**Command line arguments**

String[] args are basically command line arguments passed to the main function.

String[] args that are passed to the main function is basically the arguments that you can pass to the main function using command line (called as command line arguments) syntax goes like –

Java filename “args” “args” (while running the complied class file)

You can access these arguments in program using the args array like –

args[0]

**Input and output streams**

.out is basically reference variable for output stream and by default it is set to null and if outstream is having a null value then by default output stream is command line.

.in is basically reference variable for input stream and by default it is also set to null and if input stream is having a null value then by default input stream is keyboard.

**What are Streams in Java?**

A stream in Java is a continuous flow of data between a source and a destination. Streams are used to read data from an input source (like a file, keyboard, or network) and write data to an output destination (like a file, console, or network).

Java uses streams to handle input and output (I/O) operations efficiently. Streams process data sequentially, meaning data flows in a one-way direction.

**Primitive data types**

Primitive data types are basically data types which cannot be divided further or which you cannot break further.

For e.g. – string is not a primitive data type because we can further break it into char but char is a primitive data type because we cannot further divide this char into anything.

**Wrapper classes**

Primitive data types also have something called wrapper classes which provides additional functions to the primitive data types like the reference data types.

**Literals**

Literals are the syntactical representation of data values like char, integer, Boolean, etc.

**Reference variable**

Reference variable are just the name of the variable pointing to the data also called as identifiers.

**Shadowing**

Shadowing is a concept in programming where a variable declared in a certain scope (e.g., inside a function or block) has the same name as a variable in an outer scope, effectively "hiding" the outer variable within that inner scope.

How Shadowing Works

When a variable is shadowed, the inner variable takes precedence over the outer one within the inner scope.

The outer variable remains unchanged and is accessible outside the inner scope.

It is commonly seen in block-scoped or function-scoped languages.

**Variable length arguments –**

It is basically when you create a method that takes a variable number of arguments it is knows variable length arguments and it is also known as varArgs method.

Used when we don’t know how many arguments we are giving

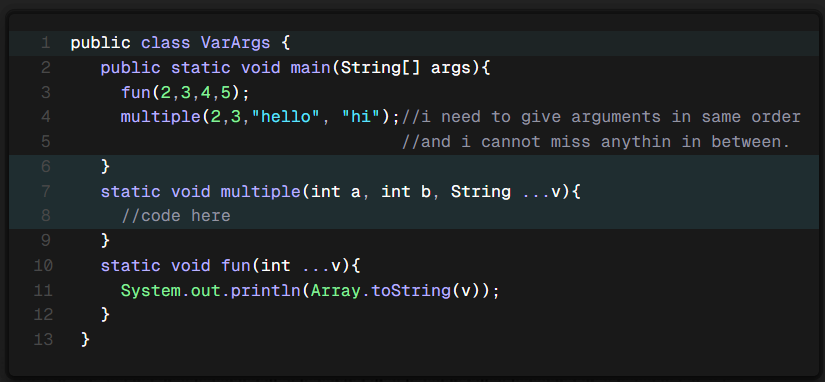
Syntax –

Static void fun(int …v){

}

This is internally taking it as a array of integers. And we can access it as an array with the name we put after the 3 dots.

You can do it for multiple data types like (String …v) or (char …v).



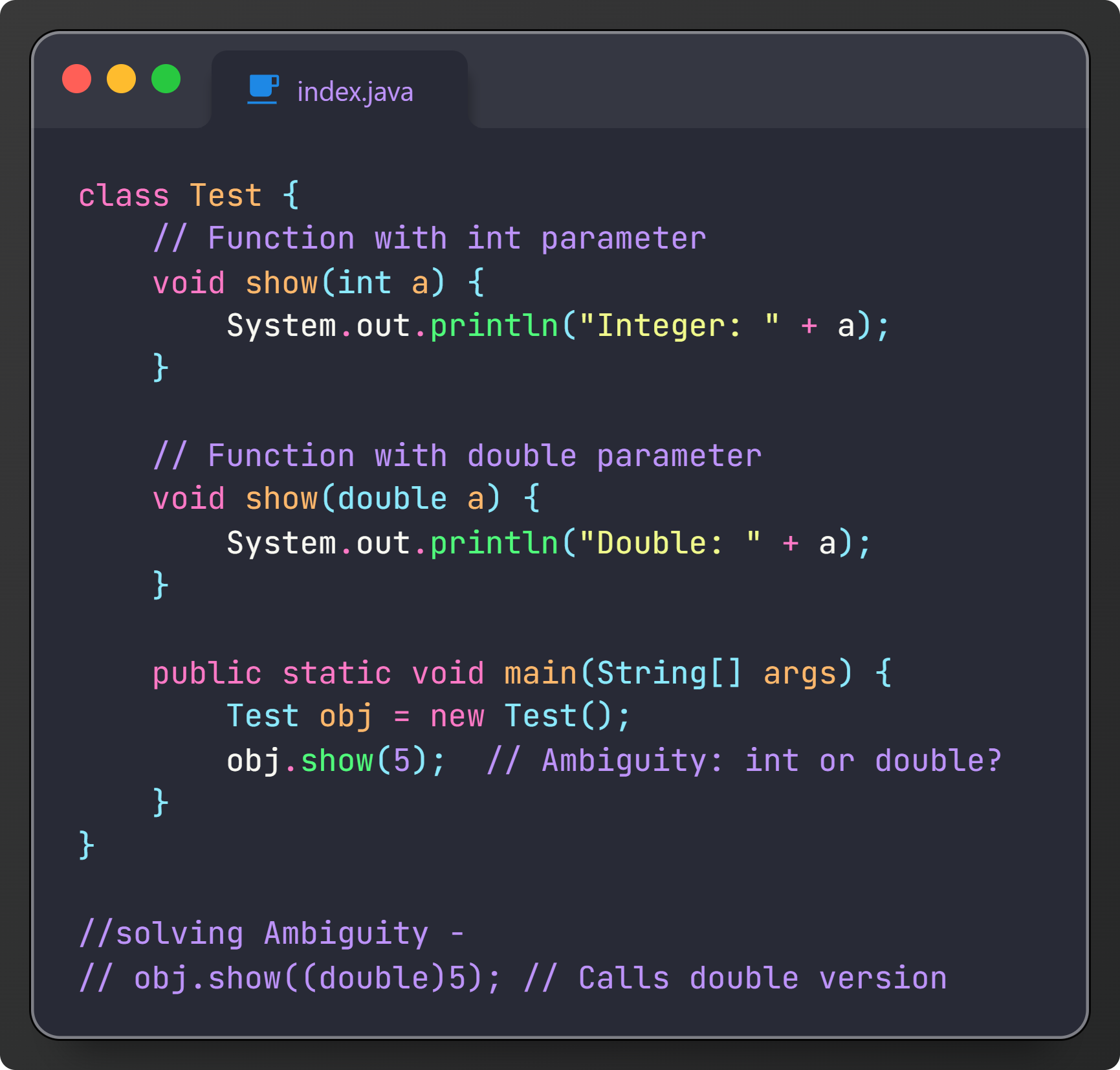
**Function overloading –**

Function Overloading is a feature in programming where multiple functions can have the same name but with different parameters (different number or types of parameters).

It allows better code readability and reusability.

Ambiguity –

**Ambiguity** refers to a situation where the compiler or interpreter cannot determine the correct interpretation of a statement, function call, or expression due to multiple possible matches. This leads to **compilation errors** or **unexpected behavior**.



**Arrays**

Int[] arr; //declaration of array. arr is getting defined in the stack. //happens at compile time

arr = new int[5]; // initialization: actually here object is being created in the heap. // happens at run time.

Array objects are in heap.

If we check out the Java language specification(JLS) they have mentioned that heap mentions are not continuous.

Heap Is the area from where memory for all the objects are created.

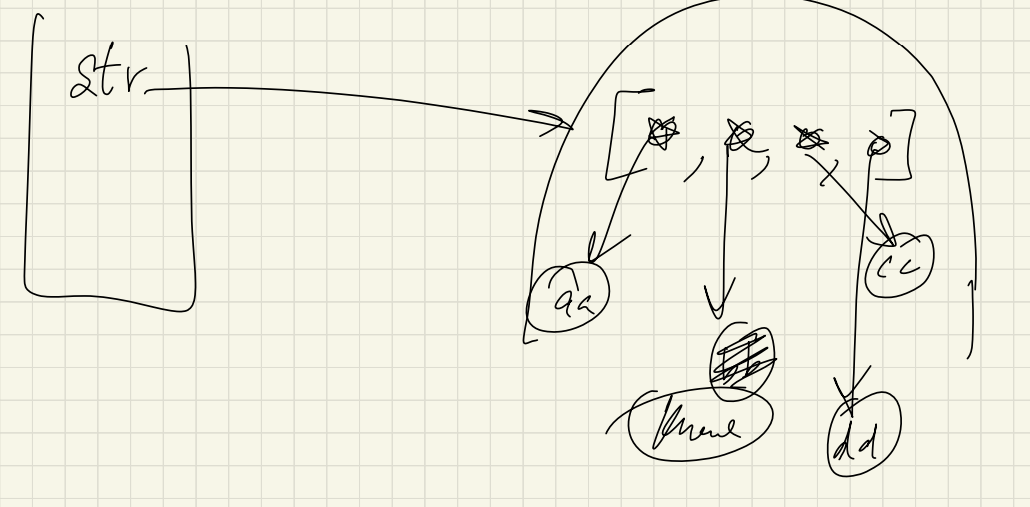
So in java arrays may not be contiguous.(unlike c and c++). // depends on jvm.

New is used to create an object in heap. Used for dynamic memory allocation.

In java initially all the array elements are initialized with 0 value for integer data type and null for string data type.

In java primitives are stored in stack only but all other objects are stored in heap.

So internally array is just a collection of reference variable pointing to an object, and every element in an array is a object itself, so every index of array is reference variable to that object. And we know that default value of non-primitive data type is null.



I cannot use for each loop for assigning value in a array.

**Null**

Null in java is a special data type in java which represents nothing. You cannot create a variable out of it but can you use it as a literal. (you cannot assign it to a primitive data type)

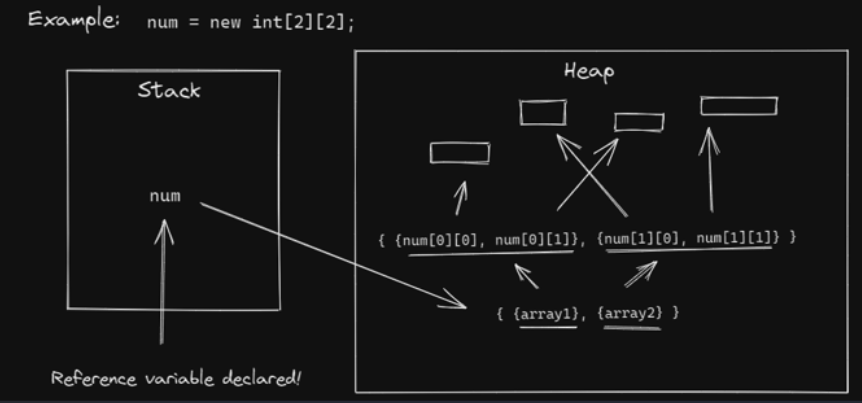
By default, all the non-primitive data types if not initialized have the value of null in an array.

**Multi dimensional Arrays –**

Multi dimensional arrays are arrays of arrays.

While declaring a multi dimensional array it is not mandatory to specify the columns but you need to specify the number of rows.

It is recommended to use till 3d arrays because it gets hard to visualize array after 3d array. (for higher dimension then that use python and nump like stuff).



**Array Lists (vectors in CPP) –**

**Why do we need array lists?**

Array lists are used when you don’t know the size of array and you need to be allocating it dynamically. Its like saying Hey, java please manage the size of array for me.

ArrayList<Integer> list = new ArrayList<>(5);

So basically in java we have implementation of most of the data structures by default in the java.utils package and we can directly use it as an object. (also we can create data structures by scratch).

We can directly print array list because it has its own toString methods that gets called on printing.

**List methods –**

**.get(index) –** used to get an item from index. // you cannot access items here like arrays by arr[0], you need to access it using method.

**.add() -**  to add an item in Array list

**.contains(x) –** to check weather the Array list contains x. return Boolean(true or false).

**.set(index, value)** – used to set value at a specific index. //can update value

**.remove(index) -**  used to remove value from an index.

**Internal working of array lists –**

Internally the size of list is fixed

Say array list fills by some amount (can be 50%, 80% or whatever)

It will create a new array list of say double the size

Copy old list elements in the new array list and the old array list is deleted from the memory. **//all of this you can check in the ArrayList source code.**

And all this is done in constant time complexity (O(1)).

You can also create multidimensional array list Syntax –

ArrayList<ArrayLIst<Integer>> list = new ArrayList<>();

**Linear search –**

Linear search is simple it basically says start looking for the element from start till you find the element.

Best case Time complexity is O(1) and worst case time complexity for linear search is O(n).

Two important functions you’ll be using with strings are

.toCharArray();

.charAt();

**Binary Search -**

Binary Search is an efficient searching algorithm used to find an element in a **sorted** array. It works by repeatedly dividing the search space in half until the target element is found or the search space is empty.

**Algorithm Steps:**

1. Start with two pointers:
   * low (beginning of the array)
   * high (end of the array).
2. Find the middle index: mid=low+high2mid = \frac{low + high}{2}mid=2low+high​
3. Compare the middle element with the target:
   * If arr[mid] == target, return the index mid.
   * If arr[mid] < target, search in the **right half** (low = mid + 1).
   * If arr[mid] > target, search in the **left half** (high = mid - 1).
4. Repeat until low is greater than high (element not found).

**Time Complexity:**

* **Best Case:** O(1)O(1)O(1) (if the element is found at the first comparison)
* **Average & Worst Case:** O(log⁡n)O(\log n)O(logn) (because we divide the search space by half each time)

**Advantages:**

* Faster than linear search for large datasets.
* Works well for searching in sorted data.

**Disadvantages:**

* Requires a sorted array.
* Not efficient for small datasets where linear search can be simpler.

**Sorting Algorithms –**

1. **Bubble sort algorithm**

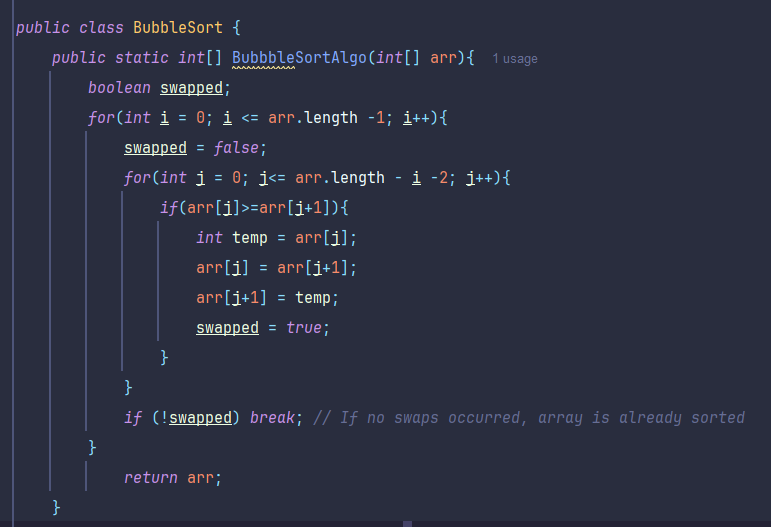
Bubble Sort is a simple comparison-based sorting algorithm that repeatedly swaps adjacent elements if they are in the wrong order. It continues until the entire array is sorted.

Algorithm Steps:

1. Start from the first element and compare it with the next one.
2. If the first element is greater than the second, swap them.
3. Move to the next pair and repeat step 2.
4. Repeat this process for all elements.
5. The largest element bubbles up to the last position in each pass.
6. Reduce the range and repeat until the array is fully sorted.

Time Complexity:

| Case | Time Complexity |
| --- | --- |
| Best Case (Already Sorted) | O(n) |
| Average Case | O(n²) |
| Worst Case (Reverse Sorted) | O(n²) |

****

1. **Selection Sort –**

Selection Sort Algorithm

Selection Sort is a simple comparison-based sorting algorithm that repeatedly finds the smallest element from the unsorted part and moves it to the correct position.

Algorithm Steps:

1. Start from the first element and assume it is the minimum.
2. Compare it with the rest of the array to find the smallest element.
3. Swap the smallest element with the first element.
4. Move to the next element and repeat the process for the remaining unsorted part.
5. Repeat until the entire array is sorted.

Time Complexity:

| Case | Time Complexity |
| --- | --- |
| Best Case (Already Sorted) | O(n²) |
| Average Case | O(n²) |
| Worst Case (Reverse Sorted) | O(n²) |

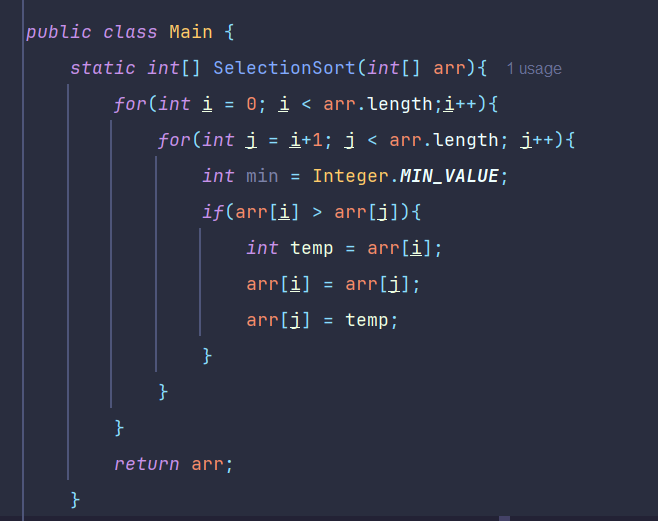
✅ Space Complexity: O(1) (In-place sorting)  
✅ Stable? ❌ (By default, not stable, but can be made stable with modifications)

Advantages:

✔️ Simple and easy to implement  
✔️ Does not require extra memory (O(1) space complexity, in-place sorting)  
✔️ Performs well for small datasets

Disadvantages:

❌ Slow for large datasets (O(n²) complexity)  
❌ Not stable (can be modified to make it stable)  
❌ Does more **swaps than Bubble Sort in the worst case**

****

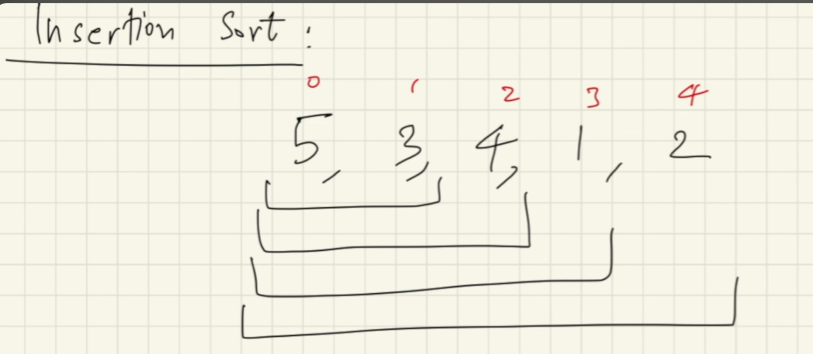
**Insertion Sort –**

Insertion Sort Algorithm

Insertion Sort is a simple comparison-based sorting algorithm that builds the sorted array one element at a time by inserting each element into its correct position.

Algorithm Steps:

1. Start with the second element (assuming the first element is sorted).
2. Compare it with the elements before it.
3. If it's smaller, shift the larger elements to the right.
4. Insert the element at its correct position.
5. Repeat the process for all elements.



Time Complexity:

| Case | Time Complexity |
| --- | --- |
| Best Case (Already Sorted) | O(n) |
| Average Case | O(n²) |
| Worst Case (Reverse Sorted) | O(n²) |

✅ Space Complexity: O(1) (In-place sorting)  
✅ Stable? ✅ Yes (Preserves the relative order of equal elements)

Advantages:

✔️ Efficient for small datasets  
✔️ Stable sorting algorithm  
✔️ Works well for nearly sorted data (O(n) in best case)  
✔️ In-place sorting (O(1) extra space)

Disadvantages:

❌ Slow for large datasets (O(n²) complexity in worst case)  
❌ Not suitable for large-scale sorting problems

**Things you can do while coding –**

While writing a code you can add edge cases like what if the array size is zero or the values provided are incorrect