Lab: Data Representation and Manipulation

This document defines the exercises for "Java Advanced" course @ Software University. Please submit your solutions (source code) of all below described problems in Judge.

Part I – Sorting and Searching Algorithms

1. Sorting

Implement the following sorting algorithms:

- **Bubble** Sort
- **Selection** Sort

Read a sequence of numbers from the console, **sort** them and **print** them back.

Examples

Input	Output
5 4 3 2 1	1 2 3 4 5
1 4 2 -1 0	-1 0 1 2 4

2. Searching

Implement the following searching algorithms:

- **Linear** Search
- **Binary** Search

Read a sequence of **sorted** numbers on the first line and a single number on the second from the console. **Find the index** of the number in the given array. Return **-1** if the element is **not present** in the array.

Examples

Input	Output
1 2 3 4 5 1	0
1 2 3 4 5 6	-1

Hints: Implement Binary Search

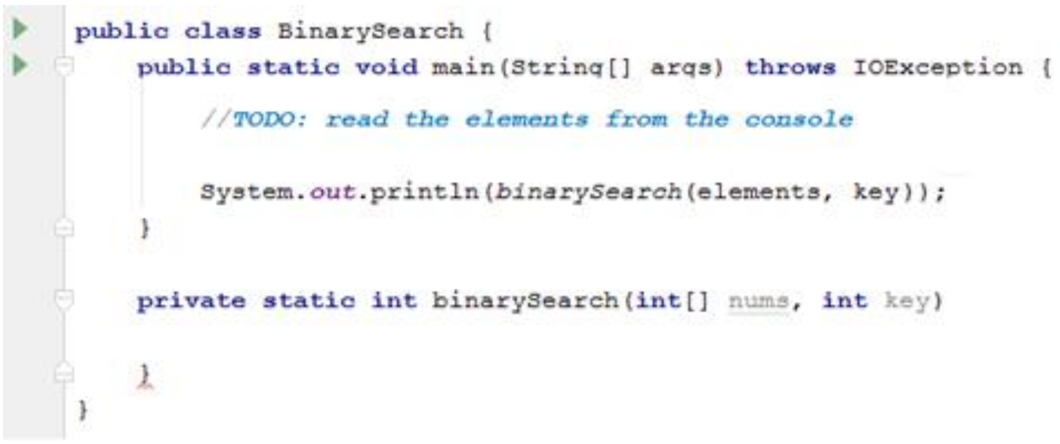
- First, if you're not familiar with the concept, read about binary search in Wikipedia.
- Here you can find a tool which shows visually how the search is performed.

In short, if we have a **sorted collection** of comparable elements, instead of doing linear search (which takes linear time), we can eliminate half the elements at each step.

We start at the middle of the collection, if we haven't found the element there, there are three possibilities:

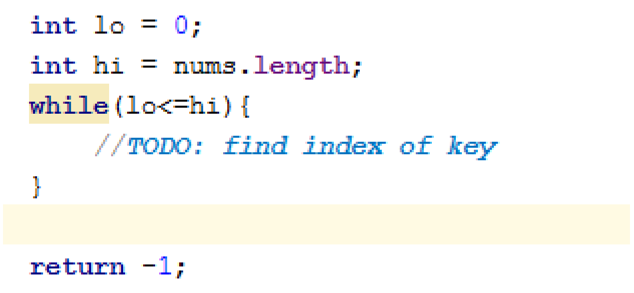
- The element we're looking for is **smaller** – then look to the **left** of the current element, we know all elements to the right are larger;
- The element we're looking for is **larger** – look to the **right** of the current element;
- The element is **not present**, traditionally return **-1** in that case.

Start by defining a class with a method:



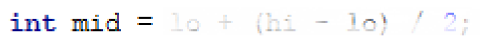
```java
public class BinarySearch {
    public static void main(String[] args) throws IOException {
        //TODO: read the elements from the console

        System.out.println(binarySearch(elements, key));
    }

    private static int binarySearch(int[] nums, int key)

    }
}
```

Inside the method, define two variables defining the bounds to be searched and a **while** loop:



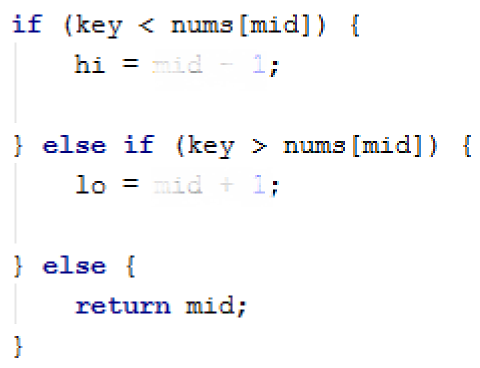
```java
int lo = 0;
int hi = nums.length;
while(lo<=hi){
    //TODO: find index of key
}

return -1;
```

Inside the while loop, we need to find the **midpoint**:



```java
int mid = lo + (hi - lo) / 2;
```

If the key is to the left of the midpoint, move the right bound. If the key is to the right of the midpoint, move the left bound:



```java
if (key < nums[mid]) {
    hi = mid - 1;

} else if (key > nums[mid]) {
    lo = mid + 1;

} else {
    return mid;
}
```

That's it! Good job!

Part II - Recursion

3. Recursive Array Sum

Write a program that finds the **sum** of all elements in an **integer array**. Use **recursion**.

Examples

Input	Output
`1 2 3 4`	`10`
`-1 0 1`	`0`

Hints

Write a **recursive** method. It will take as arguments the **input array** and the **current index**.

- The method should return the **current element** + the **sum of all next elements** (obtained by recursively calling it).
- The recursion should stop when there are no more elements in the array.

4. Recursive Factorial

Write a program that finds the **factorial** of a given number. Use **recursion**.

Examples

Input	Output
`5`	`120`
`10`	`3628800`

Hints

Write a **recursive** method. It will take as arguments an integer number.

- The method should return the **current element** * the **result of calculating factorial of current element - 1** (obtained by recursively calling it).
- The recursion should stop when there are no more elements in the array.

5. Recursive Drawing

Write a program that **draws** the figure below depending on **n**. Use **recursion**.

Examples

Input	Output

`2`	`**`
	`*`
	`#`
	`##`

5	`*****`
	`****`
	`***`
	`**`
	`*`
	`#`
	`##`
	`###`
	`####`
	`#####`

Hints

- Set the **bottom** of the recursion

```
if (n == 0) {
    return;
}
```

- Define **pre**- and **post**- recursive behavior

```
System.out.println(String.join( delimiter: "", Collections.nCopies(n,  o: "*")));
drawFigure( n: n - 1);
System.out.println(String.join( delimiter: "", Collections.nCopies(n,  o: "#")));
```