**[Java Data Structures & Algorithms + LEETCODE Exercises](https://capgemini.udemy.com/course/data-structures-and-algorithms-java/)**

DSA for Cracking the Coding Interview. Animated Examples for Faster Learning and Deeper Understanding.

**Here are the topics we will cover:**

**Technical**

* Big O notation

**Data Structures**

* Arrays
* Array Lists
* Linked Lists
* Doubly Linked Lists
* Stacks & Queues
* Binary Trees
* Hash Tables
* Heaps
* Graphs

**Algorithms**

Sorting

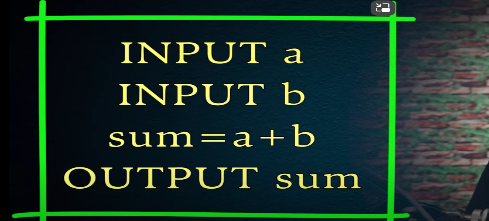
* Bubble Sort
* Selection Sort
* Insertion Sort
* Merge Sort
* Quick Sort

**Searching**

* Breadth First Search
* Depth First Search

**Telusko** **Data Structures and Algorithms (DSA) in Java 2024:**

Data structure is basically a way to organize and to store our data in the efficient way. There are multiple options available there.

The way we build an application is through Algorithms, it is a set of instructions.

We adding the value take two variables do a sum and sum give back to the user this are steps we are following, and this is **Algorithms**.

Advantage of DSA in organizations.

* It helps them to reduce the cost.
* To give the better customer experience. So that it will run faster

**ADT -> Abstract Data Type**

In software industry everything is about data, whatever we do we doing it for data

* Get the data from the user, process the data, also give output to the user, we will store the data into the database permanent storage. The thing is all about data.

**Abstract data type** - So, when we have a concept and the associated operations to it, off course with data which will be call them as abstract data type.

**Note**: We can create our own types, if we want to data inside that also we want to specify what operations we can perform that is our abstract data type.

**Arrays**: int [] numbs = {5, 6, 11, 9, 16, 14, 12};

This is the array we are inserting the new element at the end position that’s fine, but inserting in between this will take lot of time, this depends on number of elements after the position which we are adding.

Deleting in end is always welcome its not affecting our array in total. But we want to delete element in between this will difficult. It will take lot of time depends upon how many elements we have after that index value.

**Time Complexity:**

Algorithms means the real example if we want to cook something we have to certain procedures we follow some steps those steps and procedures are called **Algorithms**.

The thing is for one problem don’t have one solution we have multiple solutions, and we must pick the best one solution.

How will we choose best one?  
When we Run the application, we want our application to use less memory, or we want to take less time to execute or application and may be both are required. So, this we called Algorithms analysis.

Basically, we must analyze the algorithms so that we are make it efficient. Off course our first focus is product should work and we are thinking about optimizing it. We can optimize two ways are:

* Think about the space complexity

It means that use an algorithm which will take less memory.

* Time complexity.

It means that use an algorithm which will take less time.

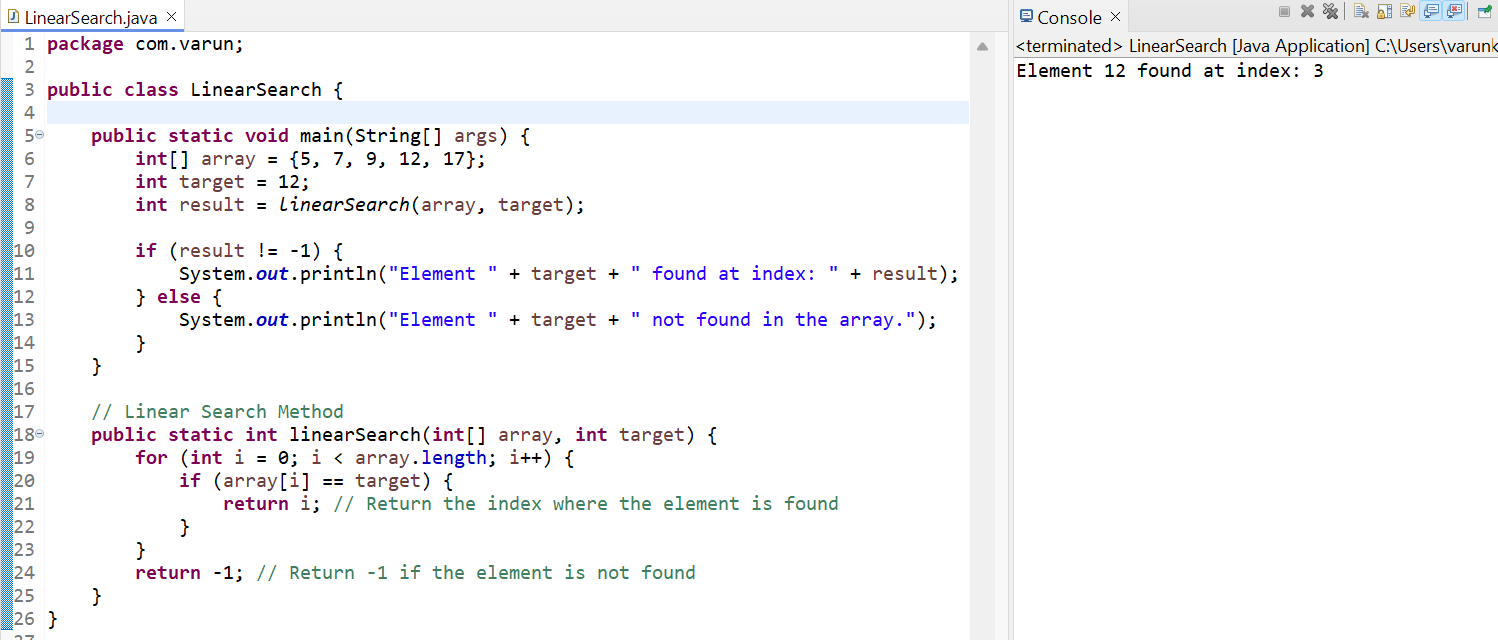
**Searching an element in a sorted Array.**

For this scenario we have multiple solutions best solutions are:

* 1. Linear search
  2. Binary search

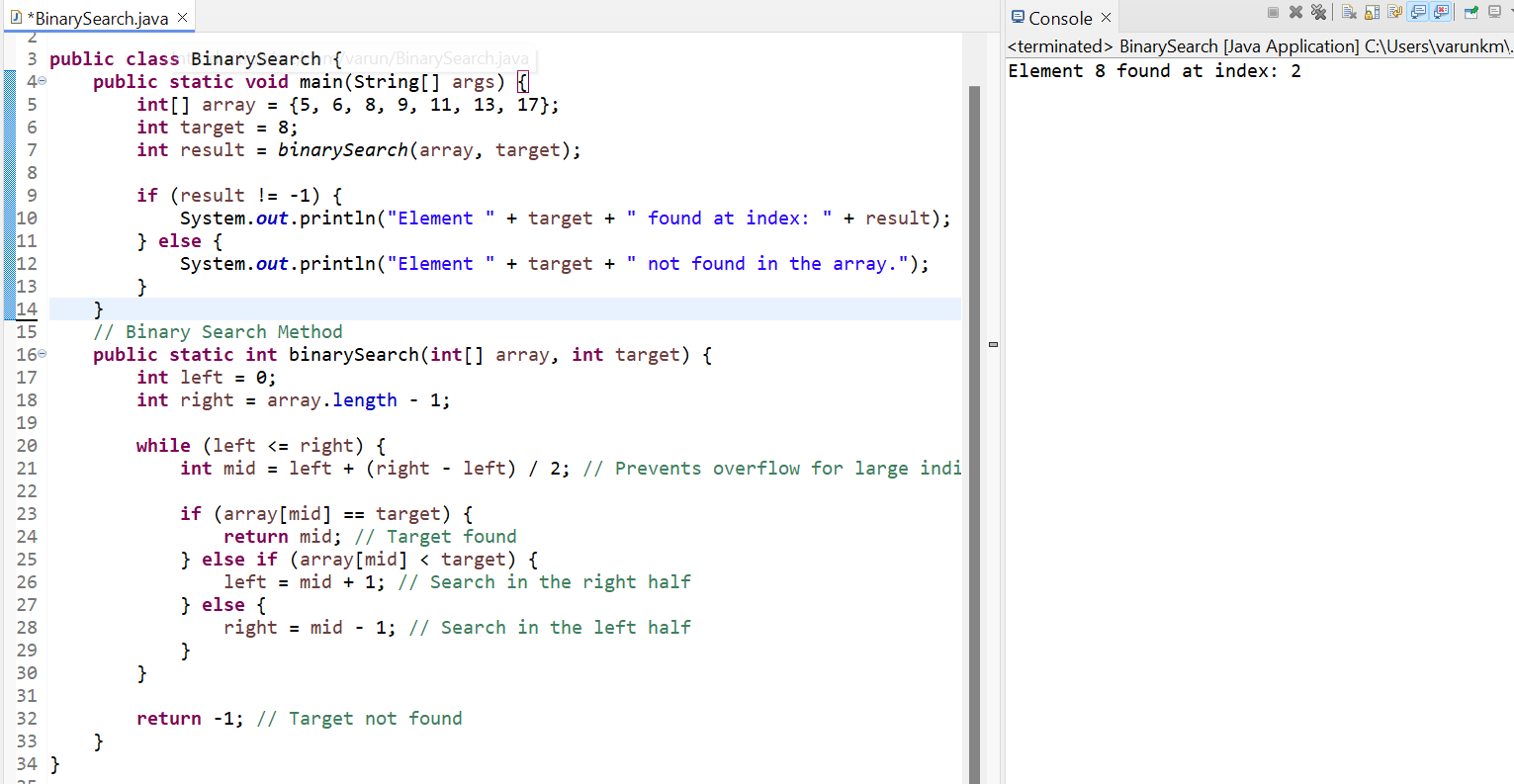
**Linear search:**

* In the sorted array I want to search exact element ex: we have array 5 7 9 12 17 in this we must search target is 12
* In linear search will go one by one first element will check if not there it will move next element. When got the value we can written the value.
* One problem is there what if the element is not there in the array.
* If the element is 5 it will take only one step. What if the element is end, steps depend upon the length of the array. The problem is when the array size is increases, number of steps also increases.



**Binary Search:**

* Binary Search is dividing our array into two parts that is the binary is how we do it. Give a starting and ending point.
* We find out the mid value using this formula **midValue = starting point + ending point / 2**
* We have an array **5 6 8 9 11 13 17** out **target value is 8** calculate this array using above formula 6/2 = 3 is the midValue here in position of 3 we have element 9.
* If the value which we got as a midValue which is less than or grater than to our target value.
* If the value is less than the value which we got as a midValue then the value after the midValue is not importance.
* Here 11 13 and 17 we directly skip them. Now we make end of the value is 9. Again, we have applied the above formula, and we check the midValue 3/2 = 1.5 we make it has 1 and we skip the 0 index value 5
* Now the starting point is element 6 and ending is 9 now again apply the formular 4/2 = 2 then finally we got the target value is 8.

****

In general, compare to linear search, Binary search works faster.

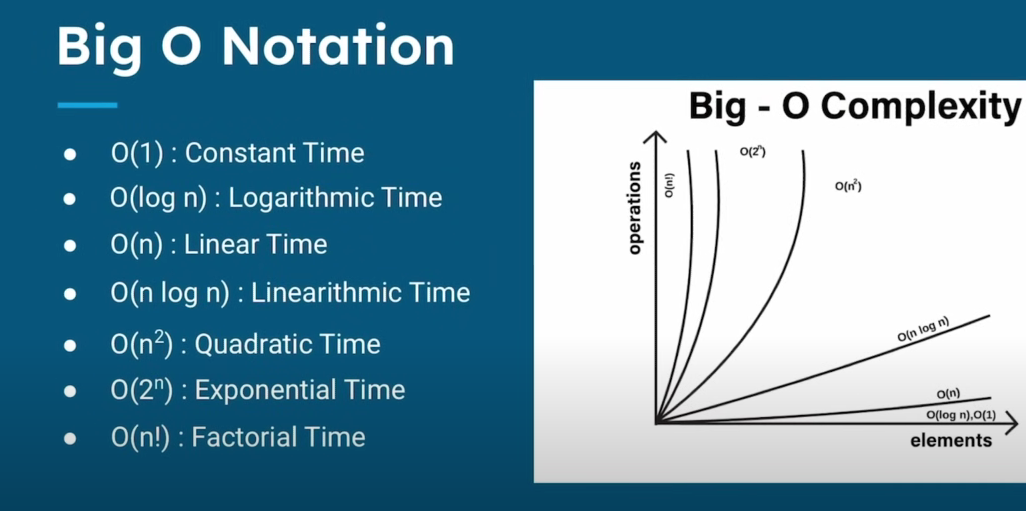
Time complexity measures of how the running time of an algorithm increases with the size of the input data.

Basically, when we are testing 10 or 20 values but what if the value goes up 1000 values how will affect our time will it increase a time in linear section. Example – let’s take 5 sec for 10 values, 10 sec for 20 values, will it take 1 million secs for 1 million values? How will we do this.

* + **Big O Notation:**

What happens in Big O notation is we use this concept to understand our time complexity of our algorithm. For example, your algorithm is not fast enaf: we must first find the Big O Notation. We represent like this below are:

* O(1) : Constant Time
* O(log n) : Logarithmic Time – This is a Binary search and it takes less time compare to linear search.
* O(n) : Linear Time – This is called Linear search
* O(n log n) : Linearithmic Time
* O(n^2) : Quadratic Time
* O(2^n) : Exponential Time
* O(n!) : Factorial Time



Practical implementation of linear and binary search to understand the time complexity which we can represents the Big O notation.

**public** **class** BothSearchSearch {

**public** **static** **void** main(String[] args) {

**int**[] nums = {5, 6, 8, 9, 11, 13, 17};

**int** target = 11;

**int** result1 = *linearSearch*(nums, target);

**int** result2 = *binarySearch*(nums, target);

**if** (result1 != -1) {

System.***out***.println("Element " + target + " found at index: " + result1);

} **else** {

System.***out***.println("Element " + target + " not found in the array.");

}

}

**public** **static** **int** linearSearch(**int**[] nums, **int** target) {

**int** steps = 0;

**for** (**int** i = 0; i < nums.length; i++) {

steps++;

**if** (nums[i] == target) {

System.***out***.println("Steps taken by linear "+ steps);

**return** i;

}

}

System.***out***.println("Steps taken by linear "+ steps);

**return** -1;

}

**public** **static** **int** binarySearch(**int**[] nums, **int** target) {

**int** steps = 0;

**int** left = 0;

**int** right = nums.length - 1;

**while** (left <= right) {

steps ++;

**int** mid = left + (right - left) / 2;

**if** (nums[mid] == target) {

System.***out***.println("Steps taken by Binary "+ steps);

**return** mid;

} **else** **if** (nums[mid] < target) {

left = mid + 1;

} **else** {

right = mid - 1;

}

}

System.***out***.println("Steps taken by Binary "+ steps);

**return** -1;

}

}

**Output**:

Steps taken by linear 5  
Steps taken by Binary 3  
Element 11 found at index: 4

In this above example binary search works better than the linear search.

**Sorting Techniques:**