Search in a Sorted Infinite Array (medium)

We'll cover the following

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++

```python
import math


class ArrayReader:

  def __init__(self, arr):
    self.arr = arr

  def get(self, index):
    if index >= len(self.arr):
      return math.inf
    return self.arr[index]


def search_in_infinite_array(reader, key):
  # TODO: Write your code here
  return -1


def main():
  reader = ArrayReader([4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30])
  print(search_in_infinite_array(reader, 16))
  print(search_in_infinite_array(reader, 11))
  reader = ArrayReader([1, 3, 8, 10, 15])
  print(search_in_infinite_array(reader, 15))
  print(search_in_infinite_array(reader, 200))


main()
```

Run | Save | Reset

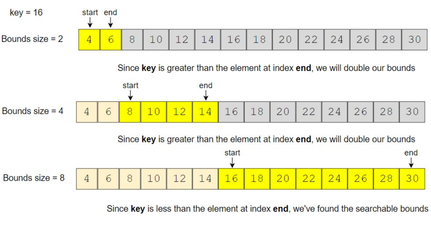
Solution

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

```python
import math


class ArrayReader:

  def __init__(self, arr):
    self.arr = arr

  def get(self, index):
    if index >= len(self.arr):
      return math.inf
    return self.arr[index]


def search_in_infinite_array(reader, key):
  # find the proper bounds first
  start, end = 0, 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)


def binary_search(reader, key, start, end):
  while start <= end:
    mid = start + (end - start) // 2
    if key < reader.get(mid):
      end = mid - 1
```

Run | Save | Reset

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 $O(1)$.