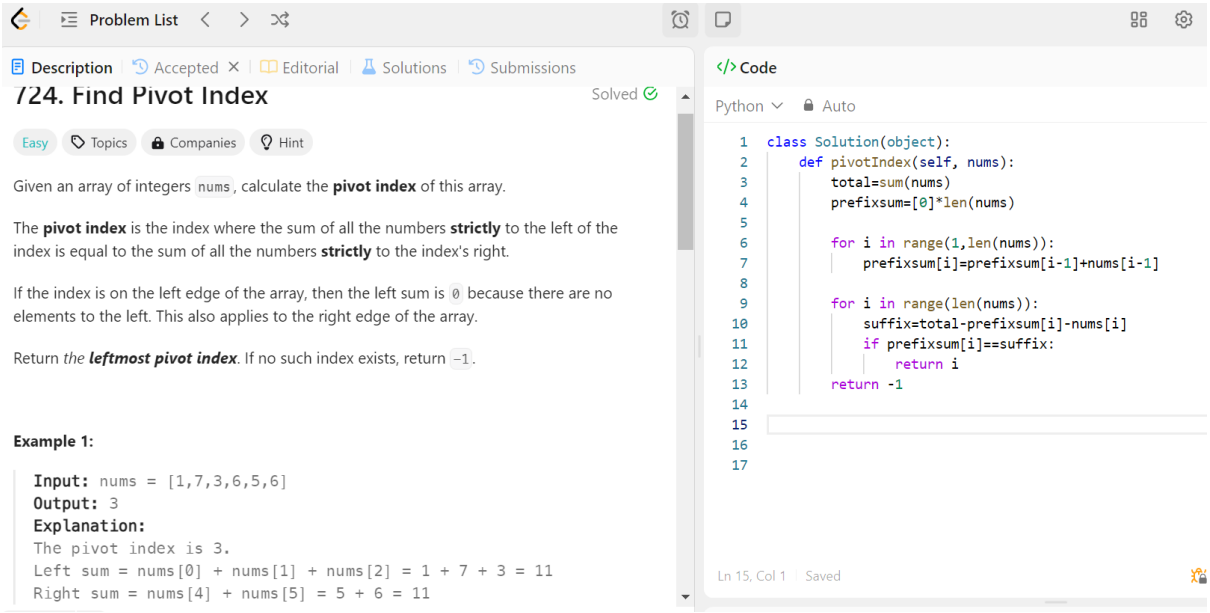
https://leetcode.com/problems/find-pivot-index/description/



Love Babbar 450 DSA Q)152
Why Quicksort is preferred for. Arrays and Merge Sort for Linked Lists ?

Quicksort is generally preferred for arrays, while Merge Sort is often favored for linked lists. This preference stems from the inherent characteristics of these data structures and how they interact with the sorting algorithms. Let's break down the reasons:

Quicksort for Arrays:

1. In-place sorting: Quicksort can be implemented to sort an array in-place, requiring only O(log n) additional space for the recursive call stack. This is memory-efficient for large arrays.
2. Cache efficiency: Quicksort's partitioning step works well with the cache, as it operates on contiguous memory locations in arrays.
3. Random access: Arrays allow O(1) random access, which is beneficial for Quicksort's partitioning step.
4. Adaptive: With good pivot selection strategies, Quicksort can adapt well to different input distributions.
5. Average-case performance: Quicksort has an average time complexity of O(n log n), which is often very fast in practice for arrays.

Merge Sort for Linked Lists:

1. Stable sorting: Merge Sort is a stable sort, which can be important for maintaining the relative order of equal elements in a linked list.
2. Predictable performance: Merge Sort always has O(n log n) time complexity, regardless of the input distribution. This consistency can be valuable in certain applications.

3. Sequential access: Linked lists are optimized for sequential access, which aligns well with Merge Sort's merging step.
4. No random access needed: Unlike Quicksort, Merge Sort doesn't require random access to elements, which is costly (O(n)) in linked lists.
5. Easier implementation: Merging linked lists is often simpler to implement than trying to partition a linked list for Quicksort.
6. Space efficiency: While Merge Sort typically requires O(n) extra space for arrays, it can be implemented with O(1) extra space for linked lists by re-linking nodes instead of creating new lists.

Why Quicksort is less ideal for Linked Lists:

1. Random access: Quicksort's partitioning step requires a lot of random access, which is O(n) for linked lists instead of O(1) for arrays.
2. Cache inefficiency: Linked list nodes may not be stored in contiguous memory locations, reducing cache efficiency.
3. Extra space: An in-place implementation of Quicksort for linked lists is more complex and less intuitive.

Why Merge Sort is less ideal for Arrays:

1. Extra space: A straightforward implementation of Merge Sort for arrays typically requires O(n) extra space, which can be significant for large arrays.
2. Cache efficiency: Merge Sort may not utilize cache as effectively as Quicksort for arrays, especially in its typical recursive implementation.

In practice, the choice between Quicksort and Merge Sort (or other sorting algorithms) depends on various factors including the size of the data, the data structure, memory constraints, stability requirements, and the specific characteristics of the input data. Modern sorting implementations often use hybrid approaches, combining the strengths of multiple algorithms.

When you're working with binary search problems, the decision of whether to update `right = mid` or `right = mid - 1` (or `left = mid + 1`) depends on the problem statement and the condition you're checking for.

Here are some general guidelines:

1. **If you're looking for the first/leftmost occurrence of a value:** In this case, when you find the target value, you need to keep searching on the left side of `mid` to find the leftmost occurrence. So, you would update `right = mid - 1`.
2. **If you're looking for the last/rightmost occurrence of a value:** In this case, when you find the target value, you need to keep searching on the right side of `mid` to find the rightmost occurrence. So, you would update `left = mid + 1`.
3. **If you're looking for the closest value that satisfies a condition:** In this case, you'll need to analyze the problem statement and the condition you're checking for. If the condition is satisfied at `mid`, you may need to update either `left` or `right` based on whether you want to find the maximum or minimum value that satisfies the condition.

- If you want to find the maximum value, update `left = mid + 1` when the condition is satisfied, so you can keep exploring the right side of the search space.
- If you want to find the minimum value, update `right = mid` when the condition is satisfied, so you can keep exploring the left side of the search space.

The decision of when to use `low < high` or `low <= high` as the termination condition for the binary search loop, and whether to return `low`, `high`, or `mid` at the end, depends on the specific problem you're trying to solve. However, there are some general guidelines you can follow:

1. **Termination condition: `low < high` or `low <= high`**
   - If you want to find the **first/leftmost** occurrence of a value or the **minimum** value that satisfies a condition, you typically use `low < high` as the termination condition.
   - If you want to find the **last/rightmost** occurrence of a value or the **maximum** value that satisfies a condition, you typically use `low <= high` as the termination condition.
2. **Return value: `low`, `high`, or `mid`**
   - If you're looking for the **first/leftmost** occurrence of a value or the **minimum** value that satisfies a condition, you usually return `low` at the end of the binary search.
   - If you're looking for the **last/rightmost** occurrence of a value or the **maximum** value that satisfies a condition, you usually return `high` at the end of the binary search.
   - In some cases, you might need to return `mid` if the problem requires finding the exact value or index that satisfies a specific condition.

Collect the AntiPrimes - Code Combat

Problem | Submissions | Leaderboard

An Anti-Prime number (AKA Highly composite number) is a natural number that has more factors than all the numbers before it.

Generate and return an array of the first 'n' antiprime /highly composite numbers

Some more context :

```
- the number 1 is the first anti-prime number
- 2 has 2 factors (1 and 2) , which is greater than 1's number of factors (1), so it is
antiprime
- 4 has 3 factors (1, 2 and 4) which is greater than 3 and 2 (each having 2 factors)
making it the next antiprime, and so on.
```

Input Format

the integer 'n' is passed to the function as input.

Constraints

returned array must be of size 'n'

Output Format

return an array of the first 'n' highly composite / antiprime numbers in the function definition.

Sample Input 0

```
5
```

Sample Output 0

```
1 2 4 6 12
```

Explanation 0

```
Factors of 1 : 1
2 : 1 2 4 : 1 2 4 6 : 1 2 3 6 12 : 1 2 3 4 6 12
```

Gives TLE error for one testcase-

```python
def GenerateAntiPrimes(count):
    # Write your code to return the array of composite numbers..
    def count_factors(num):
        count = 0
        for i in range(1, int(num**0.5) + 1):
            if num % i == 0:
                if i * i == num:
                    count += 1
                else:
                    count += 2
        return count

    antiprimes = [1]  # 1 is considered the first antiprime
    current_num = 2
    max_factors = 1

    while len(antiprimes) < n:
        factors = count_factors(current_num)
        if factors > max_factors:
            antiprimes.append(current_num)
            max_factors = factors
        current_num += 1

    return antiprimes
```

Coordinates in space - Code Combat

Problem | Submissions | Leaderboard | Discussions

You are stuck in a spaceship , and you can move the spaceship only among 2 directions x and y. You have to follow a set of rules from a manual to operate the spaceship Each rule consists of a command and a unit. 1) command - 3 types exist - commands : "j" , "k" and "l". 2) unit - units of distance to move

```
The details of each command
"l" - moves the spaceship by given units in x direction
"k" - moves the spaceship by ((given units)*-1) in y direction
"j" - moves the spaceship by given units in y direction
```

Each rule is separated by a ':' , for example : "l 5:j 5:l 8:k 3:j 8:l 2" is the set of rules. so the spaceship should act in the following way: l 5 (x position += 5)
j 5 (y position += 5) l 8 (x position += 5) k 3 (y position -= 3) j 8 (y position += 8) l 2 (x position += 2) Last Position : x = 15 , y = 10

Given the set of rules, Can you return the product of the x position and y position of the spaceship after all the rules are followed?

```
import sys



if __name__ == '__main__':
    # read input from STDIN and write output to STDOUT
    ip = sys.stdin.readline().strip()

    result = []
    for term in ip.split(':'):
        key, value = term.split()
        result.append((key, int(value)))

    x,y=0,0
    for t in result:
        if t[0]=='l':
            x+=t[1]
        elif t[0]=='k':
            y+=t[1]*(-1)
        else:
            y+=t[1]
    print(x*y)
```