Grokking the Coding Interview: Patterns for Coding Questions

67% completed

Search Course

Solution Review: Problem Challenge 2

Pattern: Two Heaps

- Introduction
- Find the Median of a Number Stream (medium)
- Sliding Window Median (hard)
- Maximize Capital (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Pattern: Subsets

- Introduction
- Subsets (easy)
- Subsets With Duplicates (easy)
- Permutations (medium)
- String Permutations by changing case (medium)
- Balanced Parentheses (hard)
- Unique Generalized Abbreviations (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Modified Binary Search

- Introduction
- Order-agnostic Binary Search (easy)
- Ceiling of a Number (medium)
- Next Letter (medium)
- Number Range (medium)
- Search in a Sorted Infinite Array (medium)
- Minimum Difference Element (medium)
- Bitonic Array Maximum (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Bitwise XOR

- Introduction
- Single Number (easy)
- Two Single Numbers (medium)
- Complement of Base 10 Number

Search in a Sorted Infinite Array (medium)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
  - Time complexity
  - Space complexity

Problem Statement

Given an infinite sorted array (or an array with unknown size), find if a given number 'key' is present in the array. Write a function to return the index of the 'key' if it is present in the array, otherwise return -1.

Since it is not possible to define an array with infinite (unknown) size, you will be provided with an interface `ArrayReader` to read elements of the array. ArrayReader.get(index) will return the number at index; if the array's size is smaller than the index, it will return `Integer.MAX_VALUE`.

Example 1:

```
Input: [4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30], key = 16
Output: 6
Explanation: The key is present at index '6' in the array.
```

Example 2:

```
Input: [4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30], key = 11
Output: -1
Explanation: The key is not present in the array.
```

Example 3:

```
Input: [1, 3, 8, 10, 15], key = 15
Output: 4
Explanation: The key is present at index '4' in the array.
```

Example 4:

```
Input: [1, 3, 8, 10, 15], key = 200
Output: -1
Explanation: The key is not present in the array.
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
class ArrayReader {

  constructor(arr) {
    this.arr = arr;
  }

  get(index) {
    if (index >= this.arr.length)
      return Number.MAX_SAFE_INTEGER;
    return this.arr[index]
  }
};


const search_in_infinite_array = function(reader, key) {
  // TODO: Write your code here
  return -1;
};

var reader = new ArrayReader([4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]);
console.log(search_in_infinite_array(reader, 16))
console.log(search_in_infinite_array(reader, 11))
reader = new ArrayReader([1, 3, 8, 10, 15])
console.log(search_in_infinite_array(reader, 15))
console.log(search_in_infinite_array(reader, 200))

```

RUN SAVE RESET

Solution

Complement of Base 10 Number (medium)
Problem Challenge 1
Solution Review: Problem Challenge 1

Pattern: Top 'K' Elements
Introduction
Top 'K' Numbers (easy)
Kth Smallest Number (easy)
'K' Closest Points to the Origin (easy)
Connect Ropes (easy)
Top 'K' Frequent Numbers (medium)
Frequency Sort (medium)
Kth Largest Number in a Stream (medium)
'K' Closest Numbers (medium)
Maximum Distinct Elements (medium)
Sum of Elements (medium)
Rearrange String (hard)
Problem Challenge 1
Solution Review: Problem Challenge 1
Problem Challenge 2
Solution Review: Problem Challenge 2
Problem Challenge 3
Solution Review: Problem Challenge 3

Pattern: K-way merge
Introduction
Merge K Sorted Lists (medium)
Kth Smallest Number in M Sorted Lists (Medium)
Kth Smallest Number in a Sorted Matrix (Hard)
Smallest Number Range (Hard)
Problem Challenge 1
Solution Review: Problem Challenge 1

Pattern : 0/1 Knapsack (Dynamic Programming)
Introduction
0/1 Knapsack (medium)
Equal Subset Sum Partition (medium)
Subset Sum (medium)
Minimum Subset Sum Difference (hard)
Problem Challenge 1
Solution Review: Problem Challenge 1
Problem Challenge 2
Solution Review: Problem Challenge 2

Pattern: Topological Sort (Graph)
Introduction
Topological Sort (medium)
Tasks Scheduling (medium)
Tasks Scheduling Order (medium)
All Tasks Scheduling Orders (hard)
Alien Dictionary (hard)
Problem Challenge 1
Solution Review: Problem

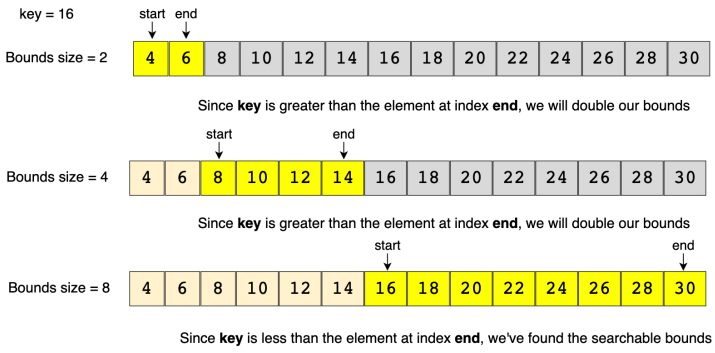
Explore
Tracks
My Courses
Edpresso
Refer a Friend

The problem follows the **Binary Search** pattern. Since Binary Search helps us find a number in a sorted array efficiently, we can use a modified version of the Binary Search to find the 'key' in an infinite sorted array.

The only issue with applying binary search in this problem is that we don't know the bounds of the array. To handle this situation, we will first find the proper bounds of the array where we can perform a binary search.

An efficient way to find the proper bounds is to start at the beginning of the array with the bound's size as '1' and exponentially increase the bound's size (i.e., double it) until we find the bounds that can have the key.

Consider Example-1 mentioned above:



Once we have searchable bounds we can apply the binary search.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
class ArrayReader {
  constructor(arr) {
    this.arr = arr;
  }

  get(index) {
    if (index >= this.arr.length) {
      return Infinity;
    }
    return this.arr[index];
  }
}


function search_in_infinite_array(reader, key) {
  // find the proper bounds first
  let start = 0;
  let end = 1;
  while (reader.get(end) < key) {
    newStart = end + 1;
    end += (end - start + 1) * 2;
    // increase to double the bounds size
    start = newStart;
  }

  return binary_search(reader, key, start, end);
}

```

RUN | SAVE | RESET

Close

Output 2.011s

```
6
-1
4
-1
```

Time complexity

There are two parts of the algorithm. In the first part, we keep increasing the bound's size exponentially (double it every time) while searching for the proper bounds. Therefore, this step will take $O(logN)$ assuming that the array will have maximum 'N' numbers. In the second step, we perform the binary search which will take $O(logN)$, so the overall time complexity of our algorithm will be $O(logN + logN)$ which is asymptotically equivalent to $O(logN)$.

Space complexity

The algorithm runs in constant space $O(1)$.

Challenge 1
Problem Challenge 2
Solution Review: Problem Challenge 2

Miscellaneous

Kth Smallest Number (hard)

Conclusions

Where to Go from Here

+ Create

Mark Course as Completed

applying to them. See how

Back

Number Range (medium)

Next

Minimum Difference Element (medium)

Mark as Completed