

Module Name - Searching & Sorting algorithms

Class - Day 1

Topic Name: Introduction to Searching

Instructor: Arun Kudiyal

Time Allocation Summary

Element	Slide numbers	Maximum time(min)
Spot Test		
Today's Agenda		
Introduction to Searching		
Total Time		120

Let's take a quick revision assessment of previously taught topics :-

- Algorithm analysis
- Time complexity
- Asymptotic notation
- Time complexities of Linear Search

Today's Agenda

- Deep-dive to Recursion
- Introduction to Binary Search
- Java Implementation of Binary Search
- Time Analysis of Binary Search

Recursion

Points of Discussion

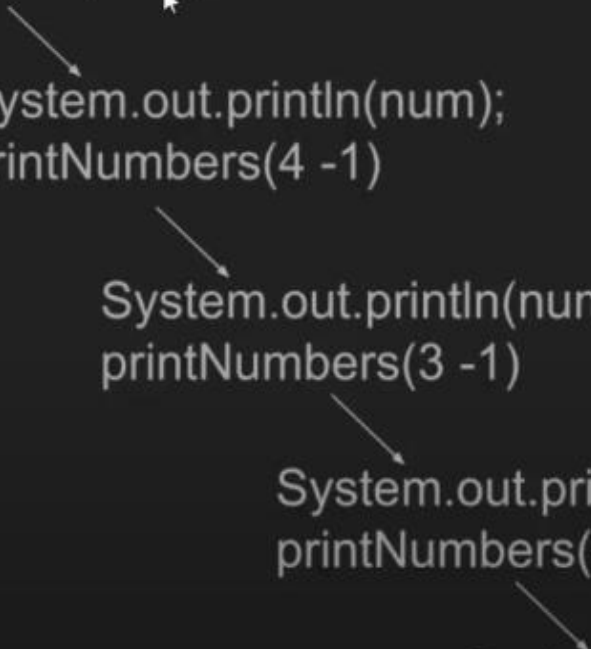
- What is the phenomenon of recursion?
- Why do we need recursion?
- Are there problems which are difficult to solve if we do not use recursion?
- How is memory affected while using recursion?
- Why does industries do not like the use of recursion in their product or services?
- What is a base condition?
- How bad can recursive code go?
- Is there any other alternative method than Recursion?

What is Recursion?

- Recursion means to define something in terms of itself.
- In programming, we use recursion to call the function by itself.
- Many data structures and functions can easily be coded when recursion is used in the code.
- Non availability of recursion can cause to complex solutions.
- Recursion is basically divided into two parts:
 - ✓ **Base Condition:** To stop the function when a specific condition is achieved or fulfilled.
 - ✓ **General Condition:** It calls the function itself on some specific criteria.

Example

```
// num is 4  
printNumbers(num)
```



```
System.out.println(num);  
printNumbers(4 -1)
```

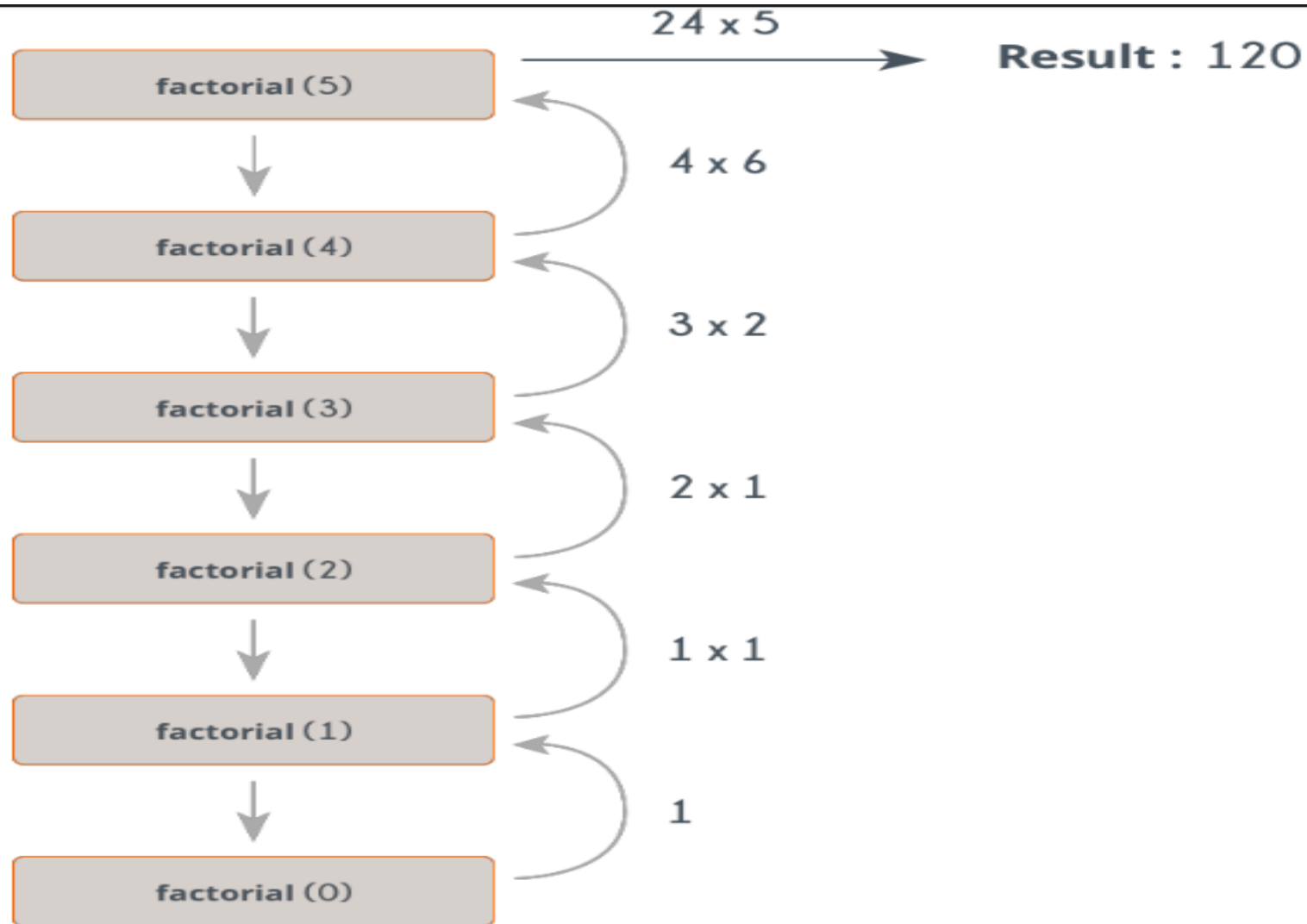
```
System.out.println(num);  
printNumbers(3 -1)
```

```
System.out.println(num);  
printNumbers(2 -1)
```

```
System.out.println(num);  
printNumbers(1 -1)
```

Output:

4
3
2
1



Exercises on Recursion

- Write numbers from n to 1.
- Find the power of a number (say num), to the degree (say deg).
- Fibonacci Series:- 0, 1, 1, 2, 3, 5, 8, 13, 21, , n

Binary Search

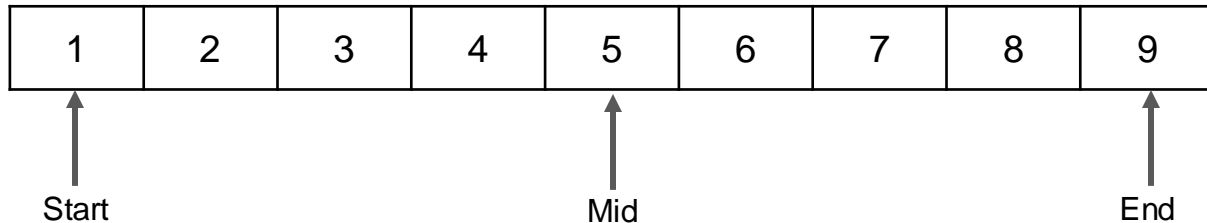
Note of caution!!!



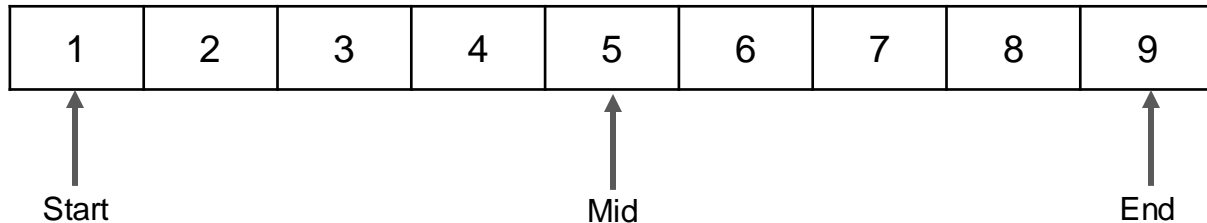
www.shutterstock.com · 1573398445

For binary search, you need a sorted array.

You cannot apply binary search on an array that's not sorted!



- Suppose you are searching for the element with the value 7.
- You start at the middle index and compare the element you're searching for with the element at the middle index.
- Here, the middle element is 5.
- Now, since 5 is not equal to our search element 7, we have to search more!



- All elements greater than 5 lie to the RIGHT of 5 in a sorted array, so...
- Since $7 > 5$, our search element 7 will lie to the RIGHT of 5
- This means that we can simply discard ALL element to the left of 5, and simply focus on the all the elements to the RIGHT of 5
- We now repeat a similar process on the remaining array

Steps in Binary Search:

- You start at the middle index and compare the element you're searching for with the element at the middle index.
- If the element at this index does not match the element you're searching for, you check if it is greater or lesser than the element at the index.
- If it is greater than the element at the middle index, you discard everything to the left and move to the right.
- You then find the middle of this array that's to the right, compare the required element with its middle, and keep doing this till you find the required element.

Let's see the code for Binary Search:

```
public int binarySearch(int[] inputArr, int key) {  
    int start = 0;  
    int end = inputArr.length - 1;  
    while (start <= end) {  
        int mid = (start + end) / 2;  
        if (key == inputArr[mid]) {  
            return mid;  
        }  
        if (key < inputArr[mid]) {  
            end = mid - 1;  
        } else {  
            start = mid + 1;  
        }  
    }  
    return -1;  
}
```


- The previous algorithm will work for MOST of the times, but not ALL.
- Can you find the bug in the code?
- Let's make it more easier. The bug is in this line:

```
int mid = (start + end) / 2;
```

- Can you find the situation where this statement will throw an error?
- What if the sum of `start` and `end` is higher than maximum int value?
- More importantly, how to resolve this bug?
- What if we replace it with the following statement?

```
int mid = low + ((high - low) / 2);
```



- Do you remember the example of a music playlist which we talked about earlier?
- Binary Search will make it much easier to find a song in an alphabetically sorted music playlist!



- What about searching for a specific book from a collection?
- Searching for a book from an alphabetically arranged shelf of books?
- Binary Search to the rescue!

- Let's analyze the time complexity of binary search now
- It's similar to the recursion problems we did in the last module, don't you think?
- So can you form the general equation for time complexity, $T(n)$ of the algorithm?

- $T(n) = 2 + T(n/2)$
- Using substitution k times, we can write the above equation as:
 $T(n) = 2k + T(n/2^k)$
- Now substitute $(n/2^k) = 1$ so that we can solve the equation in terms of $T(1)$
- Therefore, $T(n) = 2\log_2 n + T(1)$
- $T(n) = O(\log_2 n)$



Thank You!

Happy learning!