Data Structure

Lab Session #10: Searching

U Kang
Seoul National University

Goals

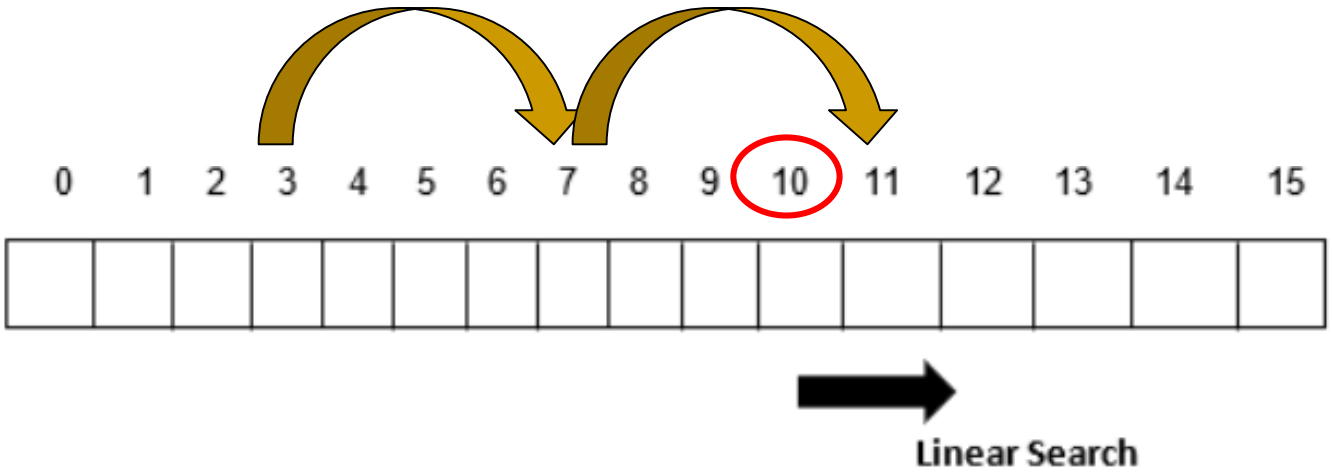
- Implement **"Jump Search" and "Binary Search"**
  - With counting the # comparisons!
    - Note. the # comparison only includes the comparison between the target value and the array value.
  - Compare the #comparisons of each algorithm.
- Optional "Interpolation Search"

Build a project

- Download the project for this lab from eTL.
- Extract the project, and import it using IntelliJ

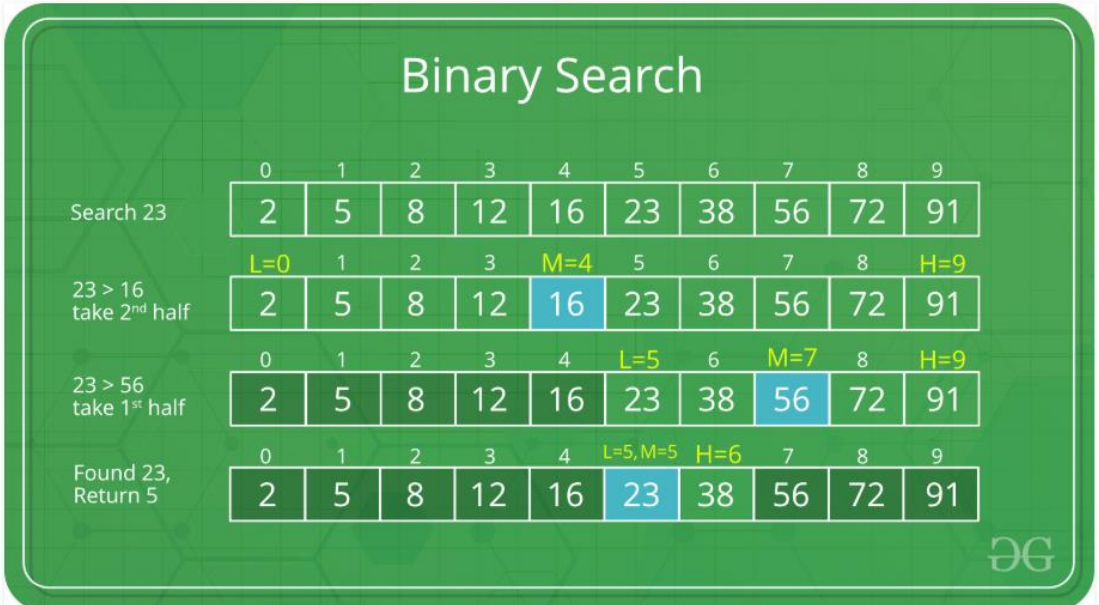
Jump Search

- Check every k'th element (L[k-1], L[2k-1]…)
  - Set k = (Array Length)^{½}
  - If the target is greater, then go on.
  - If the target is less, then use linear search on the k elements.
    - Assume, forward direction.



Binary Search

- Check every middle element
  - $mid = \frac{right - left}{2} + left$
  - If the target is greater, then ignore the left-half.
    - $left = mid + 1$
  - If the target is less, then ignore the right-half.
    - $right = mid - 1$



Assumptions

- There are no duplicate elements.
- We only count the comparison between the target and array elements.
- The array has at least 5 elements.
- Use optimal k for the jump search.
- Use forward linear search for the jump search.
- Use BigInteger for representing items.
  - new BigInteger("string of number")
  - BigInteger.compareTo(AnotherBig)
  - BigInteger.add(AnotherBig)
  - BigInteger.multiply(AnotherBig)

I/O Specification

- setItems

Input format	Output format
`Char type, int n, int a0, int d`	`None`
Description	
- **Set 'this.items' using arithmetic or geometric sequence.** - **If type is 'A', generate an arithmetic sequence with initial value a0, common difference d and length n.** - **If type is 'G', generate an geometric sequence with initial value a0, common ratio d and length n.** - **Print list of items by calling 'print' function at the end.**	
Example Input	**Example Output**
`setItems('a',10,1,1)`	`None`

I/O Specification

- doJumpSearch

Input format	Output format
`BigInteger target`	`int index`
Description	
- **Find the index of the target in the 'this.items' with counting the number of comparisons.** - **Print "[J] Index: (index), count: (count)" if target found.** - **Print "[J] Not found, count: (count)" if target not found.** - **Note that the target is a BigInteger.** - **Note that it use forward-linear search.**	
Example Input	**Example Output**
`doJumpSearch(3)`	`2`

I/O Specification

- doBinarySearch

Input format	Output format
`BigInteger target`	`int index`
Description	
- **Find the index of the target in the 'this.items' with counting the number of comparisons.** - **Print "[B] Index: (index), count: (count)" if target found.** - **Print "[B] Not found, count: (count)" if target not found.** - **Note that the target is a BigInteger.**	
Example Input	**Example Output**
`doBinarySearch(3)`	`2`

Sample Input & Output

<Input>

```
SET_ITEMS A 10 1 1
FIND_J 3
FIND_B 3
SET_ITEMS G 20 2 5
FIND_J 45
FIND_B 45
FIND_J 6250
FIND_B 6250
SET_ITEMS A 1000 1 2
FIND_J 4096
FIND_B 4096
FIND_J 1023
FIND_B 1023
```

<Output>

```
1 2 3 4 5 6 7 8 9 10
[J] Index: 2, count: 1
[B] Index: 2, count: 3
2 10 50 250 1250 6250 ……
[J] Not found, count: 4
[B] Not found, count: 4
[J] Index: 5, count: 4
[B] Index: 5, count: 4
1 3 5 7 9 11 13 15 17 ……
[J] Not found, count: 40
[B] Not found, count: 10
[J] Index: 511, count: 33
[B] Index: 511, count: 10
```

* These files are in the testcase folder!

OPTIONAL

Interpolation Search

- Can you implement the interpolation search and count the #comparison?
- Is interpolation search is always faster than binary search?
  - Find some cases that binary search is faster than interpolation search.

Questions?