Searching and sorting algorithms

Module 2

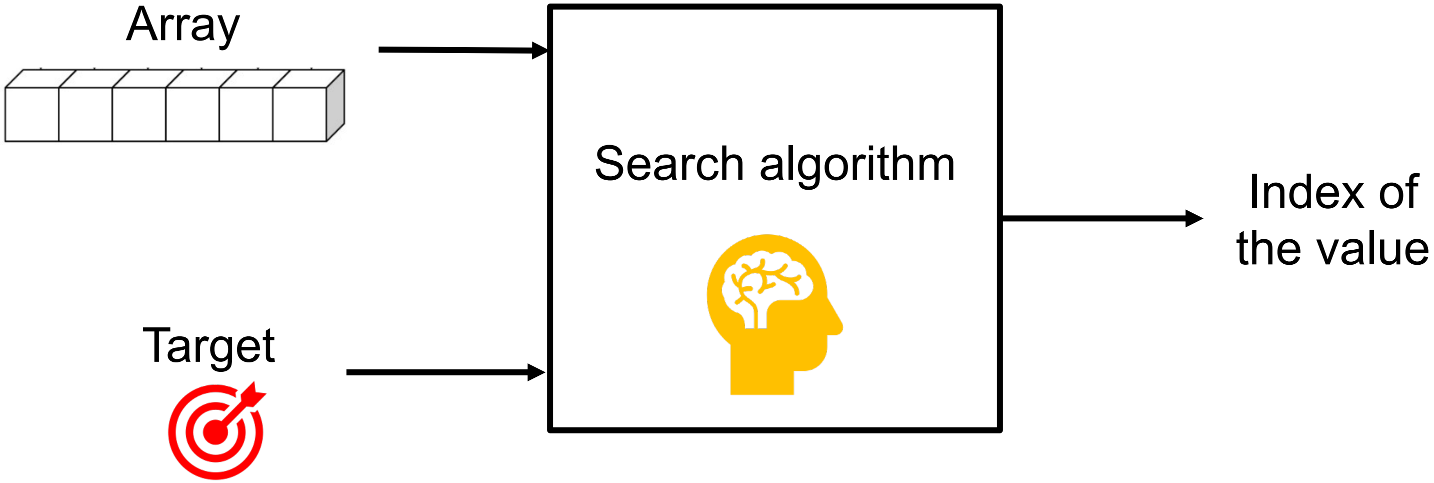
Chapter 7

Before we start

https://jolson615.github.io/createasearchalgorithm/index.html

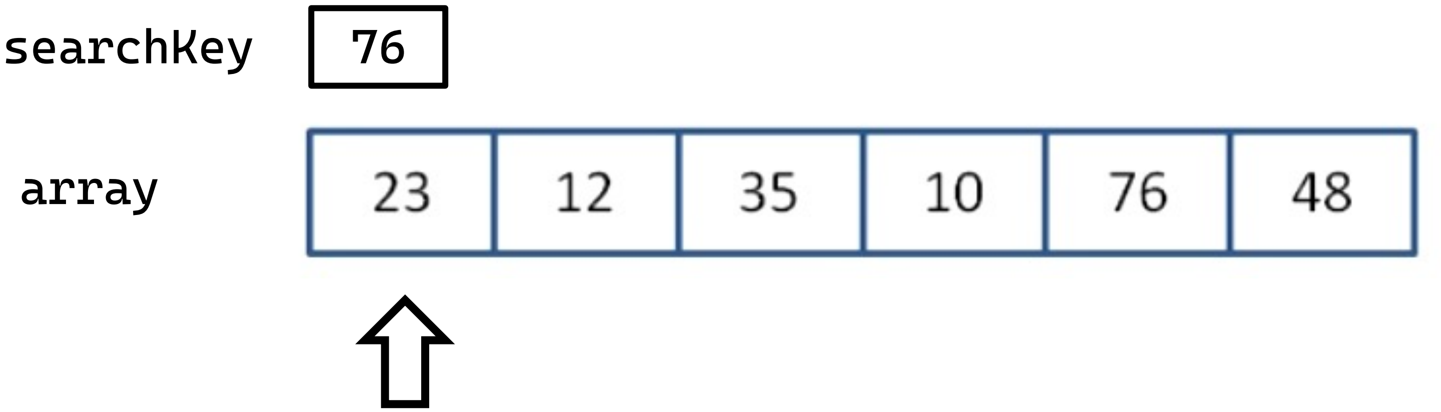
Search Algorithms

A search algorithm is useful when we are trying to value a specific value inside an array.



Linear search

The linear search algorithm consists in looking for the search key from sequentially.



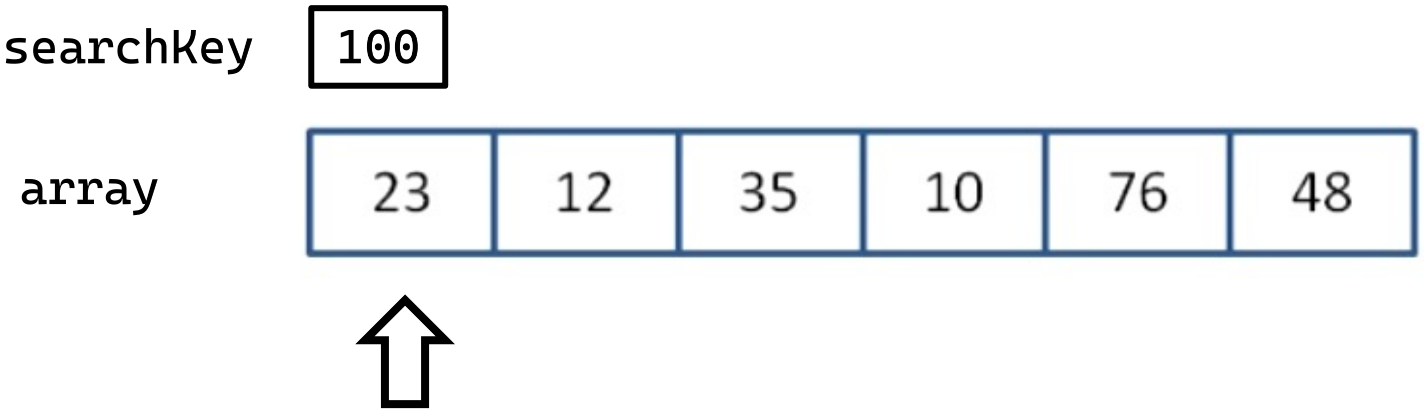
searchKey 76

array	23	12	35	10	76	48

Found!
Return index 4!

Linear search

The linear search algorithm consists in looking for the search key from sequentially.



searchKey `100`

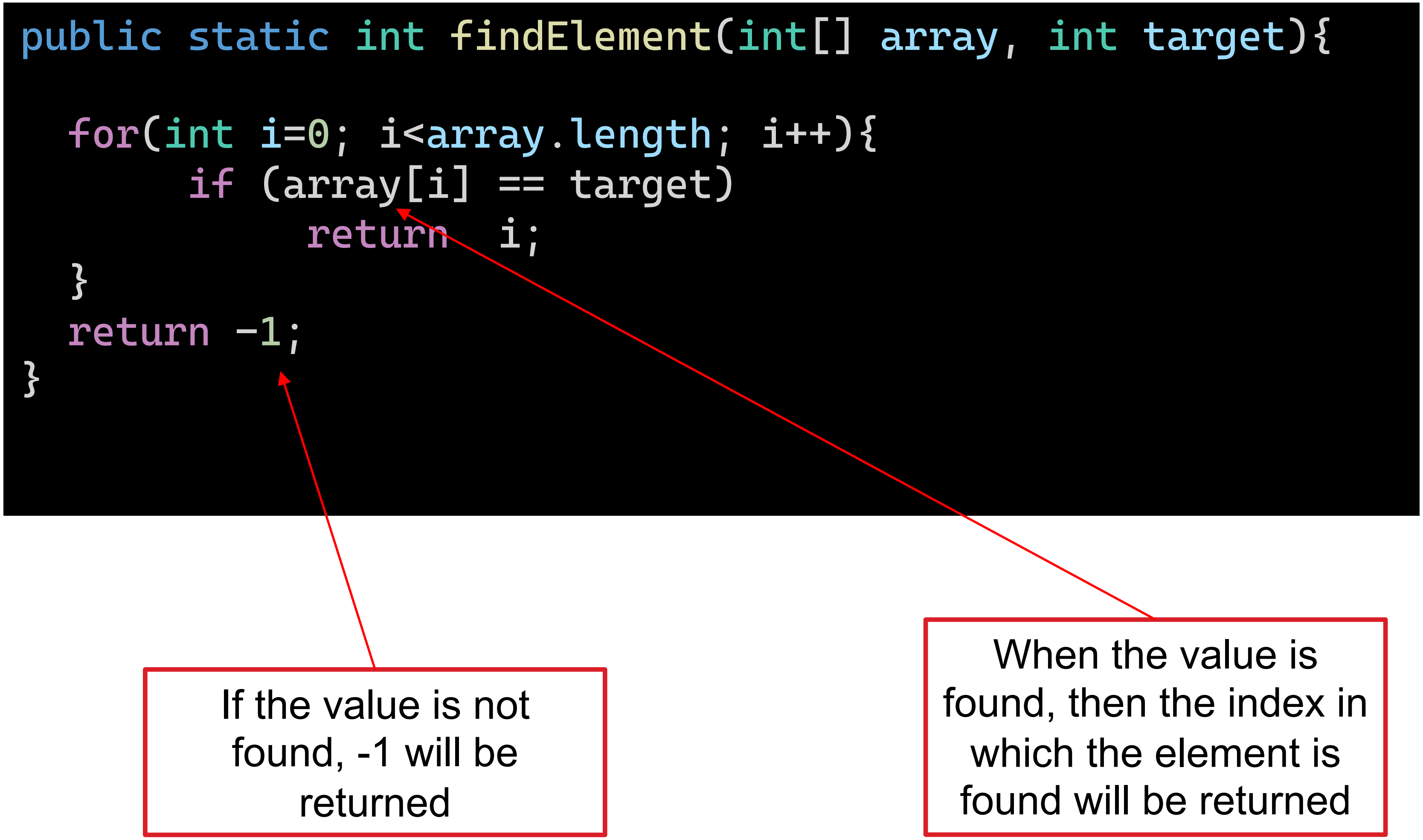
array	23	12	35	10	76	48

```
Not found!
Return -1!
```

Linear search

Linear search can be very slow, because in its worst case scenario, the value does not exist in the array.

Linear search

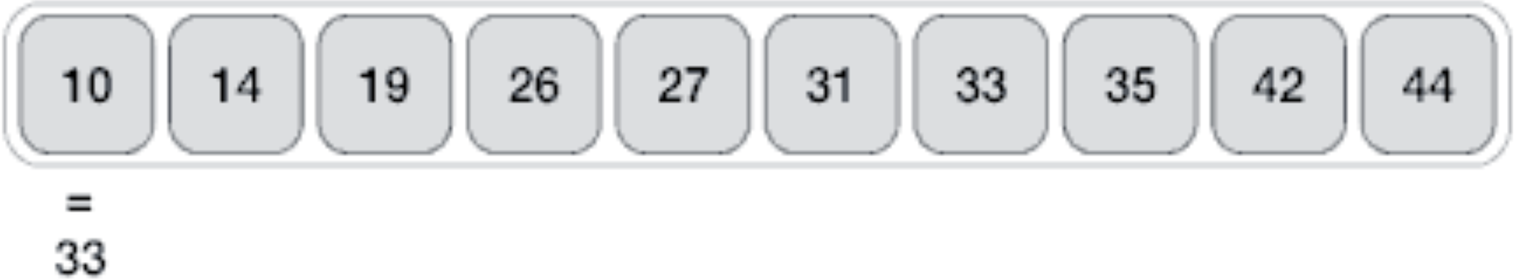


```
public static int findElement(int[] array, int target){

  for(int i=0; i<array.length; i++){
       if (array[i] == target)
            return  i;
  }
  return -1;
}
```

If the value is not found, -1 will be returned

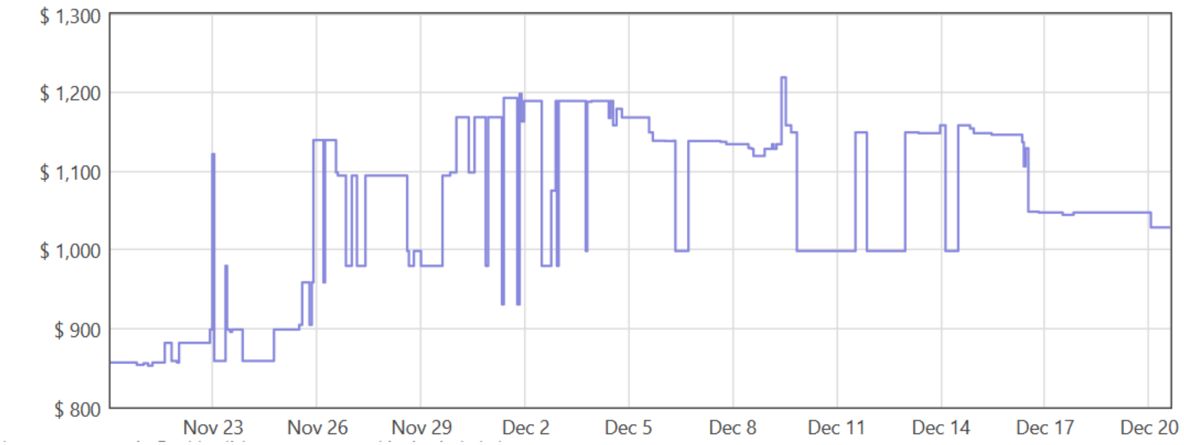
When the value is found, then the index in which the element is found will be returned

Linear Search



10	14	19	26	27	31	33	35	42	44

Exercise



Keepa
Amazon
Price Tracker

Design a java method that returns the smallest price a product has been sold for in Amazon.

```
public static void main(String[] args) throws Exception {
    double[] alexaPrices = {100, 156, 65, 100, 100, 100, 45, 45, 45, 65, 100};
    int bestPrice = findMinPrice(alexaPrices);
    System.out.println("The best price for the product is: " + alexaPrices[bestPrice]);
}

public static int findMinPrice(double[] prices) {
    //if array is invalid, return -1
    if (prices == null || prices.length <= 0) {
        return -1;
    }

    //start by assuming the smallest price will be in index 0
    int indexOfSmallestPrice = 0;

    //check against every array position
    for(int i=1;i<prices.length; i++) {
        if (prices[indexOfSmallestPrice] > prices[i]) {
            //if a smaller price is found, then replace old value
            indexOfSmallestPrice = i;
        }
     }

     return indexOfSmallestPrice;
}
```

Exercise - Swap two array elements

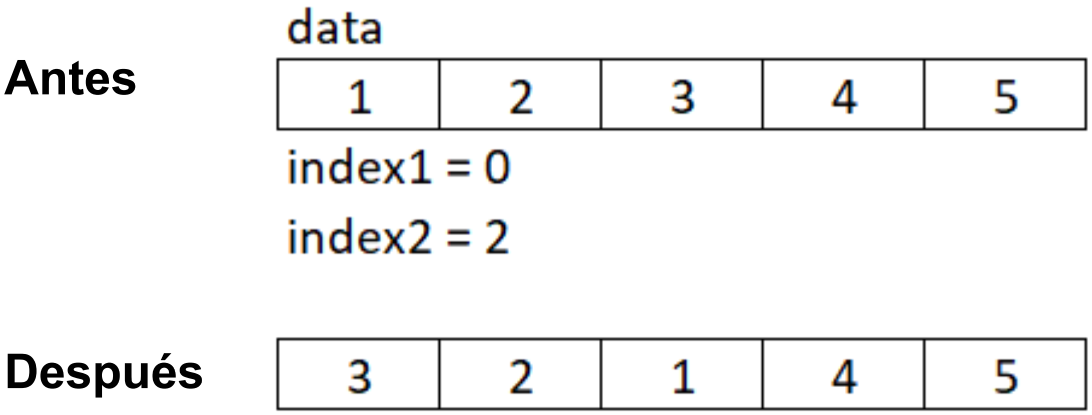
Code a method that swaps the contents of two array positions.

Inputs:

- `int [] data`
- `int index1`
- `int index2`

Outputs:

- None



Antes

data

1	2	3	4	5

index1 = 0

index2 = 2

Después

3	2	1	4	5

Sorting Algorithm

A sorting algorithm is used to place all elements in an array in some specific order

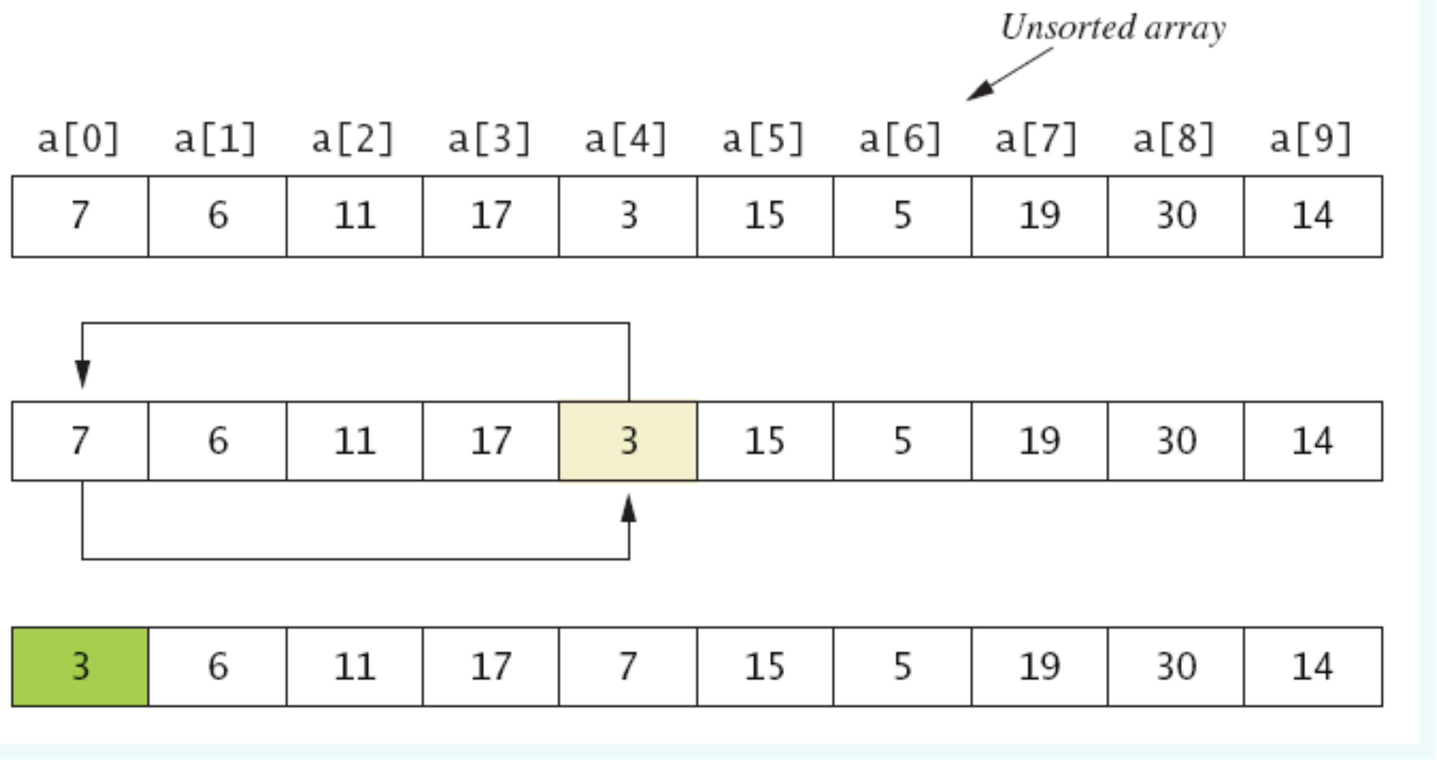
- Alphabetical
- Date
- Size
- Color

There are many sorting algorithms in computer science, some more efficient than others.

SELECTION SORT

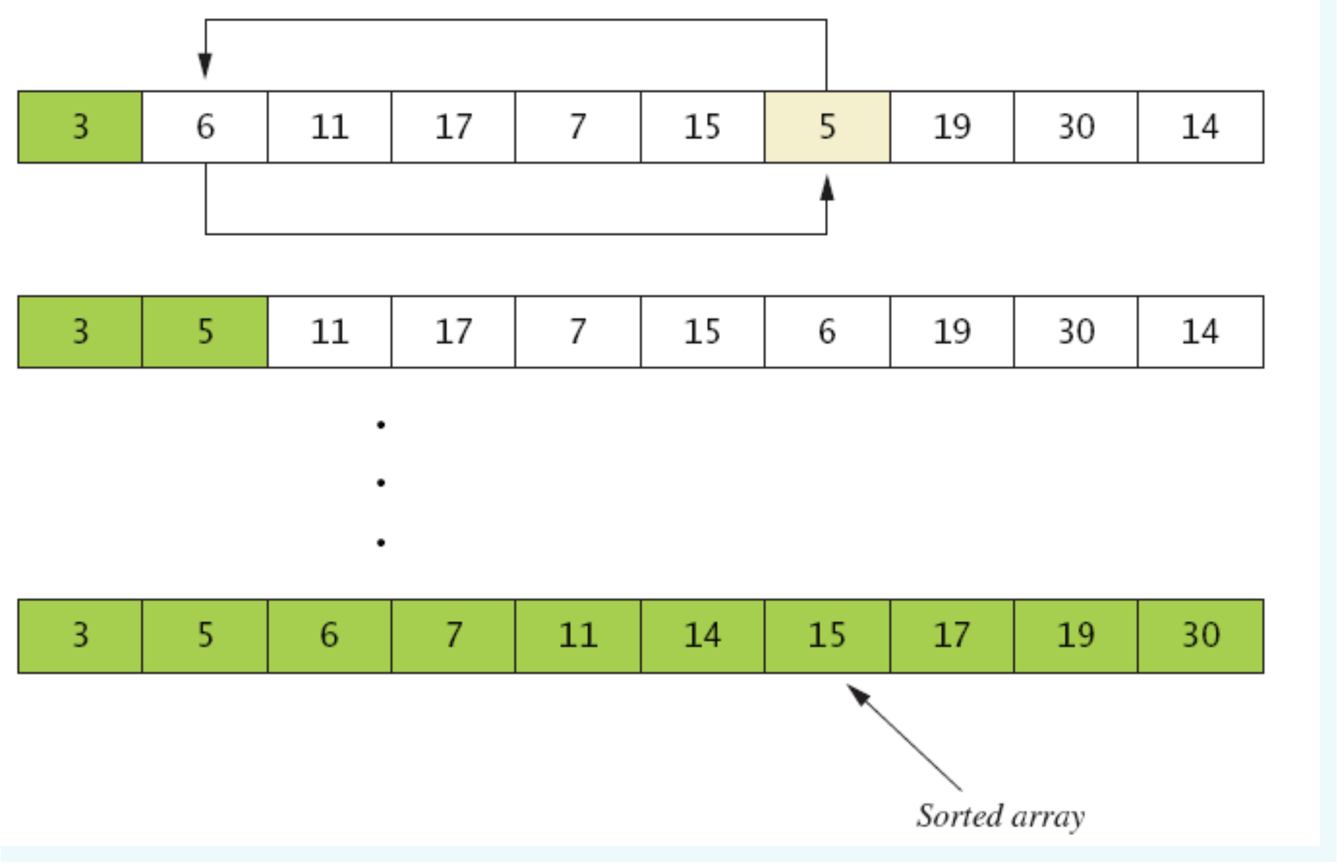
Selection Sort

Figure 7.5a



Selection Sort

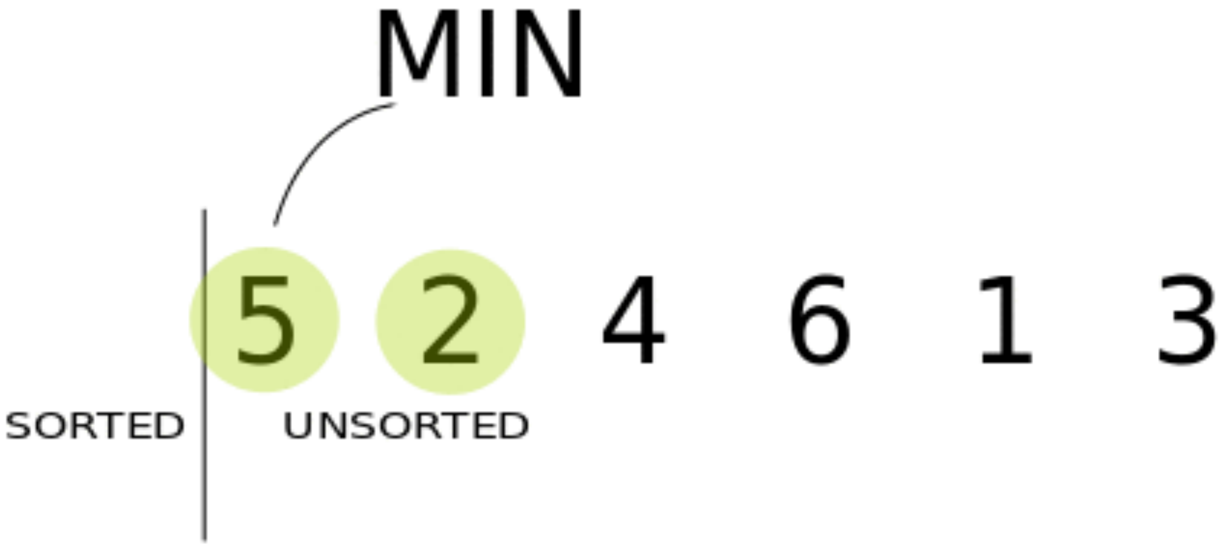
Figure 7.5b



3	6	11	17	7	15	5	19	30	14

3	5	11	17	7	15	6	19	30	14

3	5	6	7	11	14	15	17	19	30

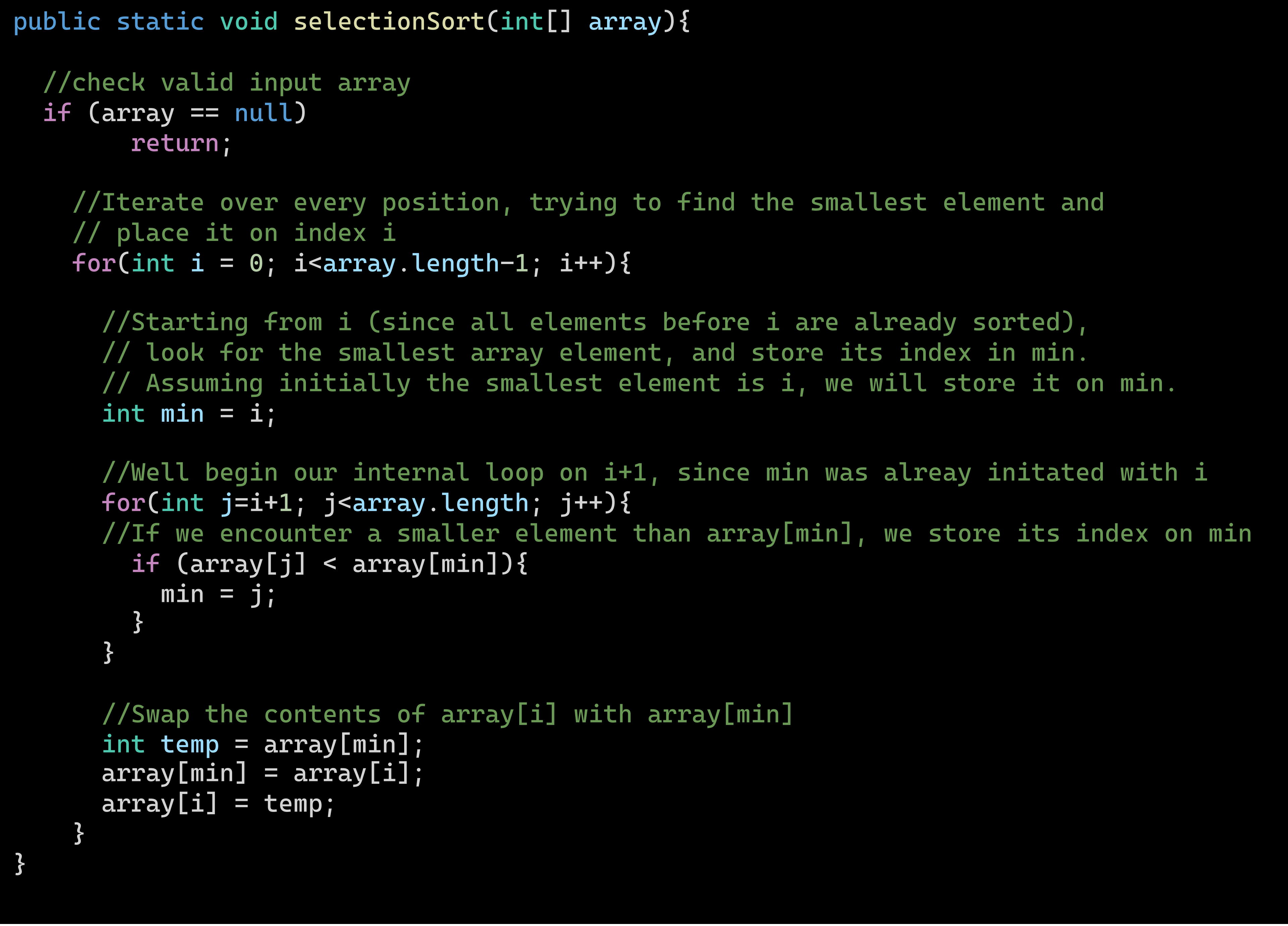


MIN

5 2 4 6 1 3

SORTED

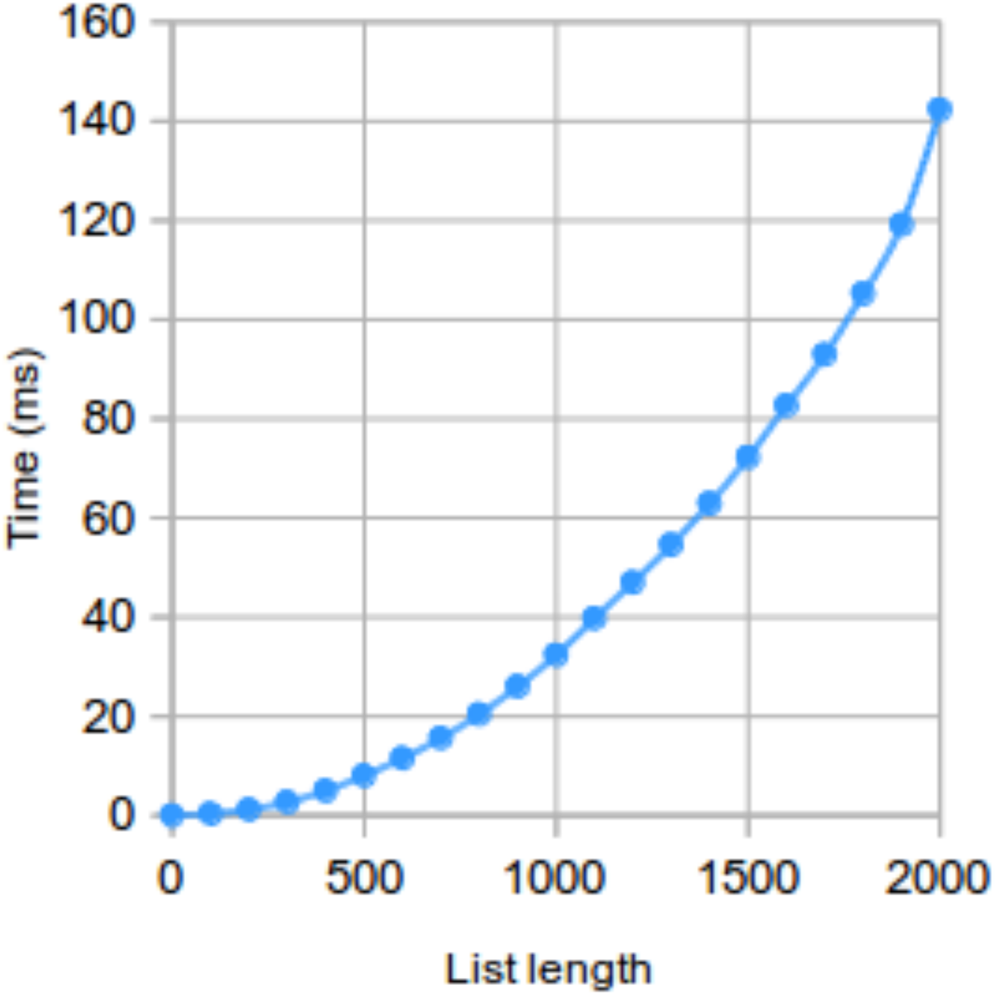
UNSORTED



```java
public static void selectionSort(int[] array){

  //check valid input array
  if (array == null)
        return;

    //Iterate over every position, trying to find the smallest element and
    // place it on index i
    for(int i = 0; i<array.length-1; i++){

      //Starting from i (since all elements before i are already sorted),
      // look for the smallest array element, and store its index in min.
      // Assuming initially the smallest element is i, we will store it on min.
      int min = i;

      //Well begin our internal loop on i+1, since min was alreay initated with i
      for(int j=i+1; j<array.length; j++){
      //If we encounter a smaller element than array[min], we store its index on min
        if (array[j] < array[min]){
          min = j;
        }
      }

      //Swap the contents of array[i] with array[min]
      int temp = array[min];
      array[min] = array[i];
      array[i] = temp;
    }
}
```

Selection Sort

Selection sort is one of the simpler sorting algorithms, **but it can be very inefficient for large arrays!**



BUBBLE SORT

Bubble Sort

BubbleSort is a sorting algorithm that works by swapping unordered **adjacent elements**.

The algorithm will finish once all elements are in order.

Bubble Sort

First Run

(**5 1** 4 2 8) → (**1 5** 4 2 8) Swap 5 and 1

(1 **5 4** 2 8) → (1 **4 5** 2 8) 5>4, swap places

(1 4 **5 2** 8) → (1 4 **2 5** 8) 5>2, swap places

(1 4 2 **5 8**) → (1 4 2 **5 8**) 5<8, no swap

Bubble Sort

Second run

(**1** **4** 2 5 8) → (**1** **4** 2 5 8)

(1 **4** **2** 5 8) → (1 **2** **4** 5 8) Since 4>2, we swap the values

(1 2 **4** **5** 8) → (1 2 **4** **5** 8)

(1 2 4 **5** **8**) → (1 2 4 **5** **8**)

Bubble Sort

Third run

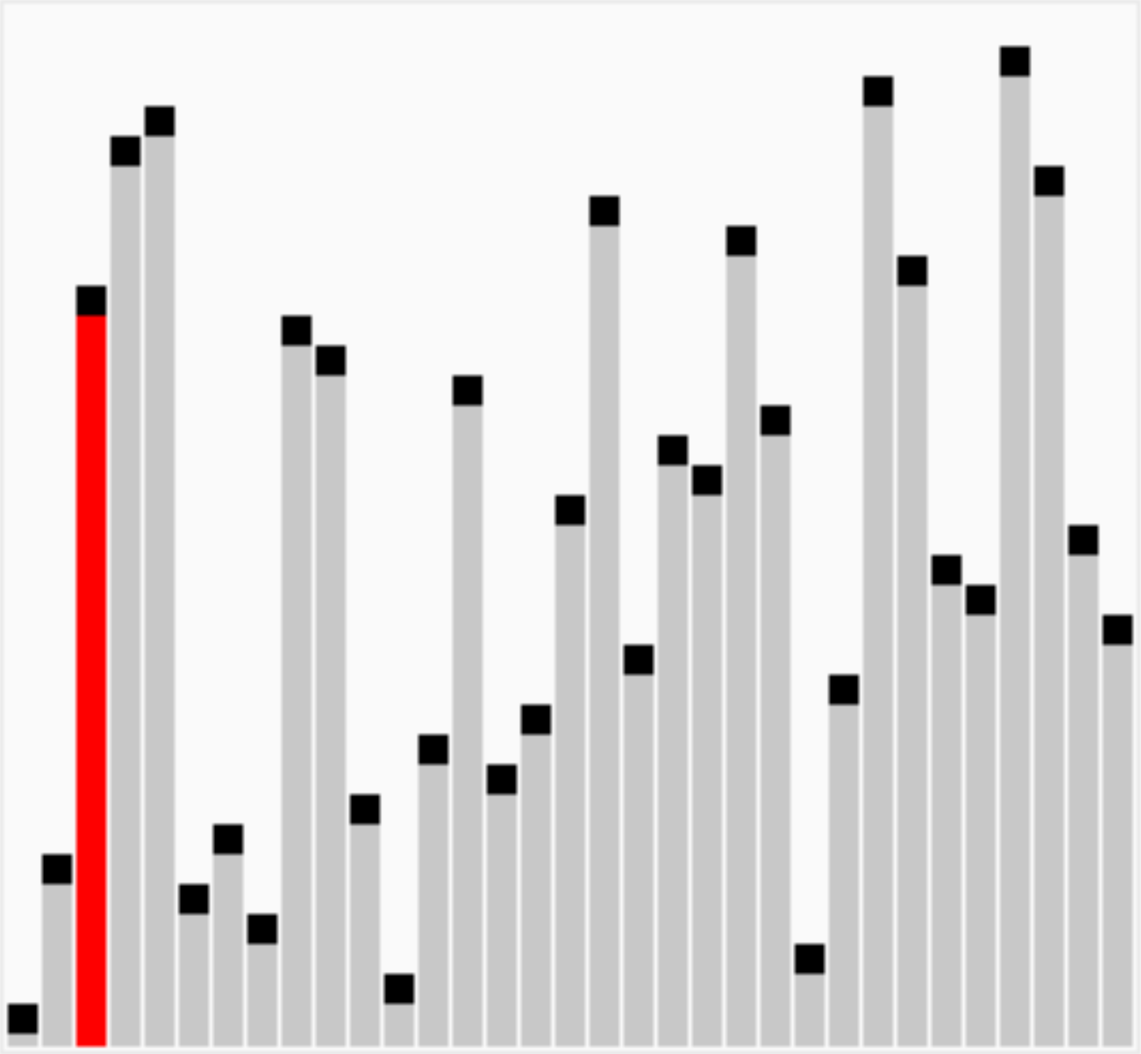
(**1 2** 4 5 8) → (**1 2** 4 5 8)

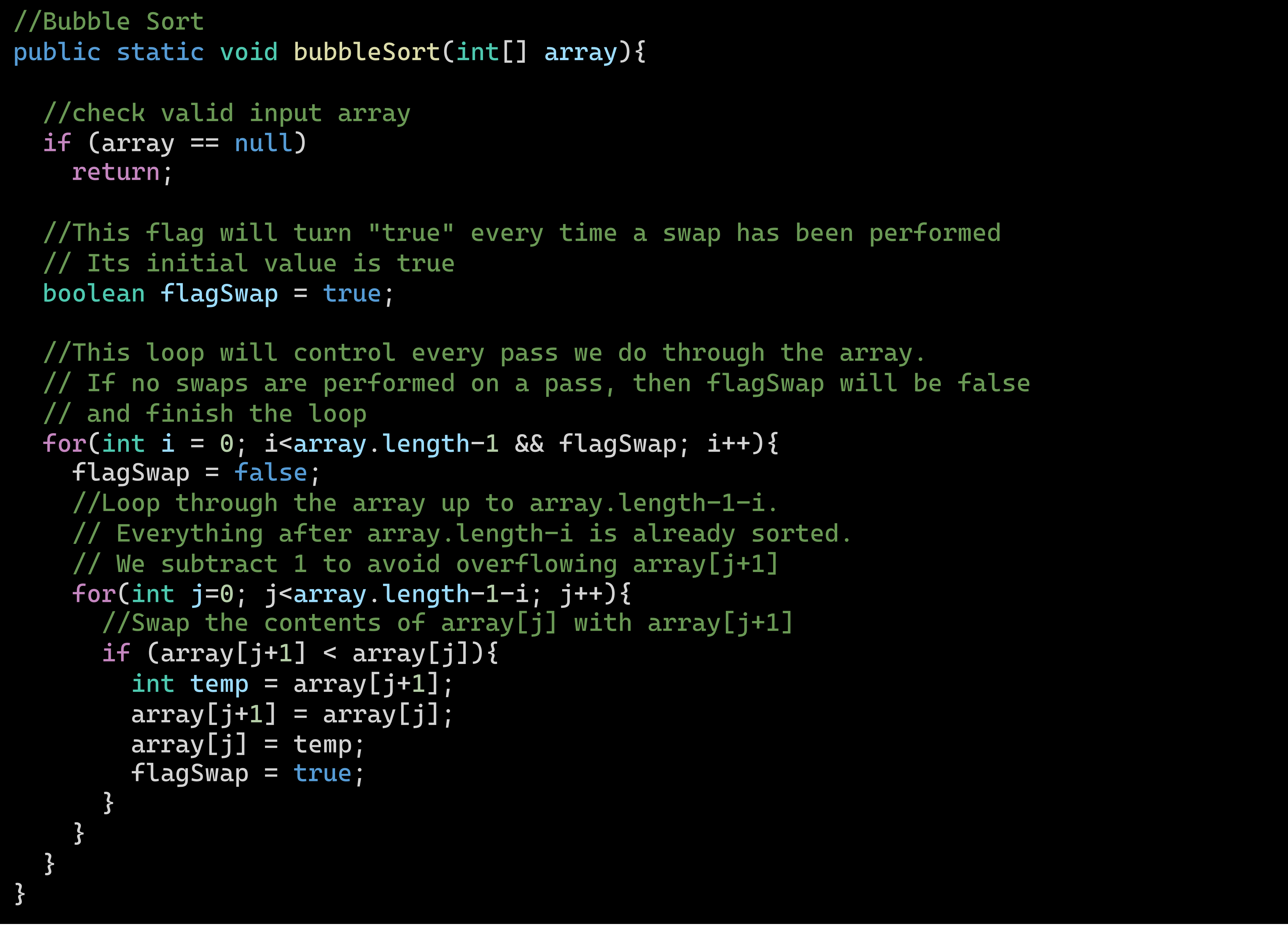
(1 **2 4** 5 8) → (1 **2 4** 5 8)

(1 2 **4 5** 8) → (1 2 **4 5** 8)

(1 2 4 **5 8**) → (1 2 4 **5 8**)

When a run is completed without doing any swaps, then the array is already ordered

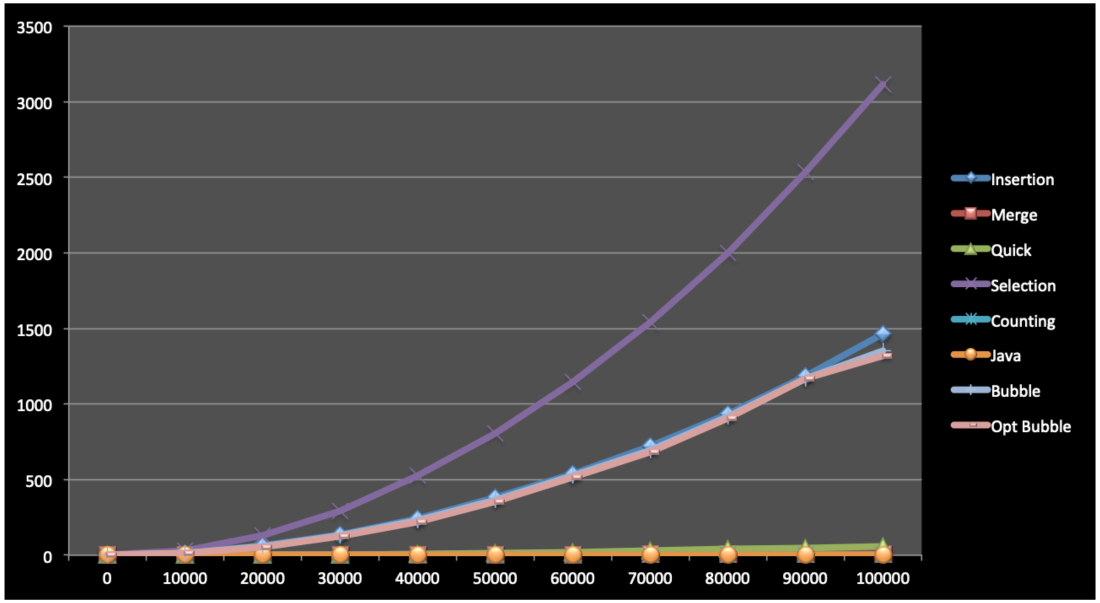




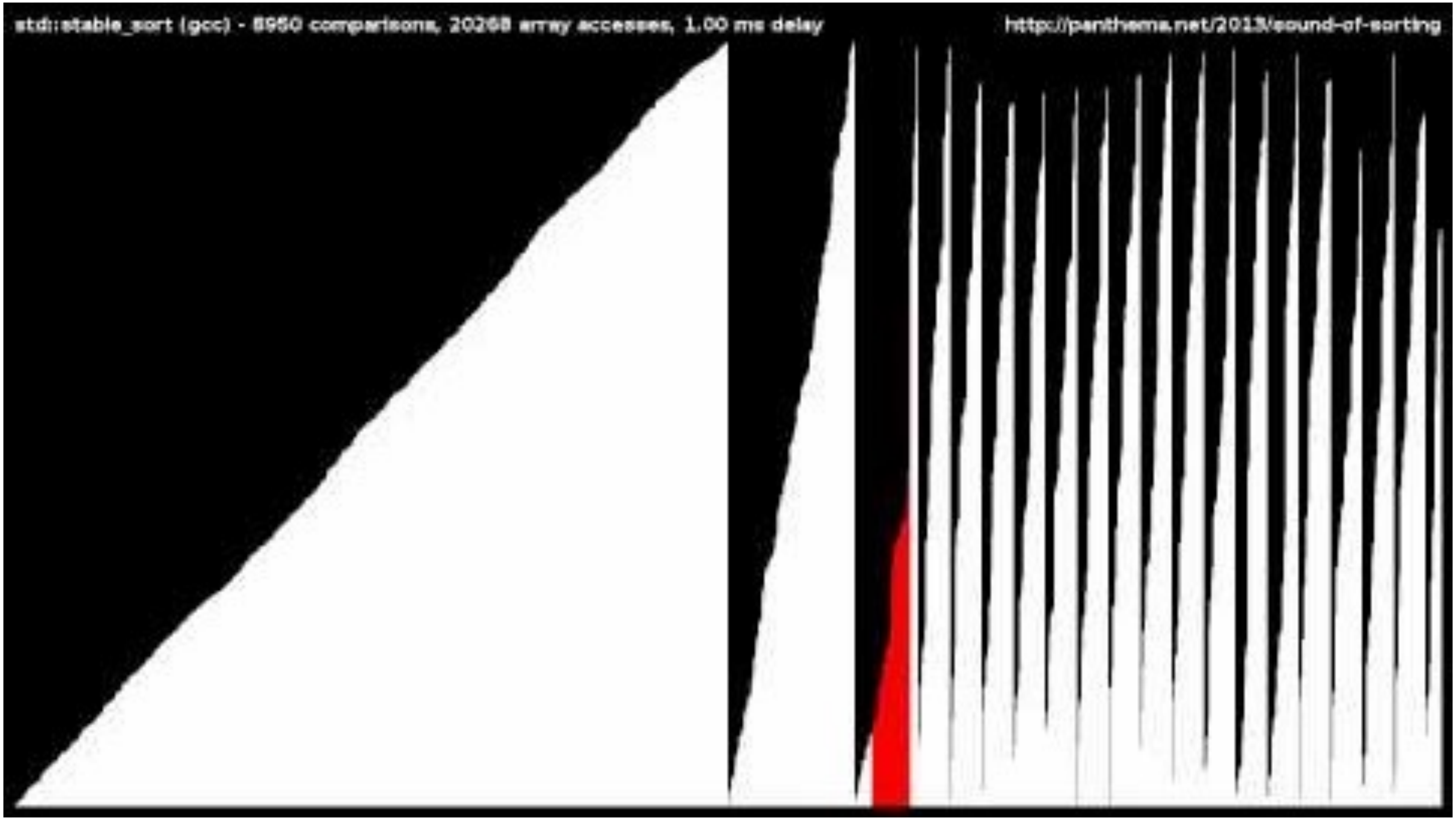
```java
//Bubble Sort
public static void bubbleSort(int[] array){

  //check valid input array
  if (array == null)
    return;

  //This flag will turn "true" every time a swap has been performed
  // Its initial value is true
  boolean flagSwap = true;

  //This loop will control every pass we do through the array.
  // If no swaps are performed on a pass, then flagSwap will be false
  // and finish the loop
  for(int i = 0; i<array.length-1 && flagSwap; i++){
    flagSwap = false;
    //Loop through the array up to array.length-1-i.
    // Everything after array.length-i is already sorted.
    // We subtract 1 to avoid overflowing array[j+1]
    for(int j=0; j<array.length-1-i; j++){
      //Swap the contents of array[j] with array[j+1]
      if (array[j+1] < array[j]){
        int temp = array[j+1];
        array[j+1] = array[j];
        array[j] = temp;
        flagSwap = true;
      }
    }
  }
}
```

Bubble Sort

- In the best-case scenario **(the array is already ordered)**, the algorithm will finish as soon as the first pass is completed.
- In the worst-case scenario, **(the array is ordered from largest to smallest),** each pass does n-1 swaps and n-1 comparisons.



Sorting algorithms



std::stable_sort (gcc) - 8950 comparisons, 20268 array accesses, 1.00 ms delay
http://panthema.net/2013/sound-of-sorting