Sorting

Savitch Chapter 7.4

Why sort

- Easier to search (binary search)
- Sorting used as a step in many algorithms

Sorting algorithms

- There are many algorithms for sorting:
  - **Selection sort**
  - **Insertion sort**
  - **Bubble sort**
  - Merge sort
  - Heap sort
  - Radix sort
  - Quick sort
  - **Stooge sort**
- Each has its advantages and disadvantages

Selection Sort

- Find the smallest item
- Put it in the first position
  - Find the 2nd smallest item
  - Put it in the 2nd position
    - Find the 3rd smallest item
    - Put it in the 3rd position

  ….

Selection Sort code

```
public void selectionSort (Comparable [] array){
   int min;
   for (int i = 0; i < array.length-1; i++) {
      min = i;
      for (int j = i+1; j < array.length; j++){
         if (array[j].compareTo(array[min]) < 0)
            min = j;
      }
      swap (array, min, i);
   }
}
private void swap(Comparable[] array, int i, int j){
   Comparable temp = array[i];
   array[i] = array[j];
   array[j] = temp;
}
```

Selection Sort code

```
public void selectionSort (Comparable [] array){
   int min;
   for (int i = 0; i < array.length-1; i++) {
      min = i;
      for (int j = i+1; j < array.length; j++){
         if (array[j].compareTo(array[min]) < 0)
            min = j;
      }
      swap (array, min, i);
   }
}
private void swap(Comparable[] array, int i, int j){
   Comparable temp = array[i];
   array[i] = array[j];
   array[j] = temp;
}
```

outer **loop**

inner **loop**

Loop Invariant for Selection Sort

```
public void selectionSort (Comparable [] array){
   int min;
   for (int i = 0; i < array.length-1; i++) {
      min = i;
      for (int j = i+1; j < array.length; j++){
         if (array[j].compareTo(array[min]) < 0)
            min = j;
      }
      swap (array, min, i);
   }
}
```

Invariants?

Loop Invariant for Selection Sort

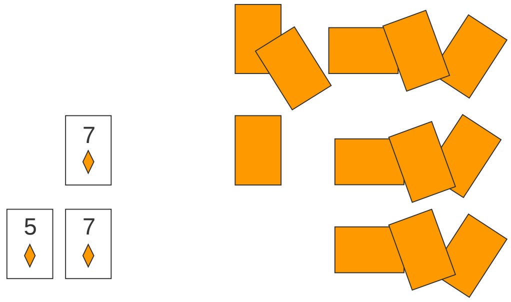
```
public void selectionSort (Comparable [] array){
   int min;
   for (int i = 0; i < array.length-1; i++) {
      min = i;
      for (int j = i+1; j < array.length; j++){
         if (array[j].compareTo(array[min]) < 0)
            min = j;
      }
      swap (array, min, i);
   }
}
```

Invariant: The elements array[0..i] are in sorted order

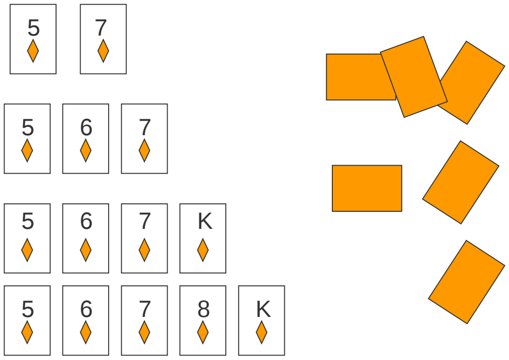
Insertion sort

- Works the same way you arrange your hand when playing cards.
  - Pick up a card and place it in your hand in the correct position relative to the cards you’re already holding.

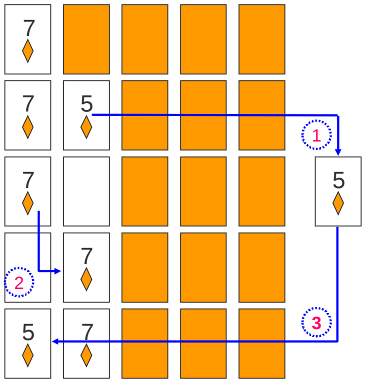
Arranging a hand of cards



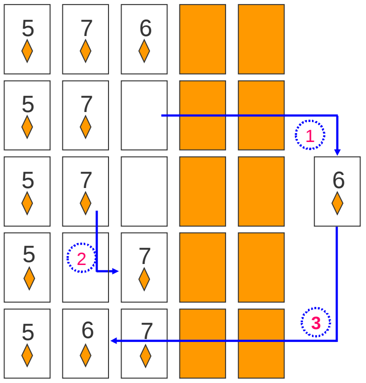
Arranging a hand of cards



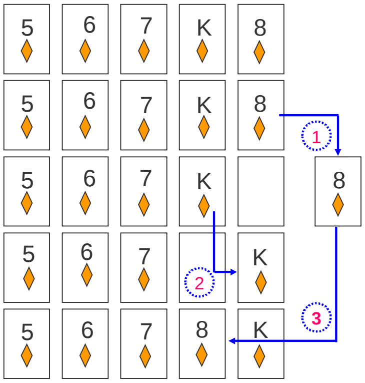
Insertion Sort



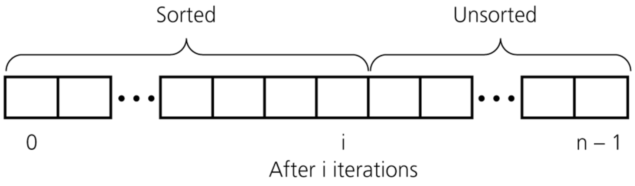
Insertion Sort (cont.)



Insertion Sort (cont.)

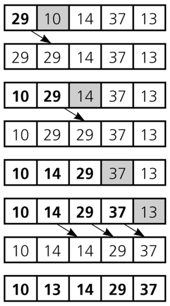


Insertion Sort – more formally



- insertion sort partitions the array into two regions: sorted, and unsorted
- *each iteration the sorted part grows by 1*

Insertion Sort – another example



Initial array:	**29**	10	14	37	13	Copy 10
	29	29	14	37	13	Shift 29
	10	**29**	14	37	13	Insert 10; copy 14
	10	29	29	37	13	Shift 29
	10	**14**	**29**	37	13	Insert 14; copy 37, insert 37 on top of itself
	10	**14**	**29**	**37**	13	Copy 13
	10	14	14	29	37	Shift 37, 29, 14
Sorted array:	**10**	**13**	**14**	**29**	**37**	Insert 13

Insertion Sort Algorithm

```
public void insertionSort(Comparable[] array) {

   for (int i = 1; i < array.length; i++) {
      Comparable temp = array[i];
      int position = i;

      // shift larger values to the right
      while (position > 0 &&
              array[position-1].compareTo(temp) > 0) {
         array[position] = array[position-1];
         position--;
      }
      // insert the current item
      array[position] = temp;
   }
}
```

With a for loop

```
public void insertionSort(Comparable[] array) {

   for (int i = 1; i < array.length; i++) {
      Comparable temp = array[i];

      // shift larger values to the right
      for (int position = i;
             position > 0 &&
              array[position-1].compareTo(temp) > 0;
                                             position--) {
         array[position] = array[position-1];
      }
      // insert the current item
      array[position] = temp;
   }
}
```

Insertion Sort Algorithm

```
public void insertionSort(Comparable[] array) {
   for (int i = 1; i < array.length; i++) {
      Comparable temp = array[i];
      int position = i;
      // shift larger values to the right
      while (position > 0 &&
              array[position-1].compareTo(temp) > 0) {
         array[position] = array[position-1];
         position--;
      }
      // insert the current item
      array[position] = temp;
   }
}
```

outer loop

inner loop

Loop Invariant for Insertion Sort

```
public void insertionSort(Comparable[] array) {
   for (int i = 1; i < array.length; i++) {
      Comparable temp = array[i];
      int position = i;
      while (position > 0 &&
              array[position-1].compareTo(temp) > 0) {
         array[position] = array[position-1];
         position--;
      }
      array[position] = temp;
   }
}
```

Invariant: *array[0…i-1] consists of elements originally in array[0…i-1] but in sorted order*

Loop Invariant for Insertion Sort

```
public void insertionSort(Comparable[] array) {
   for (int i = 1; i < array.length; i++) {
      Comparable temp = array[i];
      int position = i;
      while (position > 0 &&
             array[position-1].compareTo(temp) > 0) {
         array[position] = array[position–1];
         position--;
      }
      array[position] = temp;
   }
}
```

Invariant: *array[0…i-1] consists of elements originally in array[0…i-1] but in sorted order*

How is this different than in Selection Sort?

Sorting Linked Lists

- Accessing an element in a linked list takes time.
- Can you sort a linked list with Selection Sort or Insertion Sort maintaining the same level of efficiency as using arrays?

Bubble Sort

```
public void bubbleSort (Comparable [] array) {

   for (int position = array.length-1; position>=0;
                                       position--) {
      for (int i = 0 ; i < position; i++) {
         if (array[i].compareTo(array[i+1]) > 0)
            swap(array, i, i+1);
      }
   }
}
```

Bubble Sort

- Compares neighboring elements, and swaps them if they are not in order
  - Effect: the largest value will “bubble” to the last position in the array.
  - Repeating the process will bubble the 2nd to largest value to the 2nd to last position in the array

Bubble Sort

- Compares neighboring elements, and swaps them if they are not in order
  - Effect: the largest value will “bubble” to the last position in the array.
  - Repeating the process will bubble the 2nd to largest value to the 2nd to last position in the array

Loop Invariant: After i iterations the largest i elements are in their correct sorted position

Bubble Sort

```
public void bubbleSort (Comparable [] array) {
  for (int position = array.length-1; position>=0;
                                          position--) {
         for (int i = 0 ; i < position; i++) {
            if (array[i].compareTo(array[i+1]) > 0)
               swap(array, i, i+1);
         }
   }
}
```

outer loop

inner loop

Bubble Sort

```
public void bubbleSort (Comparable [] array) {
   for (int position = array.length-1; position>=0;
                                         position--) {
      for (int i = 0 ; i < position; i++) {
         if (array[i].compareTo(array[i+1]) > 0)
            swap(array, i, i+1);
      }
   }
}
```

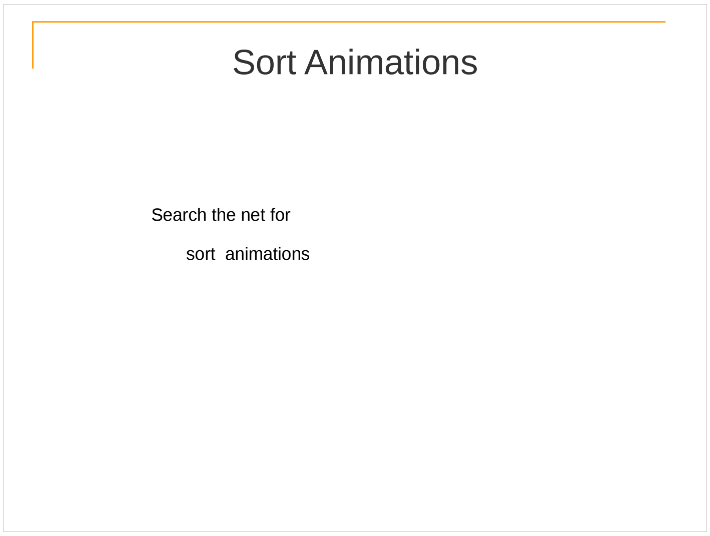
Inner Invariant: array[i] is the largest element in the first i elements in the array

Outer Invariant: After i iterations the largest i elements are in their correct sorted position

Stooge Sort

```
public void stoogeSort(Comparable [] array, int i, int j) {
   if (array[i].compareTo(array[j]) > 0 ) {
      swap(array, i, j);
   }
   if (j – i > 1) {
      int third = (j – i + 1) / 3;
      stoogeSort(array, i, j-third);  //first two thirds
      stoogeSort(array, i + third, j);//second two thirds
      stoogeSort(array, i, j-third);  //first two thirds
   }
}

public void stoogeSort(Comparable [] array) {
   stoogeSort(array, 0, array.length – 1);
}
```



Sort Animations

Search the net for

sort animations