Lesson 8

17.02.2025

```
public class Task1 {
    public static void main(String[] args) {
        String str = " Java ";
        System.out.println("Символ на позиції 0: '" + str.charAt(0) + "'");
    }
}
```

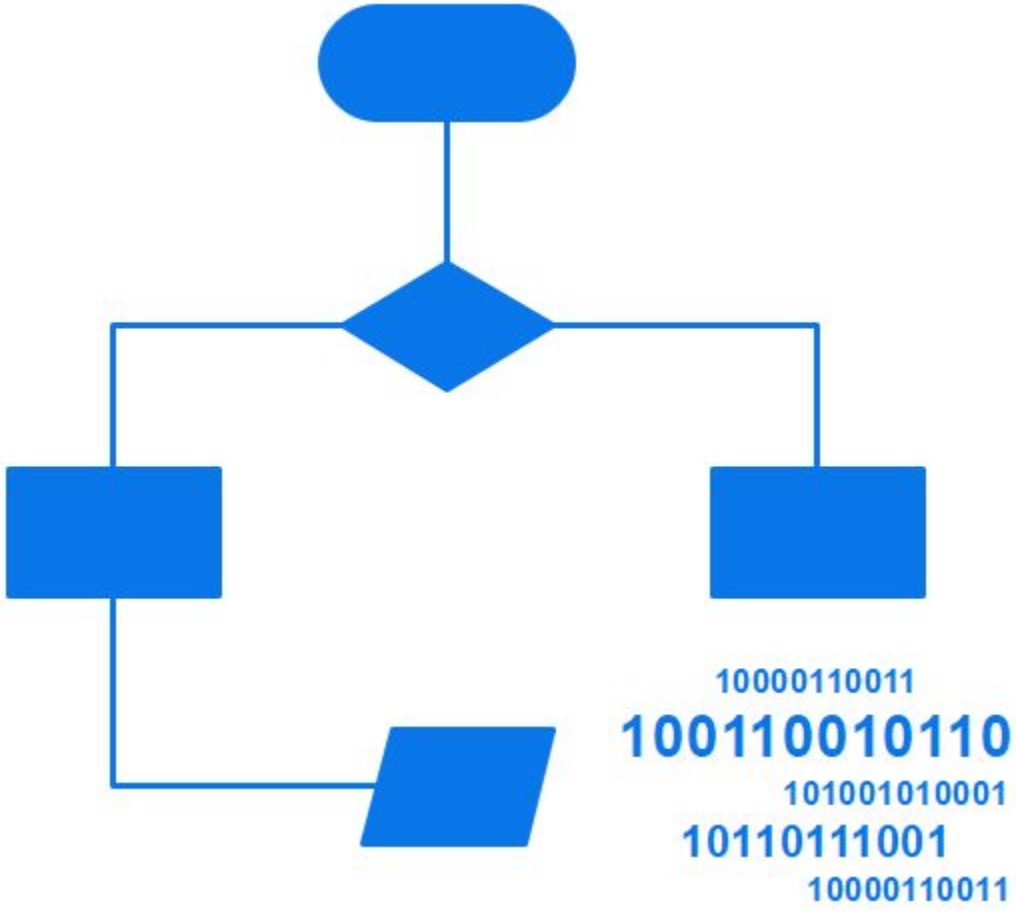
```
public class Task2 {
    public static void main(String[] args) {
        String s1 = "hello";
        String s2 = "he" + "llo";
        System.out.println(s1 == s2);
    }
}
```

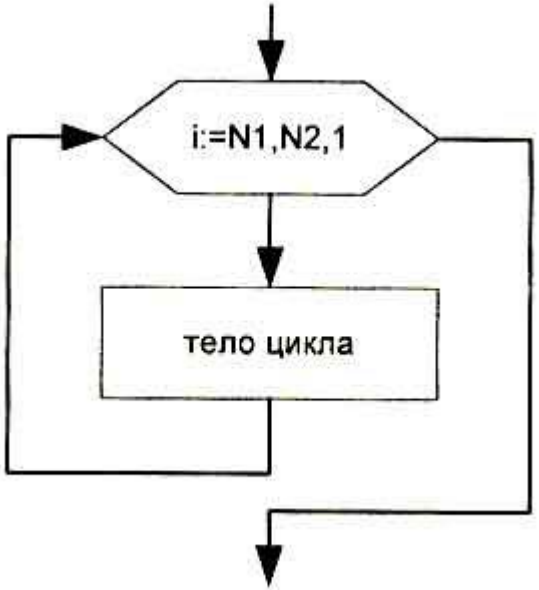
```
public class Task3 {
    public static void main(String[] args) {
        String str = "banana";
        str.replace("a", "o");
        System.out.println(str);
    }
}
```

```
public class Task4 {
    public static void main(String[] args) {
        String str = "abcdef";
        str.substring(2, 4);
        System.out.println(str);
    }
}
```

```
public class Task5 {
    public static void main(String[] args) {
        String str = "apple,,banana";
        String[] parts = str.split(",");
        System.out.println(parts.length);
    }
}
```

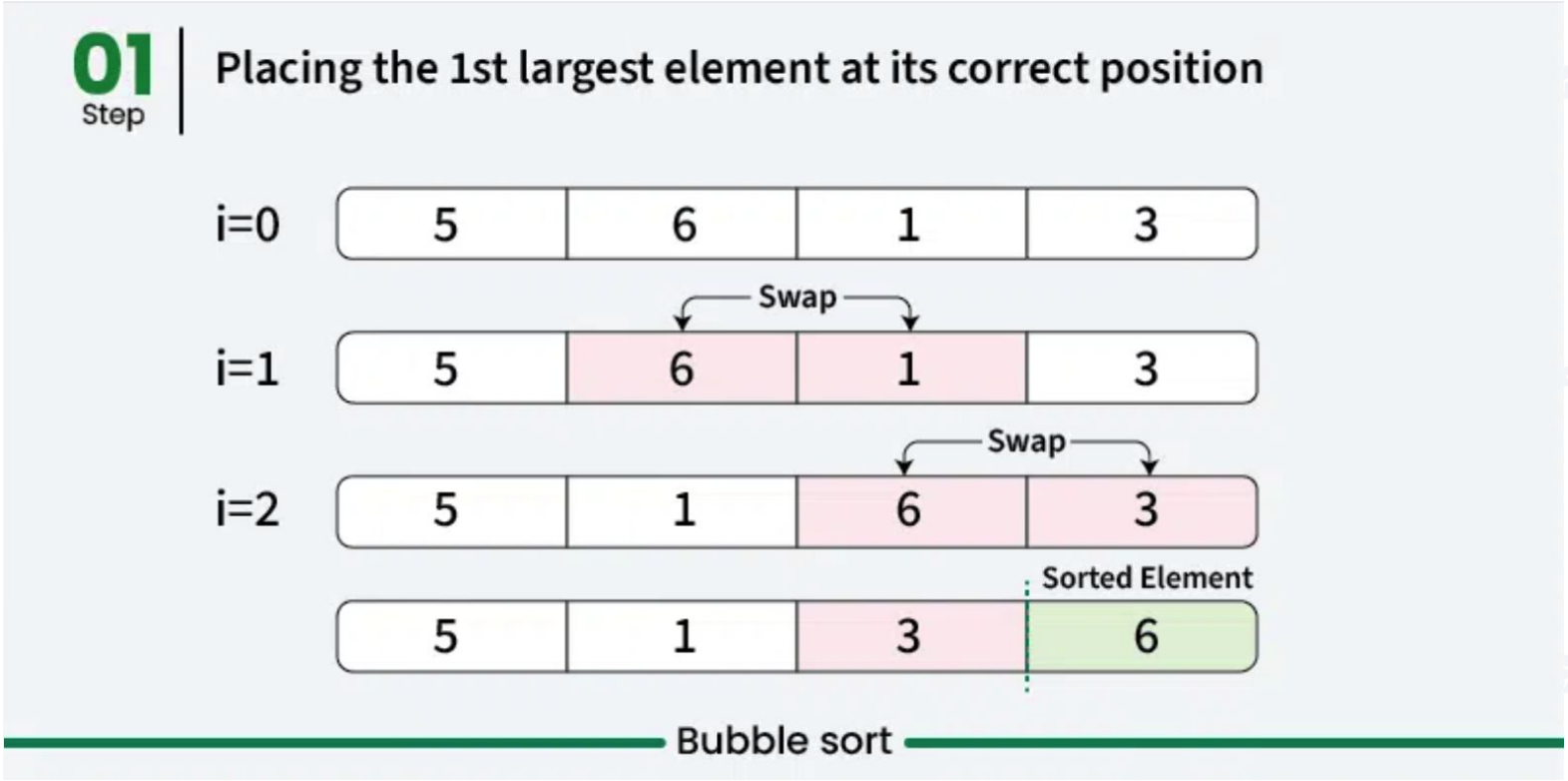
```java
public class Task6 {
    public static void main(String[] args) {
        String s1 = new String("java");
        String s2 = s1.intern();
        String s3 = "java";

        System.out.println(s1 == s2);
        System.out.println(s2 == s3);
    }
}
```


10000110011
100110010110
101001010001
10110111001
10000110011

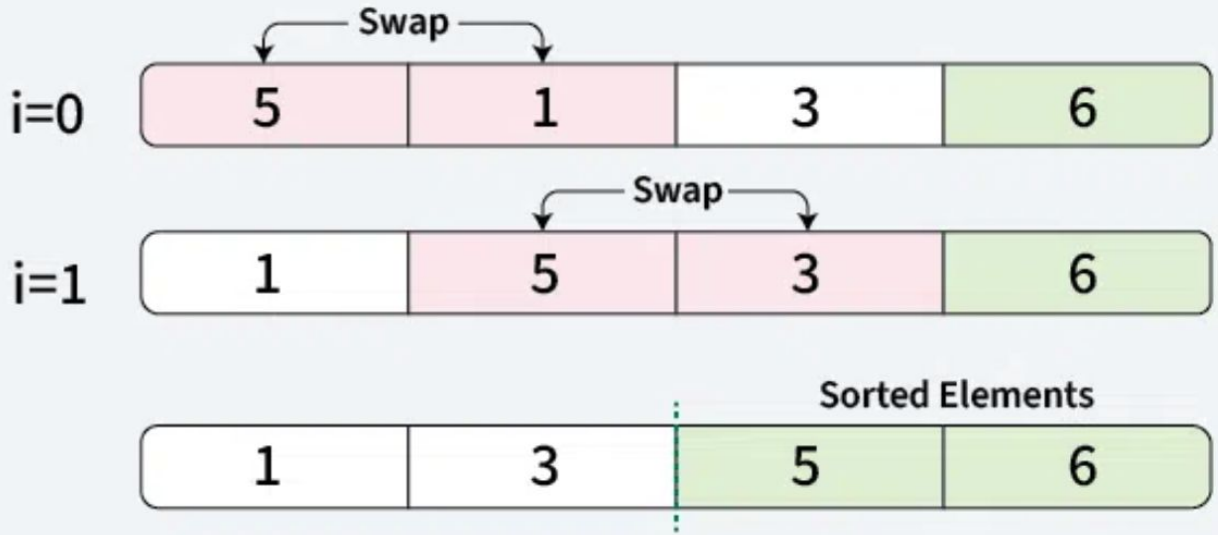

i:=N1,N2,1
тело цикла


условие
повтора
нет
да
тело цикла

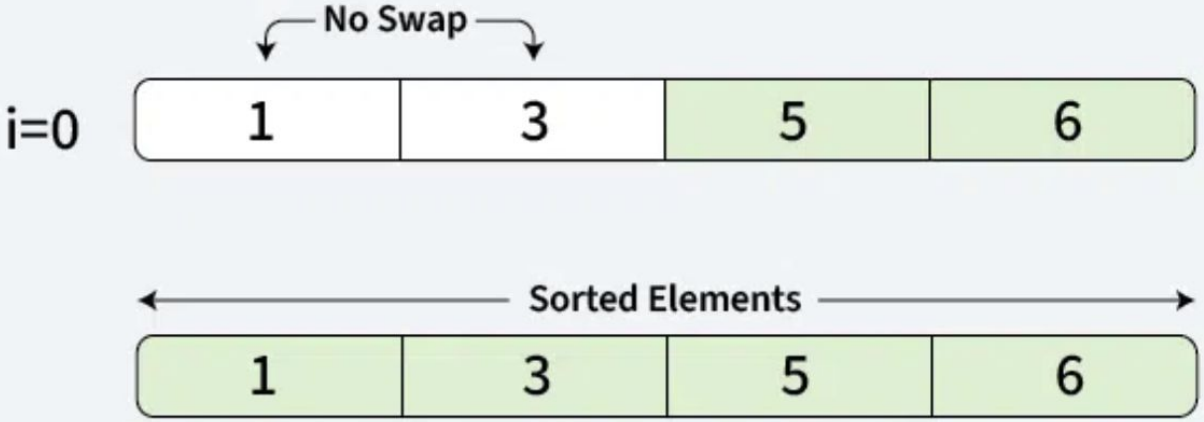
Bubble Sort Algorithm



02 Step | Placing 2nd largest element at its correct position

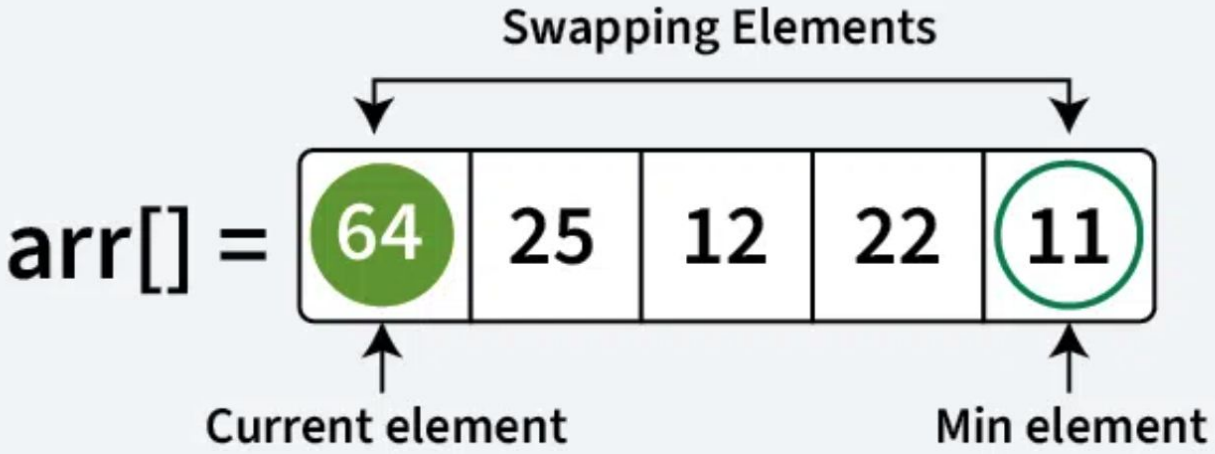


03 Step | Placing 3rd largest element at its correct position



Selection Sort

01 Step | Start from the first element at index 0, find the smallest element in the rest of the array which is unsorted, and swap (11) with current element(64).

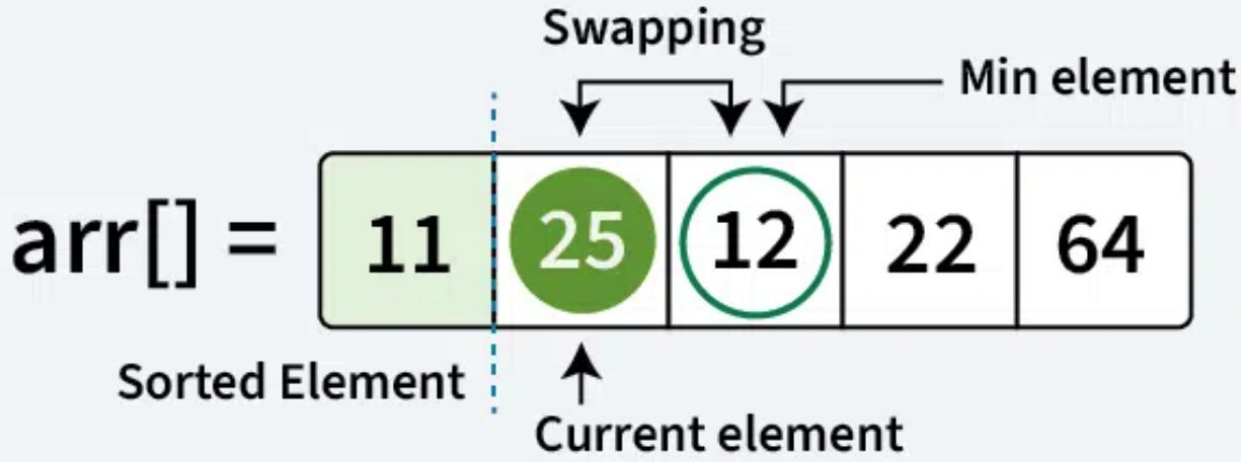


Selection Sort Algorithm

Selection Sort

02 Step

Move to the next element at index 1 (25). Find the smallest in unsorted subarray, and swap (12) with current element (25).

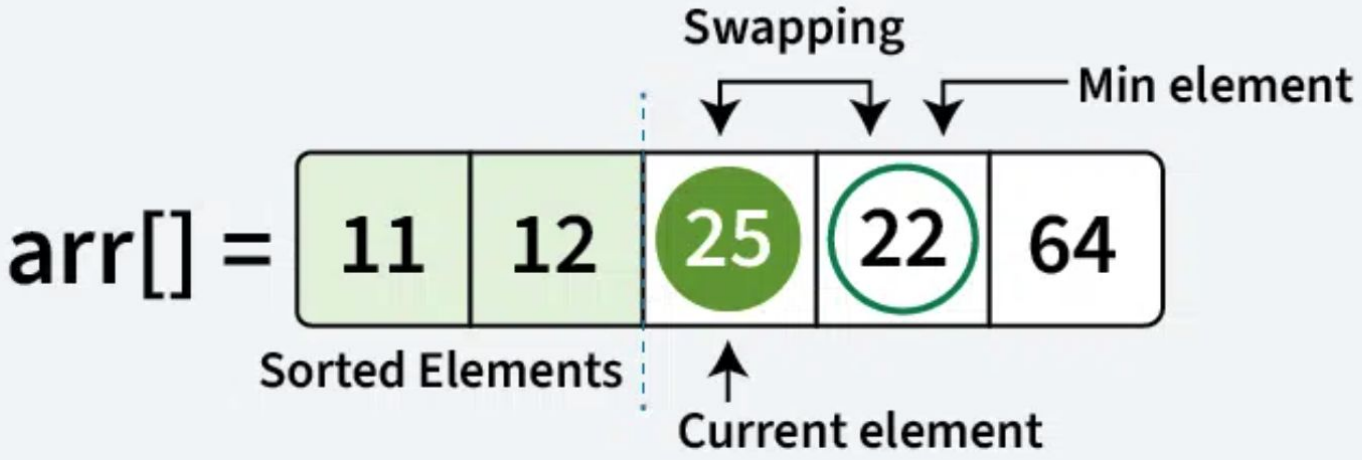


Selection Sort Algorithm

Selection Sort

03 Step

Move to element at index 2 (25). Find the minimum element from unsorted subarray, Swap (22) with current element (25).

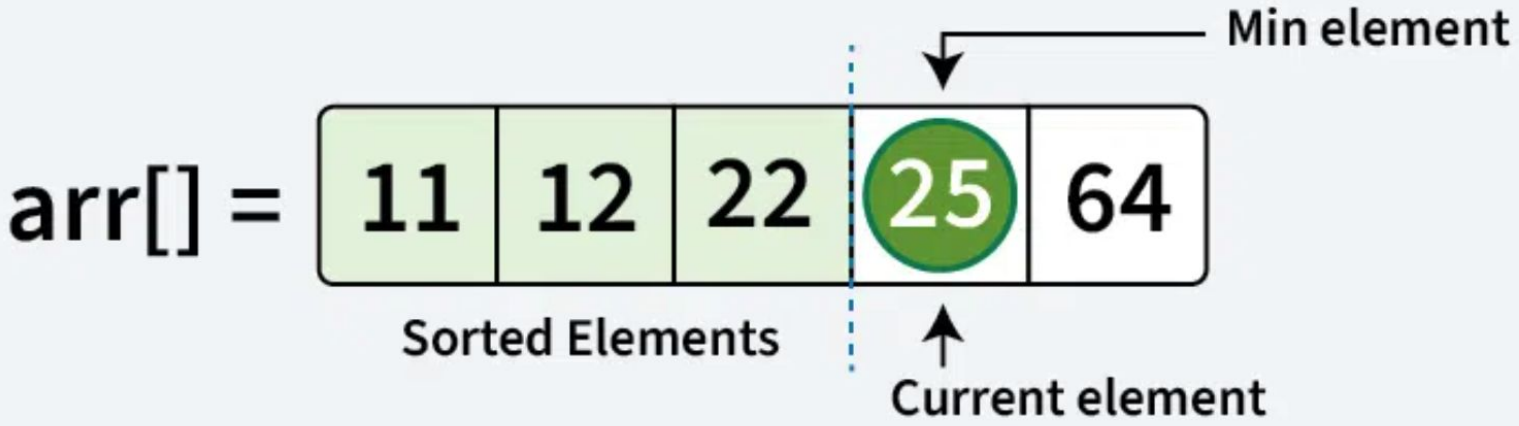


Selection Sort Algorithm

Selection Sort

04 Step

Move to element at index 3 (25), find the minimum from unsorted subarray and swap (25) with current element (25).

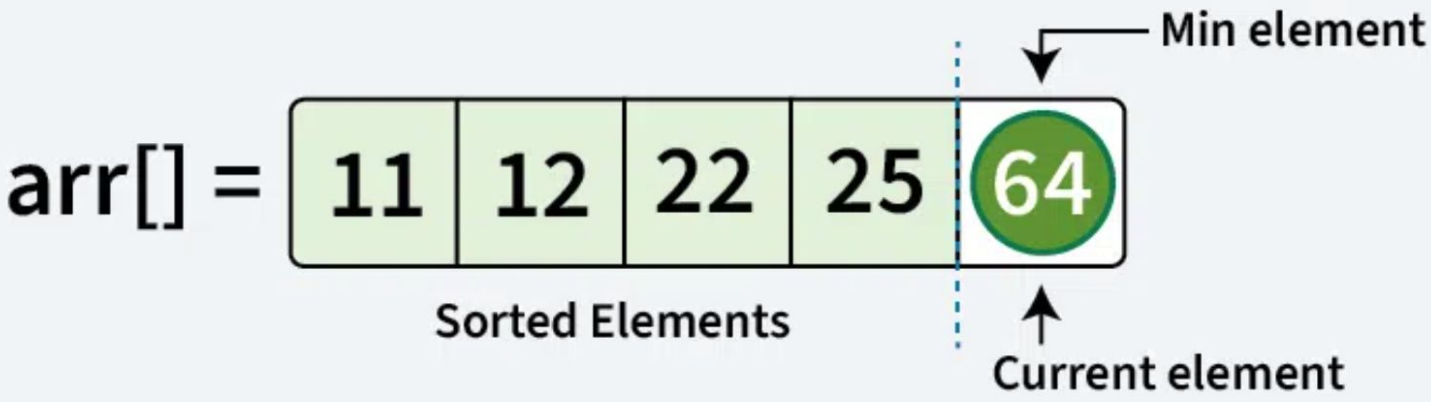


Selection Sort Algorithm

Selection Sort

05 Step

Move to element at index 4 (64), find the minimum from unsorted subarray and swap (64) with current element (64).

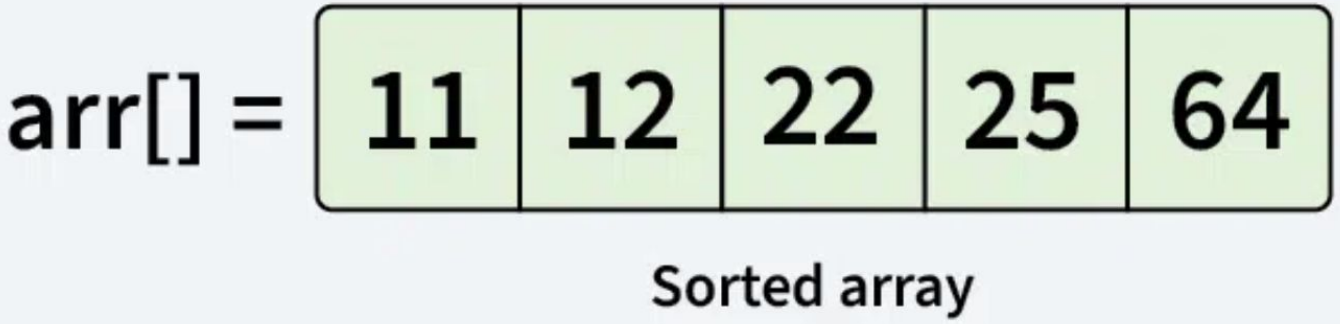


Selection Sort Algorithm

Selection Sort

06 Step

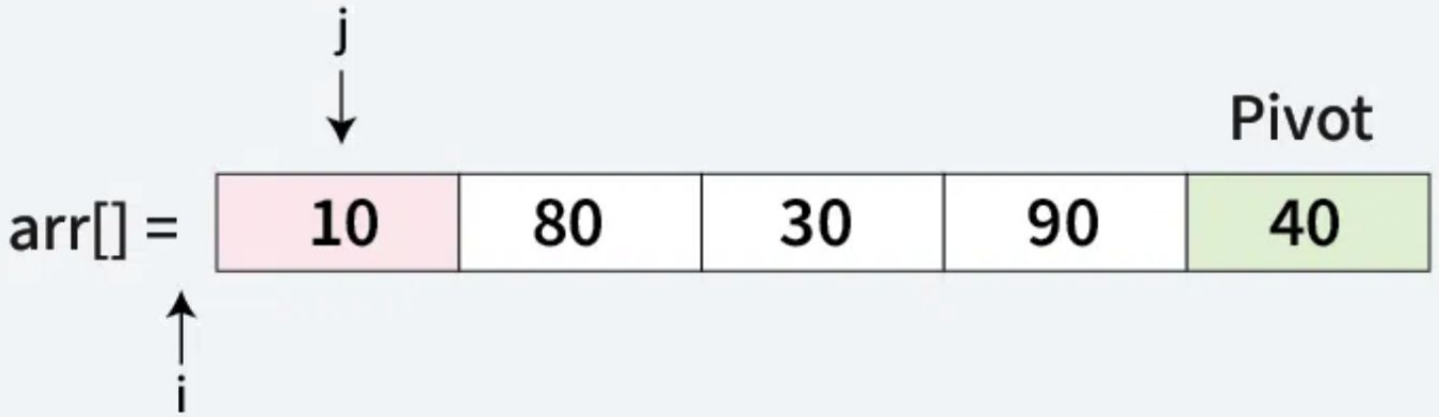
We get the sorted array at the end.



Quick Sort

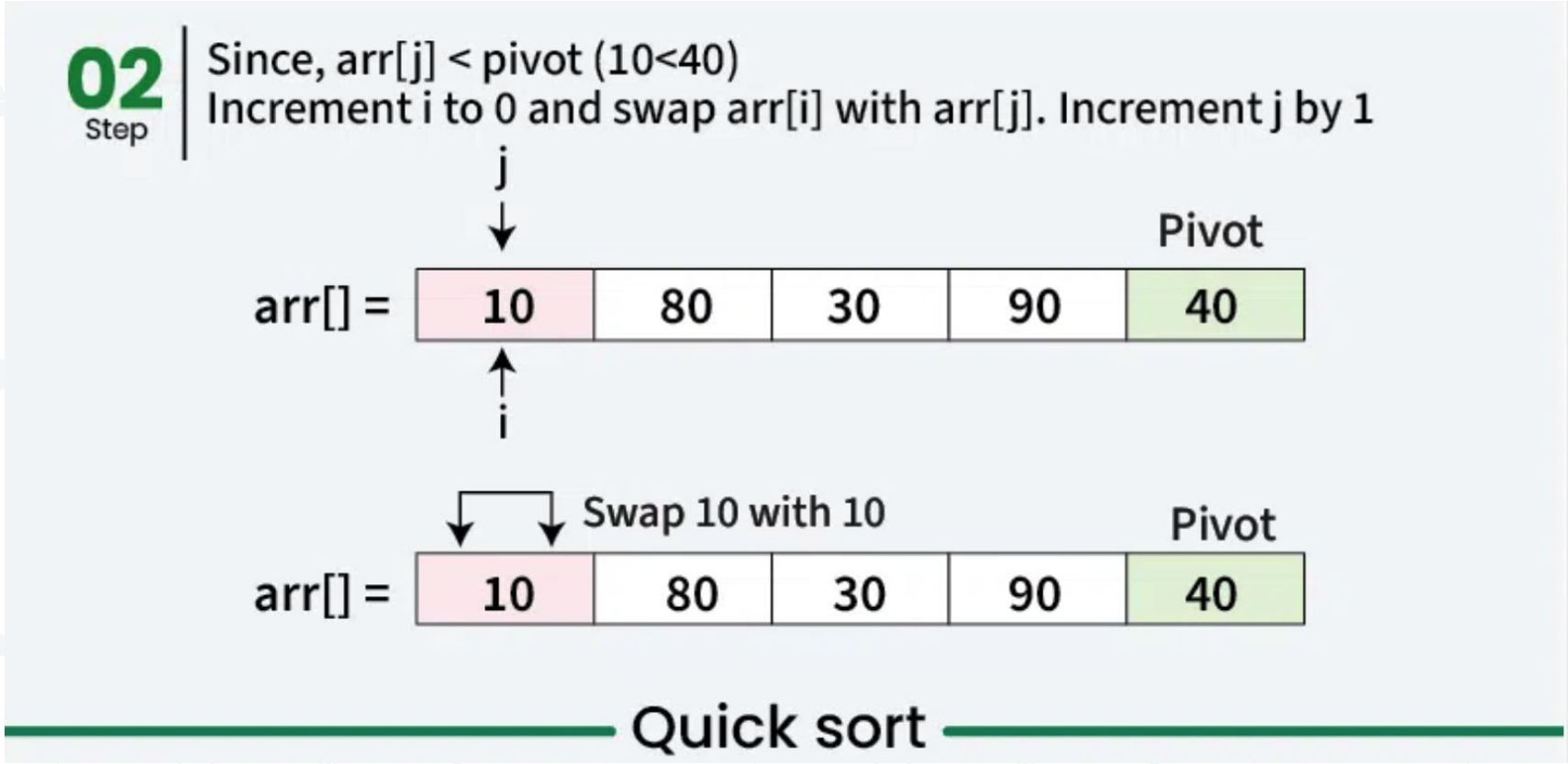
01 Step

Pivot Selection: The last element arr[4] = 40 is chosen as the pivot.
Initial Pointers: i = -1 and j = 0.



Quick sort

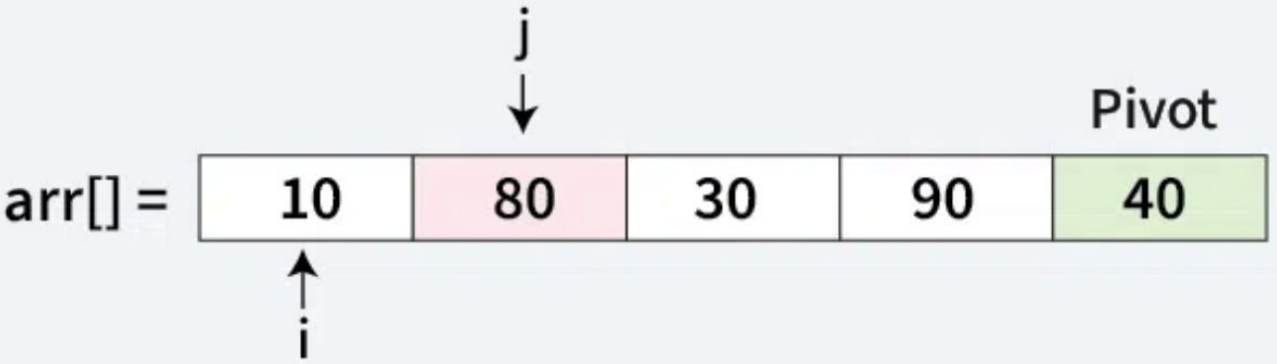
Quick Sort


02
Step
Since, arr[j] < pivot (10<40)
Increment i to 0 and swap arr[i] with arr[j]. Increment j by 1
j
Pivot
arr[] =
10
80
30
90
40
i
Swap 10 with 10
Pivot
arr[] =
10
80
30
90
40
Quick sort

Quick Sort

03 Step

Since, arr[j] > pivot (80>40)
No swap needed. Increment j by 1

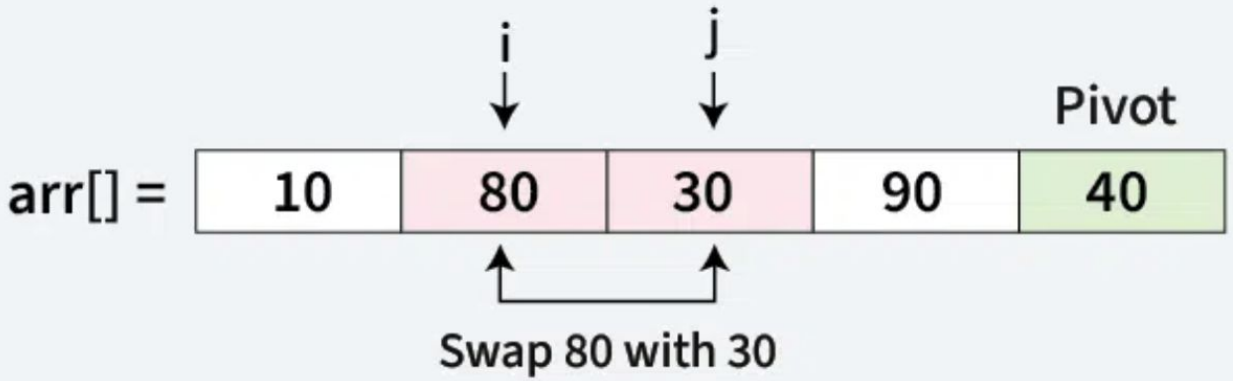


Quick sort

Quick Sort

04 Step

Since, arr[j] < pivot (30<40)
Increment i by 1 and swap arr[i] with arr[j]. Increment j by 1

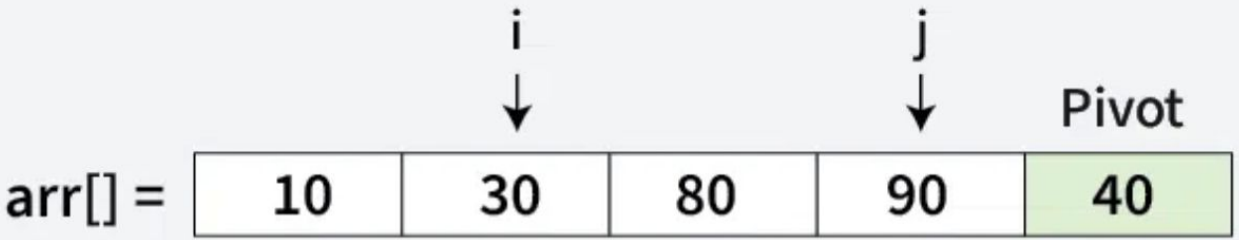


Quick sort

Quick Sort

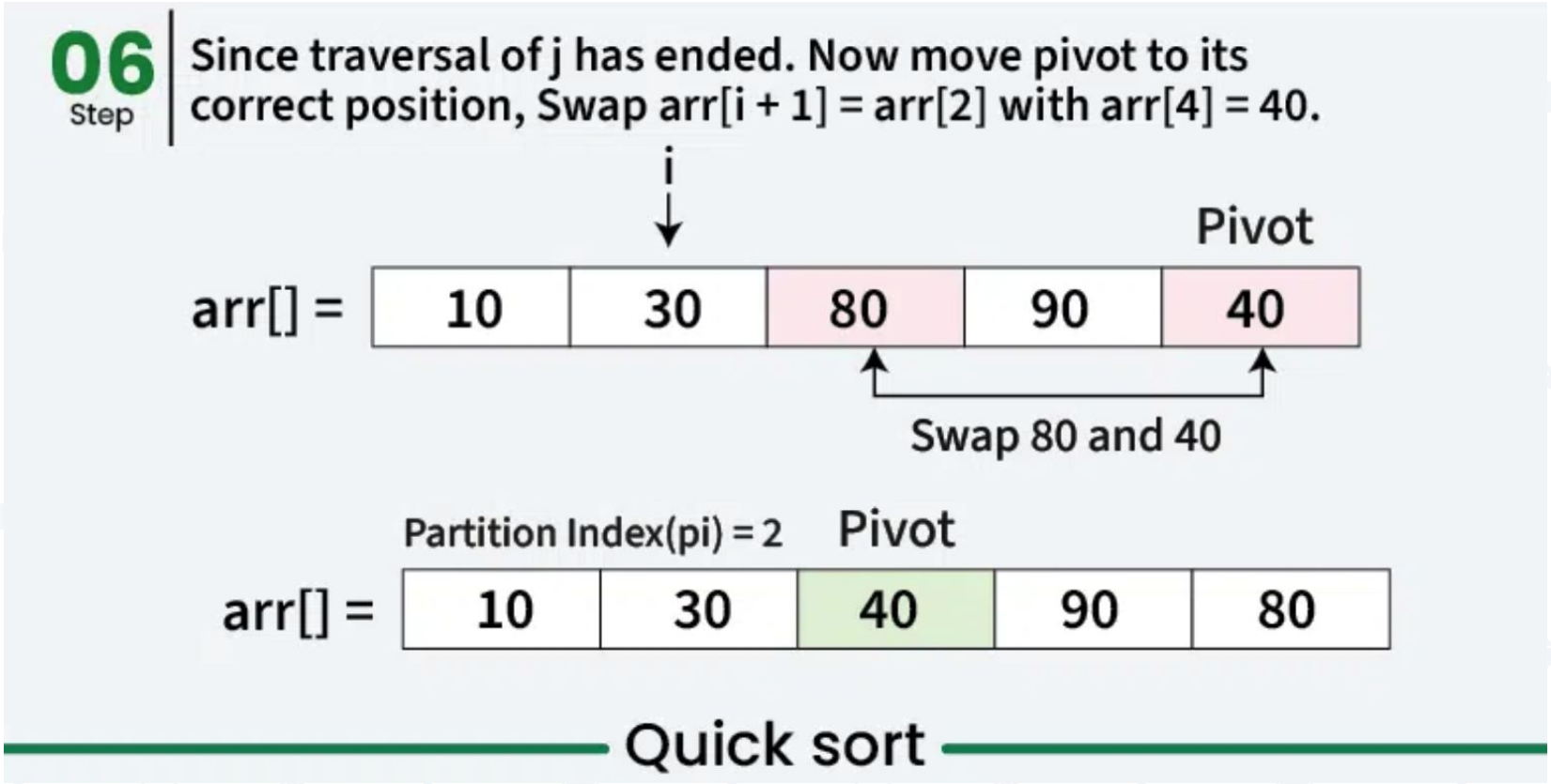
05 Step

Since, arr[j] > pivot (90>40)
No swap needed. Increment j by 1



Quick sort

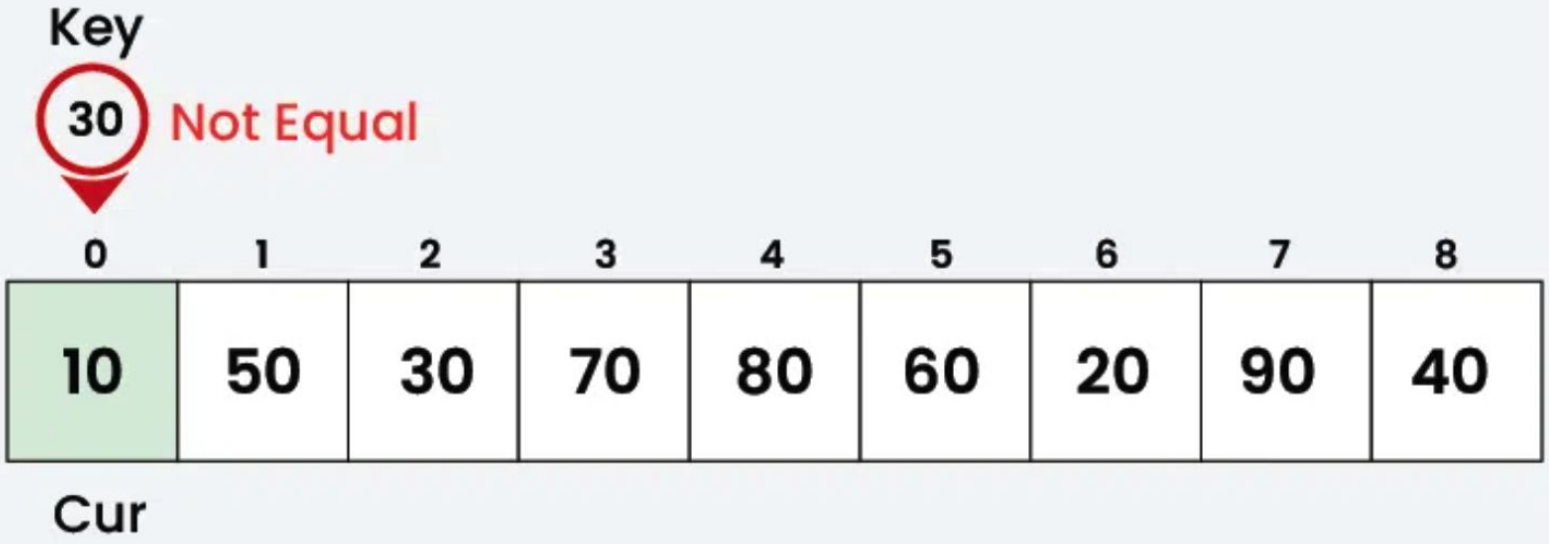
Quick Sort


06
Step
Since traversal of j has ended. Now move pivot to its correct position, Swap arr[i + 1] = arr[2] with arr[4] = 40.
i
Pivot
arr[] =
10
30
80
90
40
Swap 80 and 40
Partition Index(pi) = 2
Pivot
arr[] =
10
30
40
90
80
Quick sort

Linear Search

01 Step

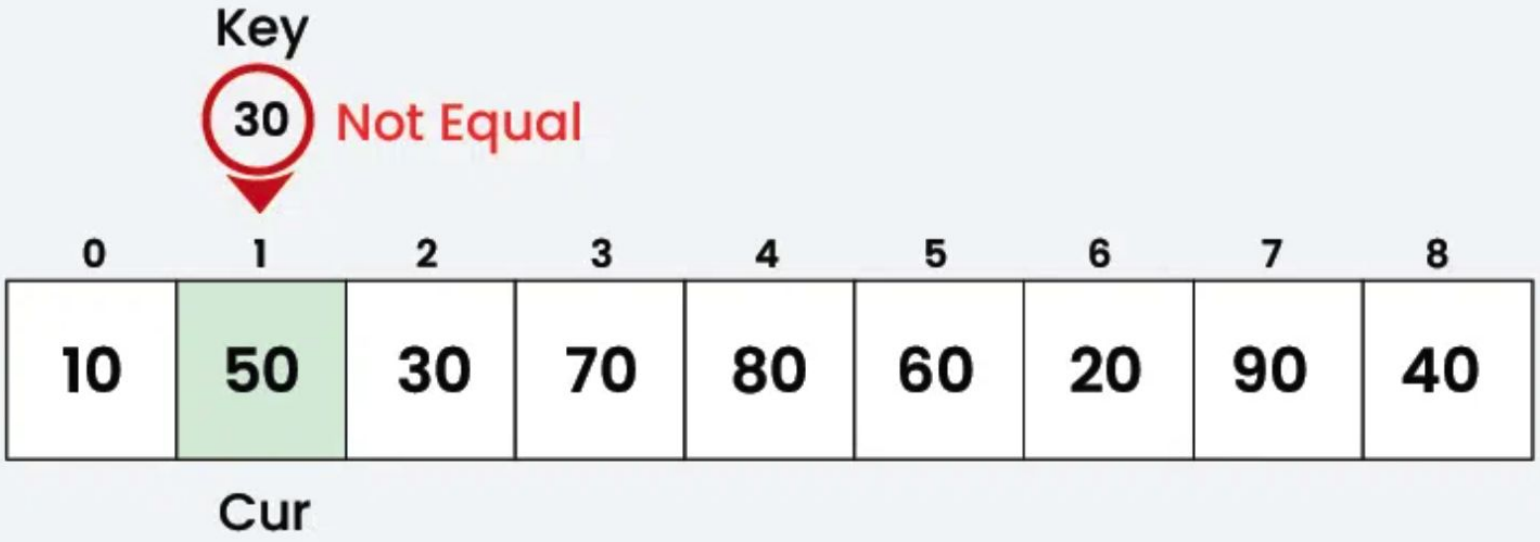
Compare the key with each element one by one starting from the 1st element.



Linear Search Algorithm

Linear Search

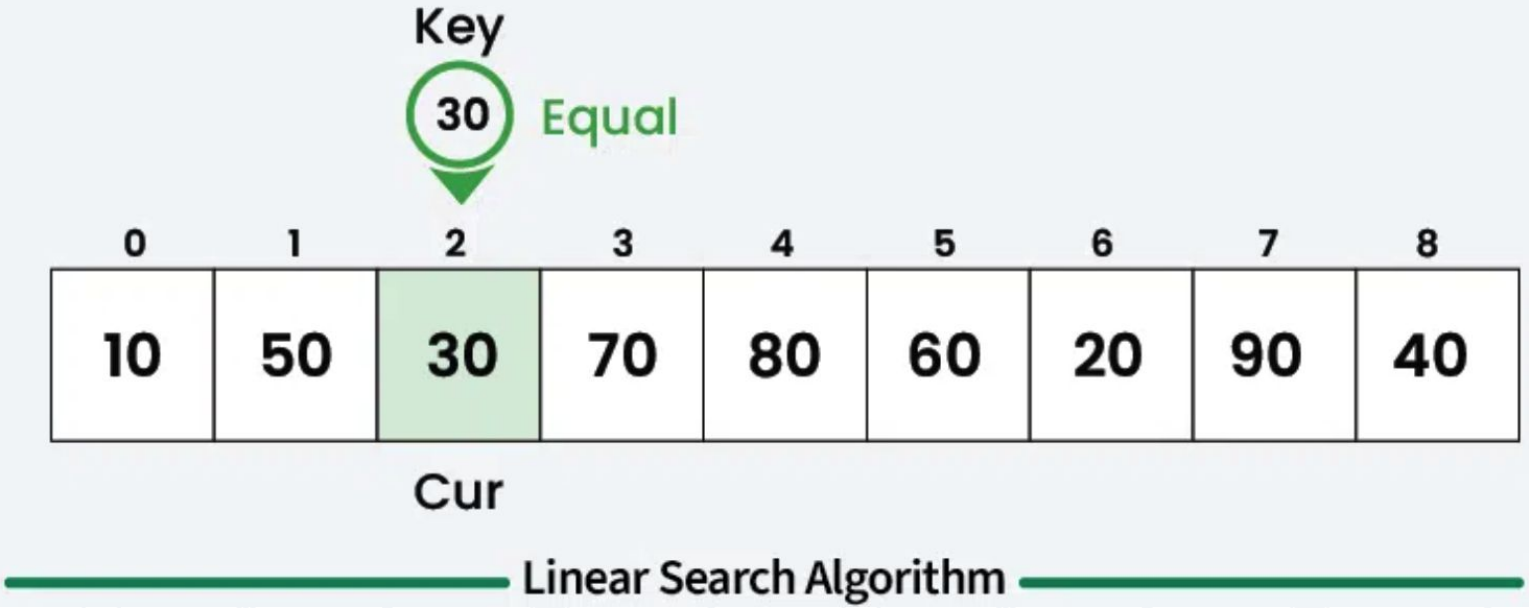
02 Step | Compare the key with 2nd element which is not equal to the key, so move to the next element



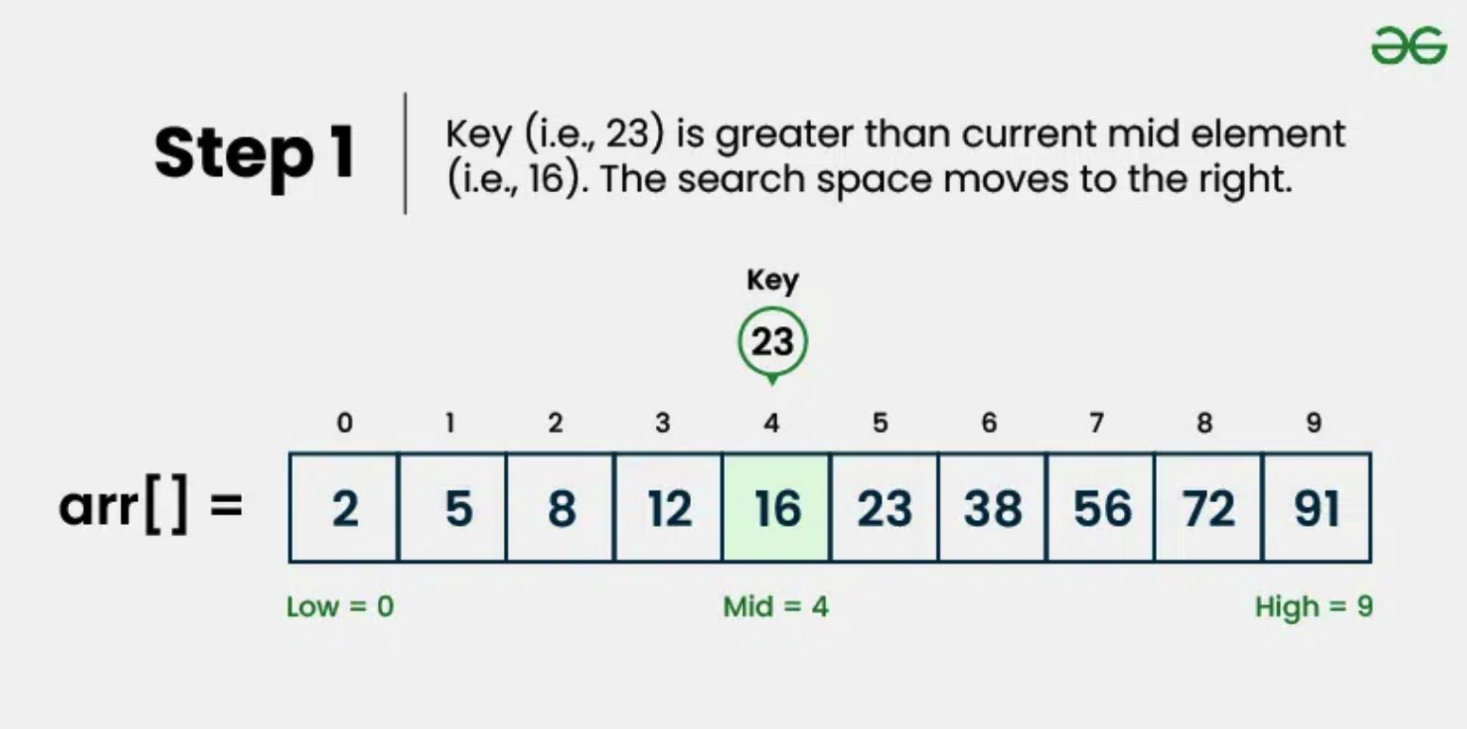
Linear Search Algorithm

Linear Search

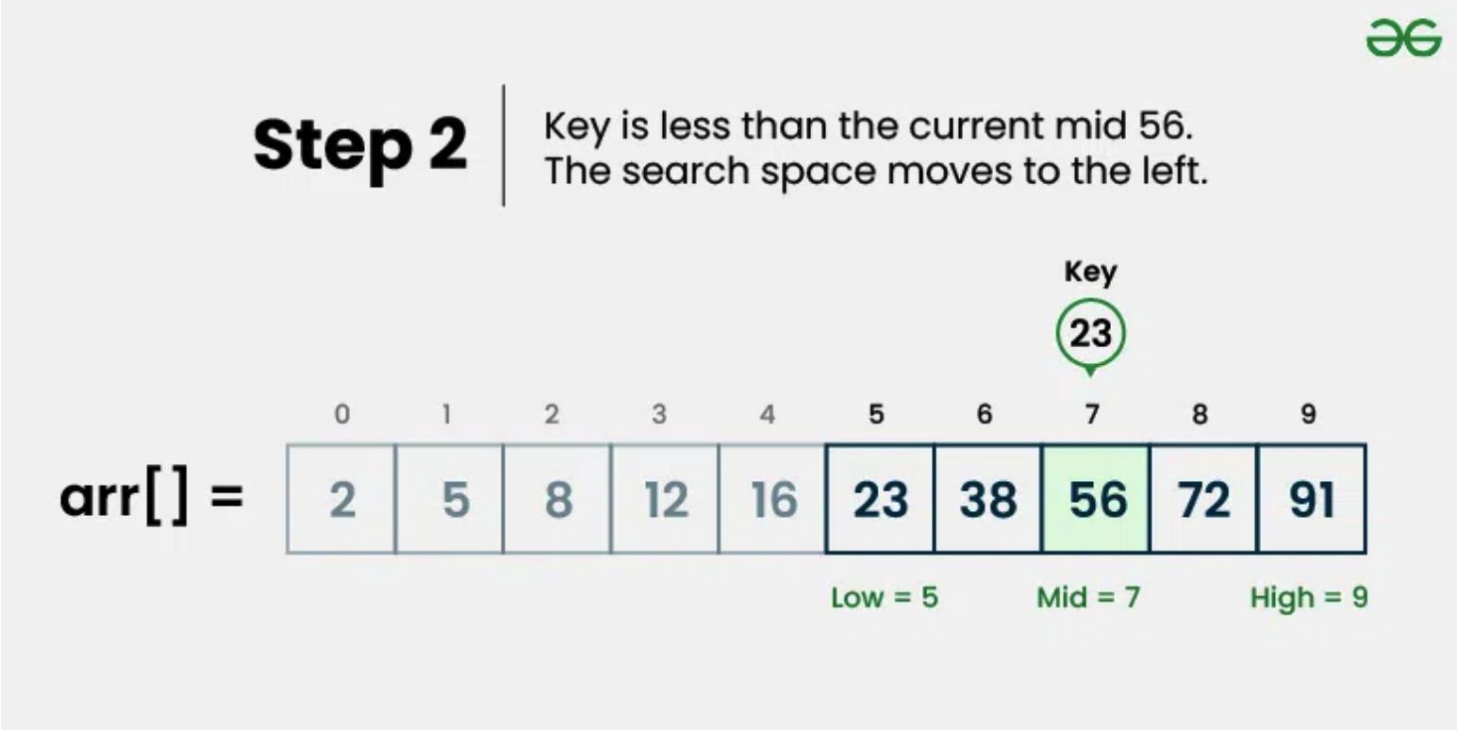
03 Step | Compare the key with the 3rd element. Key is found so stop the search.



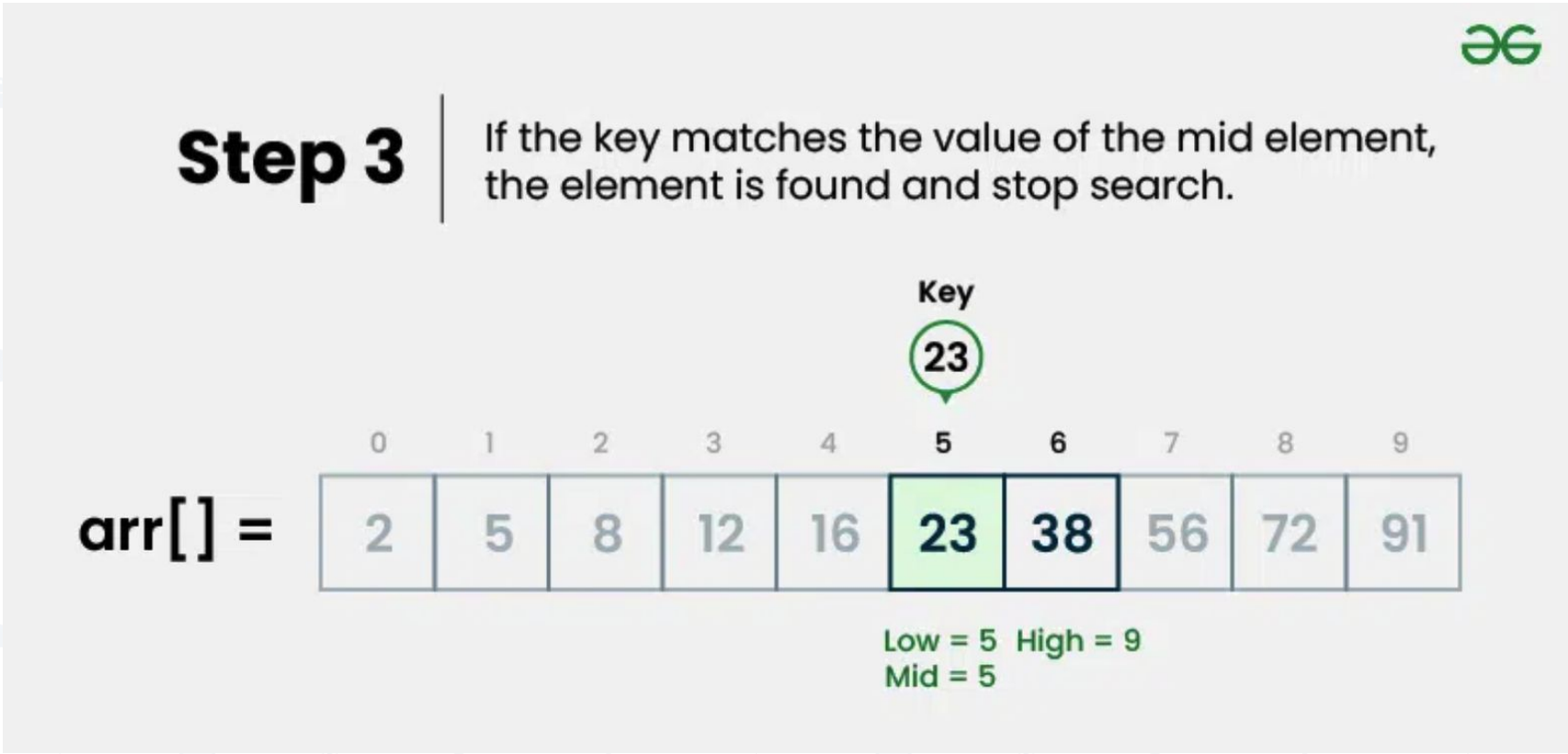
Binary Search



Binary Search



Binary Search



Складність алгоритмів

Складність алгоритму – це кількісна характеристика, що відображає споживані алгоритмом ресурси під час виконання.

1. **Логічна складність —** кількість людино-місяців, витрачених на створення алгоритму.
2. **Статична складність —** довжина опису алгоритмів (кількість операторів).
3. **Часова складність —** час виконання алгоритму.
4. **Ємнісна складність —** кількість умовних одиниць пам'яті, необхідних для роботи алгоритму.

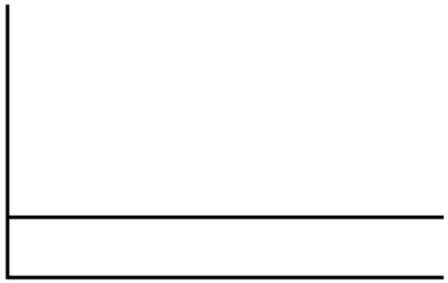
O(1)

means **constant complexity**

No matter the input size, complexity remains the same

e.g. accessing element at index from an array

```
let a = [1, 2, 3, 4, 5];
console.log(a[1]);
```



O(1) = 1
O(2) = 1
O(3) = 1
O(4) = 1
.....
O(n) = 1

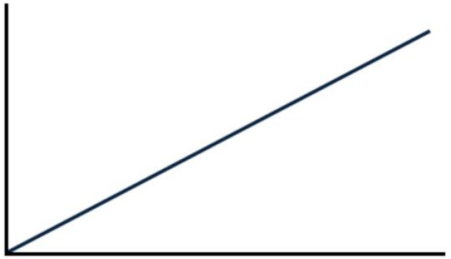
O(n)

means **linear complexity**

Complexity grows linearly over time - higher the number of inputs, higher the complexity.

e.g. looping over all the items of an array

```
for (let i = 0; i <= n; i++) {
   // do something
}
```



O(1) = 1
O(2) = 2
O(3) = 3
O(4) = 4
.....
O(n) = n

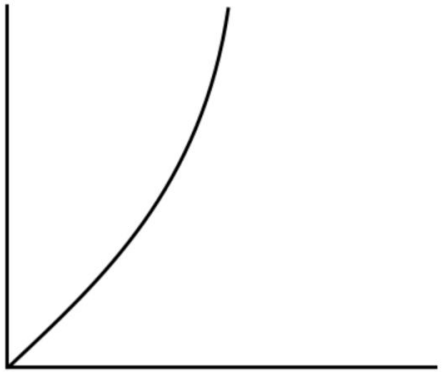
$O(n^2)$

means **quadratic complexity**

Complexity squares the number of inputs

e.g. loop within a loop

```
for (let i = 0; i <= n; i++) {
   for (let y = 0; y <= n; y++) {
      // do something
   }
}
```



$O(1) = 1$
$O(2) = 4$
$O(3) = 9$
$O(4) = 16$
.....
$O(n) = n^2$

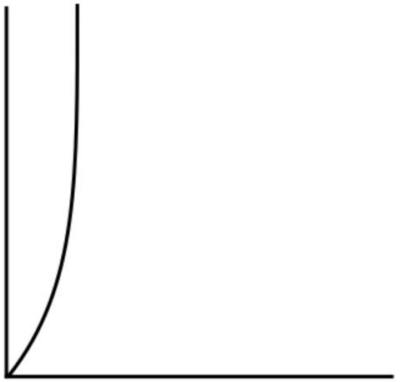
$O(2^n)$

means **exponential complexity**

complexity doubles with each addition to the input dataset

e.g. looping over all possible combinations of an array

```
function fibonacci(n) {
    if (n <= 1)  return n;
    return fibonacci(n - 2) + fibonacci(n-1);
}
```



$O(1) = 1$
$O(2) = 4$
$O(3) = 8$
$O(4) = 16$
$O(5) = 32$
$O(6) = 64$
.....
$O(n) = 2^n$

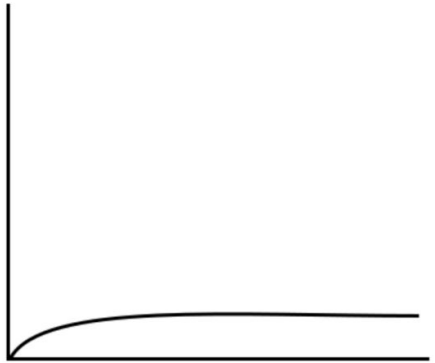
O(log n)

means **logarithmic complexity**

complexity goes up linearly while the input goes up exponentially

e.g. here is the example log 2

```
for (let i = 1; i <= n; i = i * 2) {
  console.log("hello");
}
```



O(10) = 1
O(200) = 2
O(300) = 3
O(400) = 4
O(500) = 5
O(600) = 6
.....
O(n) = log n

another famous example is binary search