UBC OKANAGAN

COSC 121

Computer Programming II

Sorting

Part 1/2

Dr. Mostafa Mohamed

Acknowledgement: The slides mainly rely on the textbook of Y. D. Liang titled "Introduction to Java Programming, 10th Ed.", Pearson Edu. Inc.

Outline

Today:

- Algorithms Efficiency
- Best, Average, and Worst Cases
- Simple Sort Techniques
  - Selection Sort
  - Bubble Sort
  - Insertion Sort

Next lecture:

- More Advanced Sort Techniques
  - Merge Sort
  - Quick Sort
- Positive Integers Sorting: Bucket and Radix Sort

Algorithms Efficiency

Objectives

By the end of this chapter, you should be able to:

- Recognize that different algorithms for the same problem may be **significantly different in terms of their performance**.
- Analyze time complexity of various sorting algorithms
- Implement and analyze simple sorting techniques:
  - Bubble, selection, and insertion.
- Explain more complex sorting techniques:
  - Quick and merge
- Explain specialized sorting algorithm
  - bucket and radix

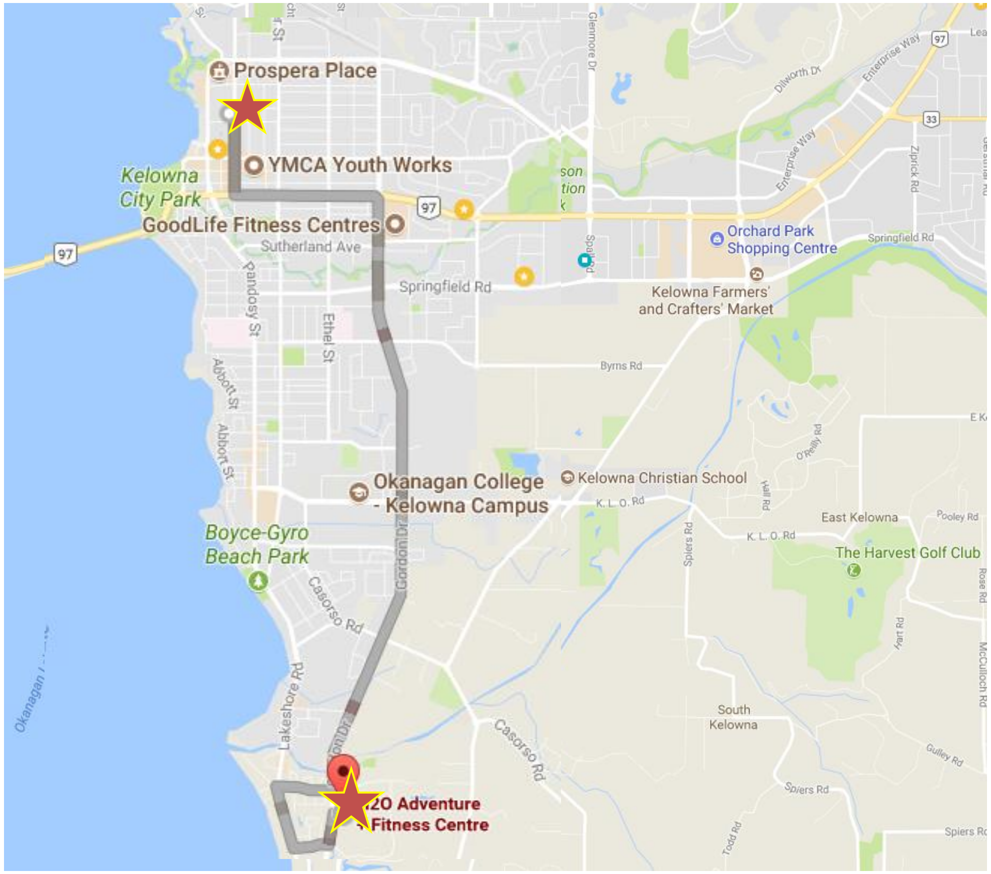
Why are some programs faster than others?

Recall that an ***algorithm*** is a sequence of steps to solve a problem.

<u>Example1; Routing Problem</u>

Problem

Go from Okanagan Library to H2O



Desired Output

One route from Lib to H2O

Algorithm

1) move south on Ellis St.

2) turn left on Hwy33

3) move south on Gordon Dr

etc.

Why are some programs faster than others?

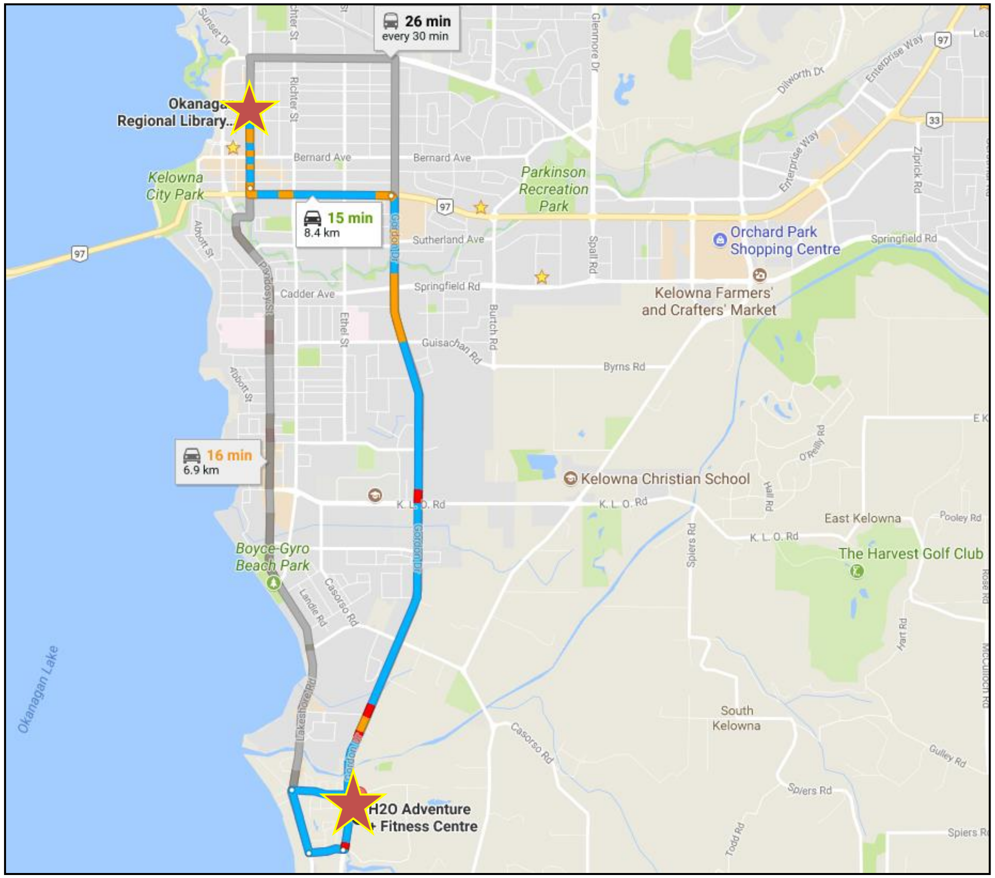
<u>Example1; Routing Problem</u>

More than one algorithm?

- More than one route using more than one means

Several routes can be used. Some **<u>take longer</u>** than others.

- Several algorithms can be used for the same problem
- But some are more efficient than others



Why are some programs faster than others?

Example 2: printing a LinkedList in reverse order

```java
//Loop1:
for (int i = myList.size()-1; i >=0; i--)
    System.out.println(myList.get(i));

//Loop2:
ListIterator<String> itr = myList.listIterator();
while(itr.hasNext())
    itr.next();
while(itr.hasPrevious())
    System.out.print(itr.previous() + " ");
```

- For 100,000 elements, the reported time
  - Algorithm 1: 4.212 sec
  - Algorithm 2: 0.016 sec

Why are some programs faster than others?

Example3: data sorting

- http://www.sorting-algorithms.com/

- http://blocks.org/andrewringler/raw/3809399/

In general, the performance of an algorithm when implemented on a computer depends on the **approach used to solve the problem and the actual steps taken.**

Although faster hardware makes all algorithms faster, algorithms that solve the same problem can be compared in **a hardware-independent way** using ***big-Oh*** notation.

Best, Average, and Worst Cases

Very few algorithms have the exact same performance every time because **the performance of an algorithm typically <u>depends on the size of the inputs</u> it processes.**

The ***best case*** performance of the algorithm is the most efficient execution of the algorithm on the **"best" data inputs**.

The ***worst case*** performance of the algorithm is the least efficient execution of the algorithm on the **"worst" data inputs**.

The ***average case*** performance of the algorithm is the average efficiency of the algorithm on **the set of all data inputs**.

Best, worst, and average-case analysis typically express **efficiency** in terms of the **input size of the data**.

- The input size is often a function of ***n***.

Why study sorting?

Sorting is a classic subject in computer science.

Why you study them?

1. Excellent examples to demonstrate algorithm performance
2. Many creative approaches to problem solving.
   - these approaches can be applied to solve other problems.
3. Good for practicing fundamental programming techniques
   - using selection, loops, methods, and arrays.

Note: Java API has several sort methods in these classes:

- java.util.Arrays
- java.util.Collections

Today's assumptions: what data to sort?

The data to be sorted might be of almost any type: integers, doubles, characters, or objects.

For simplicity, this section assumes:

- data to be sorted are **integers**,
- data are sorted in **ascending order**, and
- data are stored in **an array**.

The programs can be easily modified to sort other types of data, to sort in descending order, or to sort data in an ArrayList or a LinkedList.

Code Used for Testing!

```
public static void main(String[] args) {
    // part1 is used for showing sorted output
    int[] a = {6,2,3,7,4,1,0,9,8,5};
    sort(a); printList(a);
    // part2 is used for demonstrating efficiency (we will time the algorithm)
    int N = 100000;  int[] b = new int[N];    // try different values of N
    initializeRandom(b);                      // try initialize sorted A/D
    double time = System.currentTimeMillis(); // record start time
    sort(b);
    time = System.currentTimeMillis() - time; // compute elapsed time
    System.out.printf("Sorting %d elements took %.3f seconds\n",N,time/1000);
}
static void initializeRandom(int[] a) {
    for(int i=0;i<a.length;i++) a[i] = (int)(Math.random()*a.length -
    a.length/2);
}
static void initializeSortedAssending(int[] a) {
    for(int i=0;i<a.length;i++) a[i] = i;
}
static void initializeSortedDescending(int[] a) {
    for(int i=0;i<a.length;i++) a[i] = a.length-i-1;
}
static void printList(int[] a) {
    for(int i = 0; i<a.length; i++)   System.out.printf("%-3d", a[i]);
    System.out.println();}
```

Demonstration

Here are a few websites that can be used to demonstrate the different sorting algorithms in this unit.

For comparison:

- https://www.toptal.com/developers/sorting-algorithms

For visualization

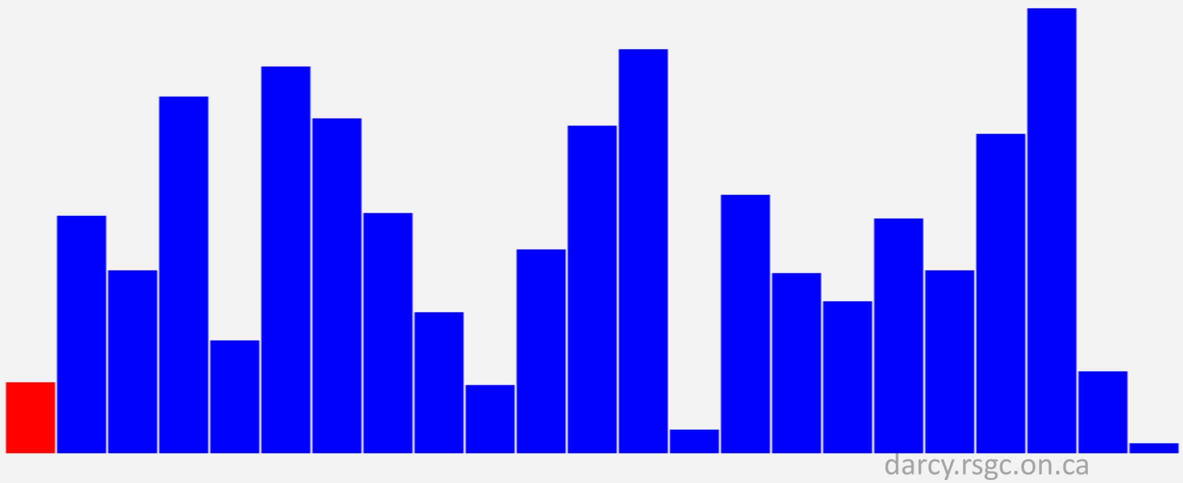
- https://visualgo.net/en/sorting
- https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Textbook demos include:

- http://cs.armstrong.edu/liang/animation/web/SelectionSort.html
- http://cs.armstrong.edu/liang/animation/web/InsertionSort.html

Simple Sort Techniques

- Selection Sort
- Insertion Sort
- Bubble Sort



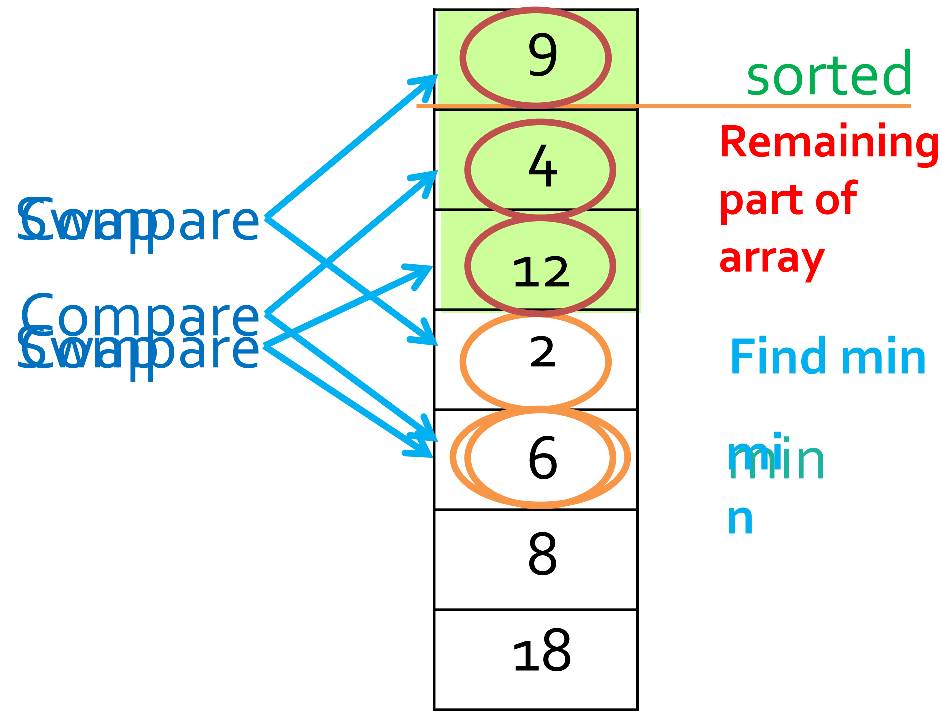
Selection Sort

Selection Sort

Algorithm:

For each element in the list:

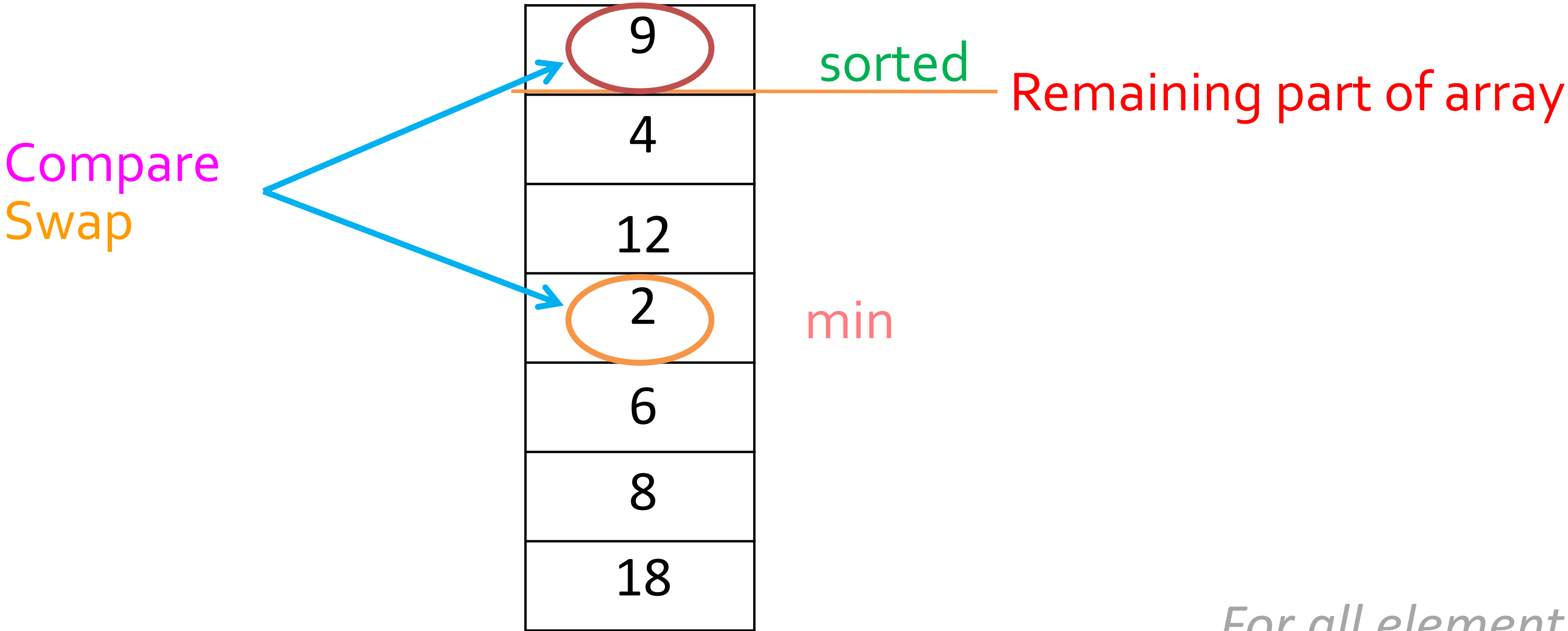
- Find the smallest element
- Swap it with the element.
- Repeat with the unsorted part.
  - Ignore the first element and apply the same algorithm on the remaining smaller list.



Runtime efficiency

- **Worst/Average/Best case: O(n²)** 2 nested loops

cs.armstrong.edu/liang/animation/web/SelectionSort.html




```
for (int index = 0; index < list.length; index++) {
      minIdx = index;
      for (int scan = index+1; scan < list.length; scan++)
         if (list[scan] < list[min])
               minIdx = scan;
      temp = list[minIdx];
      list[minIdx] = list[index];
      list[index] = temp;
}
```

For all elements in the array

Find min in remaining part

swap

Selection Sort: Implementation

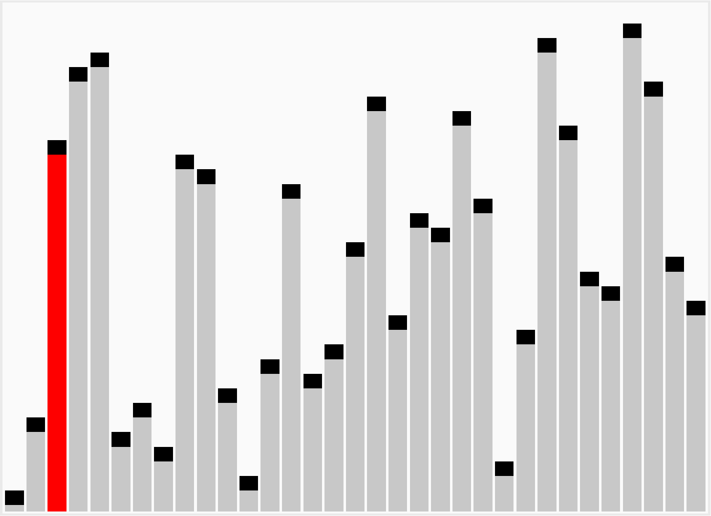
Using two nested loops

```
private void selectionSort(int[] list) {
    for (int index = 0; index < list.length; index++) {
        // find the smallest element
        int idxMin = index;
        for (int scan = index + 1; scan < list.length; scan++)
            if (list[idxMin] > list[scan])
                idxMin = scan;
        // swap the element at i with the smallest element
        int temp = list[index];
        list[index] = list[idxMin];
        list[idxMin] = temp;
    }
}
```

Selection Sort: Implementation, cont.

Remember: Recursive Selection Sort

```
    public static void sort(int[] list) {
        sort(list, 0, list.length - 1); // Sort the entire list
    }
    private static void sort(int[] list, int low, int high) {
        if (low < high) {
            // Find the smallest number and its index in list[low .. high]
            int indexOfMin = low;
            int min = list[low];
            for (int i = low + 1; i <= high; i++)
                if (list[i] < min) {
                    min = list[i];
                    indexOfMin = i;
                }
            // Swap the smallest in list[low .. high] with list[low]
            list[indexOfMin] = list[low];
            list[low] = min;
            // Sort the remaining list[low+1 .. high]
            sort(list, low + 1, high);
        } //else if (low == high) return; //stopping cond: list has only one element
    }
}
```



Wikipedia

Bubble Sort

Bubble Sort

IDEA: Starting from the first item, compare adjacent items and keep “bubbling” the larger one to the right. Repeat for remaining sublist.

Wikipedia

Algorithm:

- Starting on the left: for each pair of elements in the list, swap them if they are not in order. The largest element is bubbled to position N.
- Start on the left again, and bubble the second largest element to position N-1.
- And so on.

Runtime efficiency

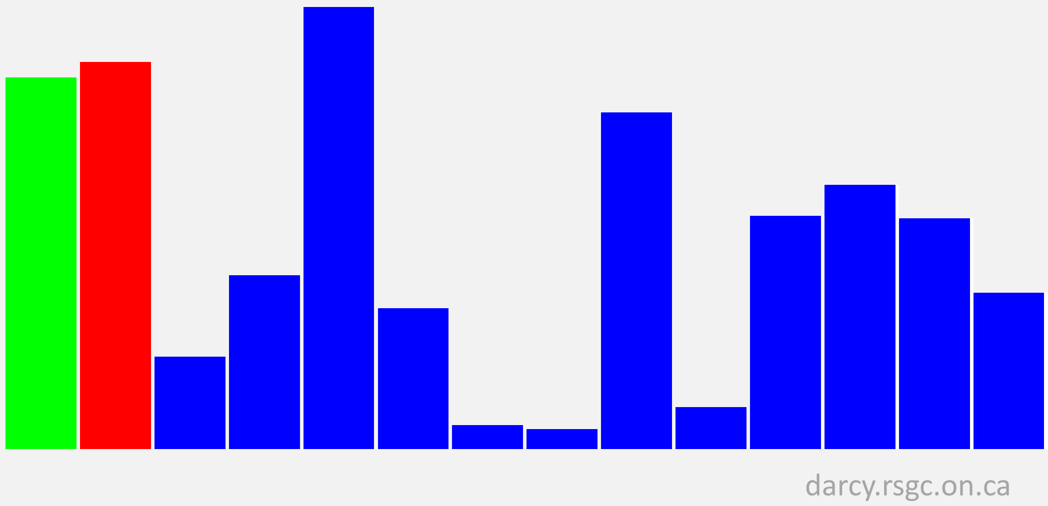
- **Worst /Average case: O(n^2)** 2 nested loops
- **Best case: O(n)** for almost sorted list

Bubble Sort Code

```
public static void bubbleSort(int[] list){

    // repeat a number of passes equal to list length
    for (int pass = 0; pass < list.length ; pass ++) {
        // at beginning of every pass, assume list is sorted

        // for each pair of elements, swap if not in order
        for (int i = 0; i < list.length-pass-1; i++) {
            if(list[i]>list[i+1]){
                int temp = list[i];
                list[i] = list[i+1];
                list[i+1]=temp;


            }
        }
    }
}
```

Improved Bubble Sort Code

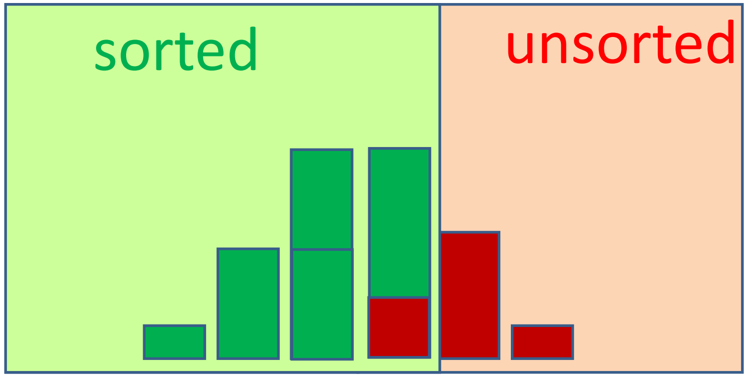
```
public static void bubbleSort(int[] list){
    boolean sorted = false;    // list is not sorted.
    // repeat a number of passes equal to list length
    for (int pass = 0; pass < list.length && !sorted; pass ++) {
        // at beginning of every pass, assume list is sorted
        sorted = true;
        // for each pair of elements, swap if not in order
        for (int i = 0; i < list.length-pass-1; i++) {
            if(list[i]>list[i+1]){
                int temp = list[i];
                list[i] = list[i+1];
                list[i+1]=temp;
                //if we ever need to swap, then list is unsorted; we need another pass
                sorted = false;
            }
        }
    }
}
```



Insertion Sort

Insertion Sort

IDEA: start with a sorted list of 1 element. Repeatedly insert an unsorted element into a sorted sublist until the whole list is sorted.



Algorithm:

- For each element **e** in the unsorted part,
  - Keep a copy of **e**
  - Insert **e** it into in the sorted list such that this list remains sorted.
    - By moving all elements larger than **e** forward by one step.

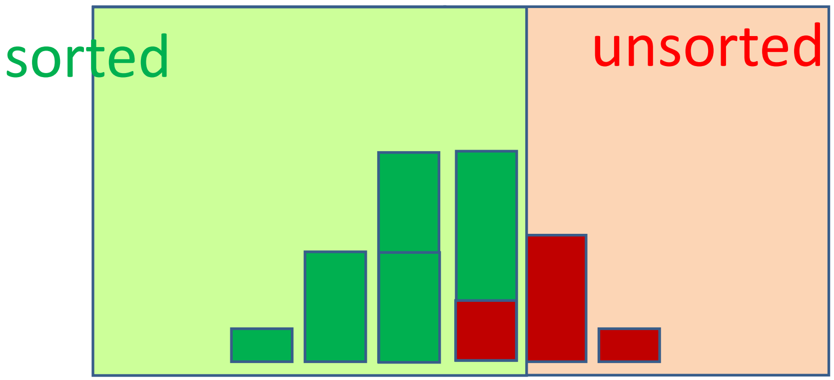
Runtime efficiency

- **Worst/Average case: $O(n^2)$** 2 nested loops
- **Best case: O(n)** for nearly sorted list

http://cs.armstrong.edu/liang/animation/web/InsertionSort.html

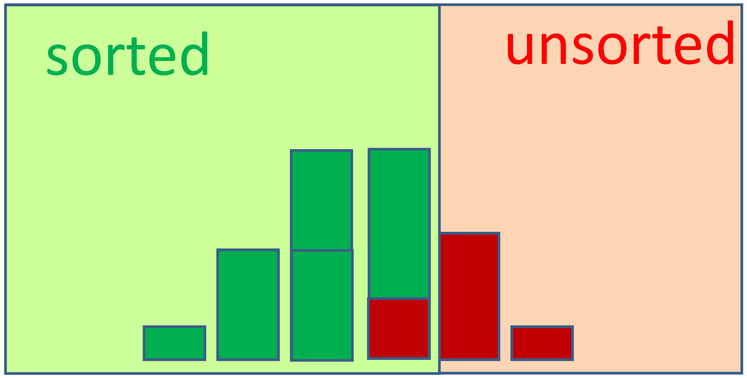
Insertion Sort

```
public static void insertionSort(int[] list){
   for(int i=1; i<list.length; i++){
      // Remember first item in the unsorted list
      int item = list[i];
      // for each previous element
      int pos;
      for(pos = i; pos > 0; pos--)               //if previous element is larger than item
         if(list[pos-1]>item) list[pos]=list[pos-1]; //shift-right prev element
         else                 break;             //otherwise, we found right insertion point
      // Place item in its correct position
      list[pos] = item;
   }
}
```



Insertion Sort, another solution

```
public static void insertionSort(int[] list){
  for(int i=1; i<list.length; i++){
     //Remember first item in the unsorted list
     int item = list[i];
     // Shift previous elements > item to the right
     // and find the correct position for the item in the sorted list
     int pos;
     for(pos = i; pos > 0 && list[pos-1]>item ; pos--)
        list[pos] = list[pos-1];
     // Place item in its correct position
     list[pos] = item;
  }
}
```



Insertion Sort

```java
public static void insertionSort(int[] list){
  for(int i=1; i<list.length; i++){
    //Remember first item in the unsorted list
    int item = list[i];
    // Shift previous elements > item to the right
    // and find the correct position for the item in the sorted list
    int pos = i;
    while(pos > 0 && item < list[pos-1]){
      list[pos] = list[pos-1];
      pos--;
    }
    // Place item in its correct position
    list[pos] = item;
  }
}
```

Same code using a while loop