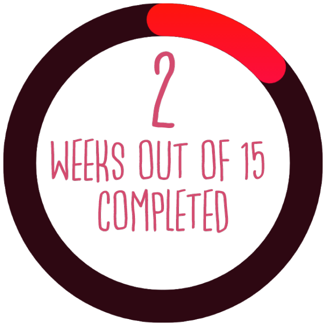
CS121 Data Structures
Quadratic Sorting

Monika Stepanyan
`mstepanyan@aua.am`

Spring 2024

Important Data and Statistics



- 26 classes remaining
- 39 days till the Midterm exam I
- 30 days till the Spring Break

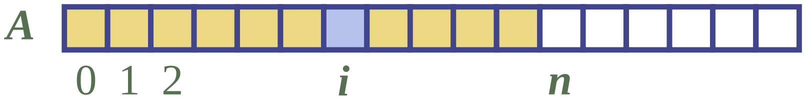
Array Definition

An **array** is a sequenced collection of variables all of the same type.

Each variable, or **cell**, in an array has an **index**, which uniquely refers to the value stored in that cell.

The cells of an array A, are numbered 0, 1, 2, and so on.

Each value stored in an array is often called an array **element**.



Array Length and Capacity

The length of an array determines the maximum number of things that can be stored in the array

We will sometimes refer to the length of an array as its **capacity**.

In Java, the length of an array named *a* can be accessed using the syntax `a.length`.

Thus, the cells of an array *a* are numbered 0, 1, 2, and so on, up through $a.\texttt{length} - 1$.

The cell with index *k* can be accessed with syntax $a[k]$.

Sorting an Array

Sorting problem: start with an unordered array of elements and rearrange them into nondecreasing order

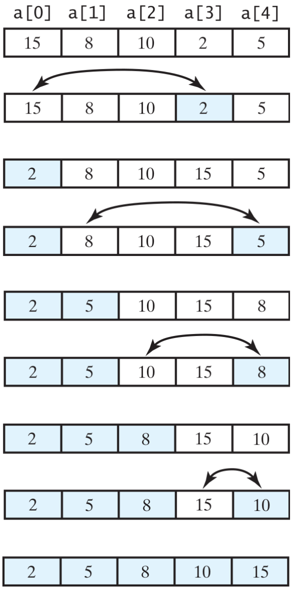
- Selection Sort
- Insertion Sort
- Bubble Sort

Selection Sort

The **selection sort** algorithm works as follows:

1. Find (**select**) the smallest element in the array, and exchange it with the element in the first position
2. Find the second smallest element and exchange it with the element in the second position
3. Continue in this way until the entire array is sorted

Selection Sort: Illustration



Selection Sort: An Implementation

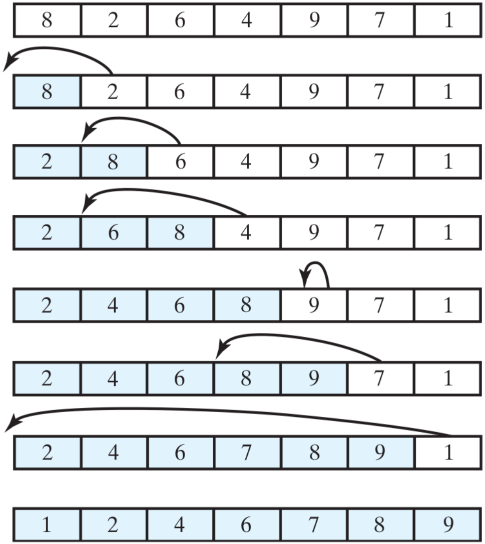
```
/** Selection-sort of an array of integers into nondecreasing order */
public static void selectionSort(int[ ] data) {
  int n = data.length;
  for (int k = 0; k < n - 1; k++) {      // begin with first position
    int min = k;                          // position of current minimum
    for (int j = k + 1; j < n; j++)      // start at next position
      if (data[j] < data[min])            // a new minimum found
        min = j;                          // update position of minimum
    int temp = data[k];               // put minimum in correct position
    data[k] = data[min];
    data[min] = temp;
  }
}
```

Insertion Sort

The **insertion sort** algorithm proceeds by considering one element at a time, placing (**inserting**) the element in the correct position relative to those before it.

1. First element—trivially sorted by itself
2. Second element: if it is smaller than the first, swap them
3. Third element: swap it leftward until in its proper place
4. Continue in this way until the entire array is sorted

Insertion Sort: Illustration



Insertion Sort: An Implementation

```java
/** Insertion-sort of an array of integers into nondecreasing order */
public static void insertionSort(int[ ] data) {
  int n = data.length;
  for (int k = 1; k < n; k++) {          // begin with second integer
    int cur = data[k];                   // time to insert cur=data[k]
    int j = k;                           // find correct index j for cur
    while (j > 0 && data[j-1] > cur) {  // thus, data[j-1] must go after cur
      data[j] = data[j-1];               // slide data[j-1] rightward
      j--;                               // and consider previous j for cur
    }
    data[j] = cur;                       // this is the proper place for cur
  }
}
```

Bubble Sort

Keep passing through the array, swapping adjacent elements that are out of order, continuing until the array is sorted.

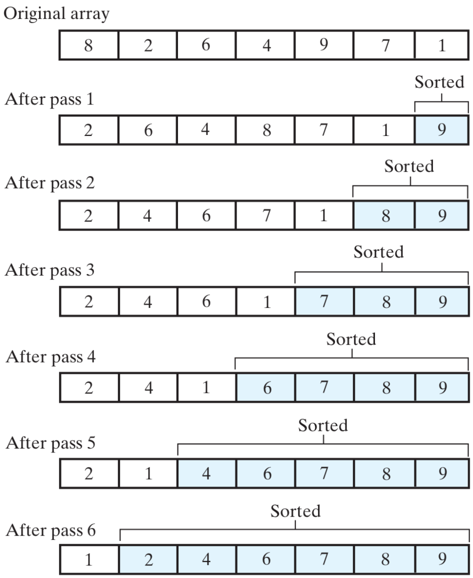
Assume at each outer pass, we **‘bubble’** from the beginning of the array to its end.

After one pass, the largest element is in place.

After a second pass, the second largest element is in place.

And so on, until all elements are in place.

Bubble Sort: Illustration



Bubble Sort: An Implementation

```java
/** Bubble-sort of an array of integers into nondecreasing order */
public static void bubbleSort(int[ ] data) {
  int n = data.length;
  for (int k = 0; k < n - 1; k++) {      // find last k + 1 elements
    boolean swapped = false;
    for (int j = 0; j < n - k - 1; j++) // consider successive elements
      if (data[j] > data[j + 1]) {      // compare successive elements
        int temp = data[j];             // swap successive elements
        data[j] = data[j + 1];
        data[j + 1] = temp;
        swapped = true;
      }
    if (!swapped)                       // stop if there were no swaps
      break;
  }
}
```

Sorting in Quadratic Time

Sorting problem: start with an unordered array of elements and rearrange them into nondecreasing order

- Selection Sort
- Insertion Sort
- Bubble Sort

All three algorithms have quadratic complexity, i.e. $O(n^2)$.

Summary

Reading

Section 3.1 Using Arrays

Questions?