COMPSCI 105

Principles of Computer Science

Sorting algorithms

Sorting

Many different methods of sorting exist

- Insertion sort
- Selection sort
- Bubble sort
- Merge sort
- Quick sort

Trade-offs

- Simplicity
- Performance

Considerations

- Memory
- Number of Comparisons
- Number of Swaps
- Best performance, average performance, worst performance

Swapping

A common operation when sorting.

- swaps two elements over
- fast and easy with arrays

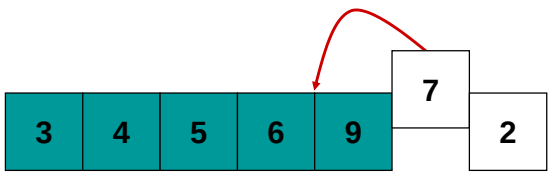
```
public void swap(int data[], int i, int j) {
        int temp;
        temp = data[i];
        data[i] = data[j];
        data[j] = temp;
}
```


Insertion sort

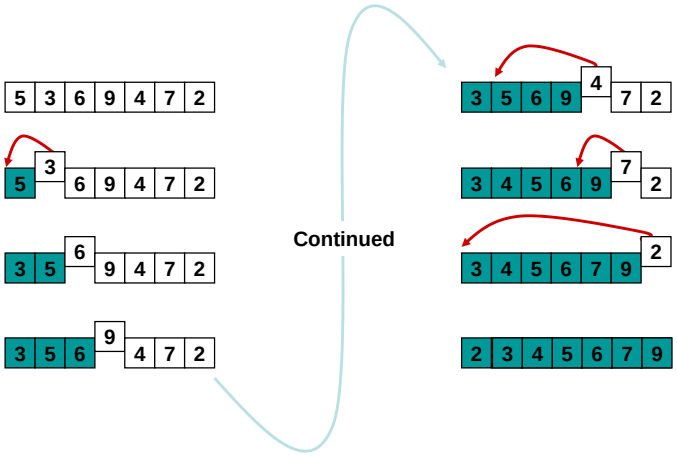
Each element is inserted into a sorted list

Imagine a set of cards

- Start sorting from one end
- Keep the sorted cards and unsorted cards separate
- Take the next unsorted card and insert into the sorted cards



Insertion sort



Insertion Sort

for each element

insert the element into a sorted list

```
public void insertionSort(int[] data){
       for (int unsorted = 1; unsorted < data.length; unsorted++)
              insert(data, unsorted);
}


public void insert(int[] data, int n){
       //inserts the nth element into the sorted elements

       for (int j = n; j > 0 && data[j] < data[j-1]; j--)
              swap(data, j, j-1);
}
```


Insertion sort

Easiest of the sorting methods

- Which code is better and why?
- What is the big-O complexity?

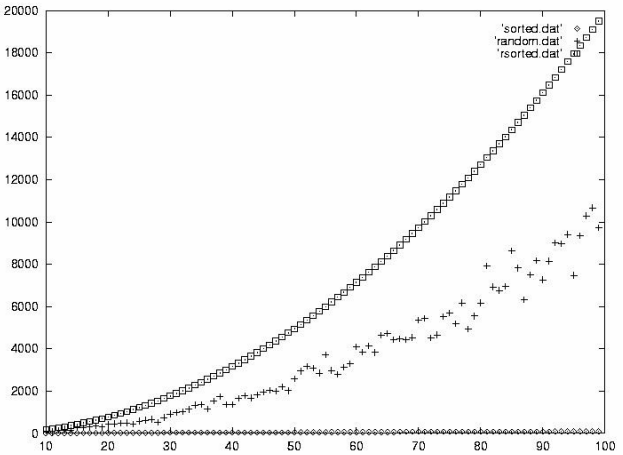
Which is better and why?

```
public void insertionSort(int data[]){
       for(int i=1; i<data.length; i++)
              for(int j=i; j>0; j--)
                     if (data[j] < data[j-1])
                            swap(data, j, j-1);
}
```

```
public void insertionSort(int data[]){
       for(int i=1; i<data.length; i++)
              for(int j=i; j>0 && data[j] < data[j-1]; j--)
                     swap(data, j, j-1);
}
```

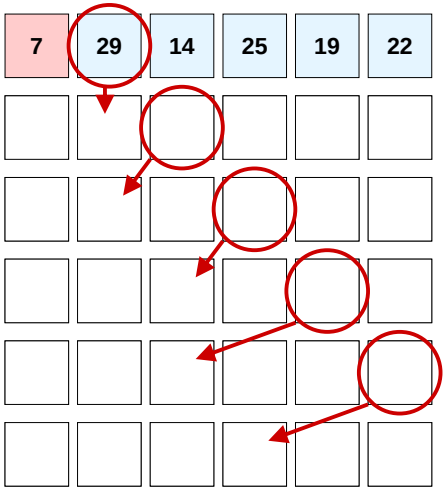

Insertion Sort performance

.



Exercise: Insertion Sort

Show the state of the array after each iteration of the outer loop:



Recursive Insertion sort

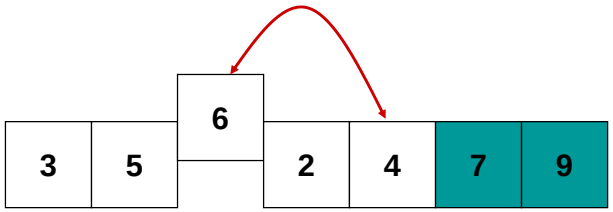
```
public void insertionSort(int data[]){
        //A simple wrapper to hide the recursive call
        insertSort(data, data.length-1);
}

public void insertSort(int data[], int i){
        if (i==0)                       //base case
                return;
        insertSort(data, i-1);          //sort the rest of the data
        insert(data, i);                //insert data[i] in place
}

public void insert(int data[], int j){
        if (j>0 && data[j] < data[j-1]){
                swap(data, j, j-1);
                insert(data, j-1);
        }
}
```


Selection sort

The next largest element is put in correct place

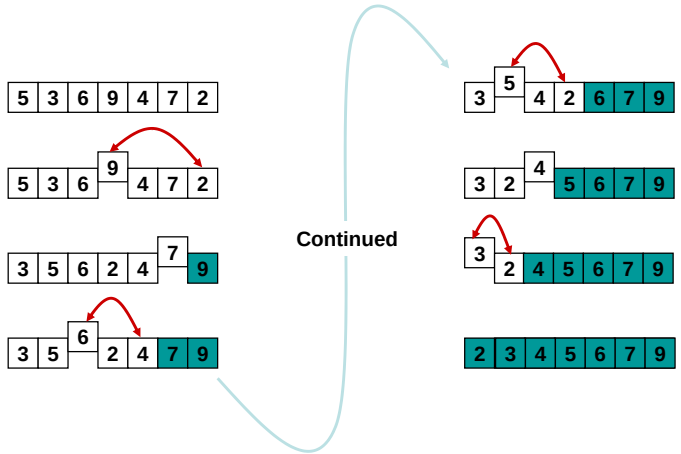
Imagine a set of cards

- Select the biggest card from the unsorted part
- Swap it with the end of the unsorted part



Selection sort

.



Selection sort

while not finished
  select biggest unsorted element
  swap with last unsorted element

```
public void selectionSort(int[] data){
	for(int i=data.length-1; i>0; i--){
		int maxIndex = getMaximum(data, i);
		swap(data, i, maxIndex);
	}
}

public int getMaximum(int[] data, int n){
	int max = 0;
	for(int j=0; j<=n; j++)
		if (data[max] < data[j])
			max = j;
	return max;
}
```


Selection sort

What is the big-O complexity?

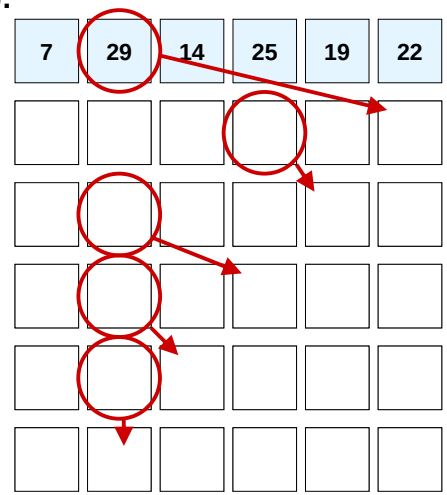
Compare with insertion sort

- Under what conditions would this sort be better?
- Under what conditions would this sort be worse?

```
public void selectionSort(int[] data){
	for(int i = data.length-1; i > 0; i--){
		int maxIndex = 0;
		for(int j = 0; j <= i; j++) {
			if (data[maxIndex] < data[j]) {
				maxIndex = j;
			}
		}
		swap(data, i, maxIndex);
	}
}
```


Exercise: Selection Sort

Show the state of the array after each iteration of the outer loop:



Recursive Selection sort

.

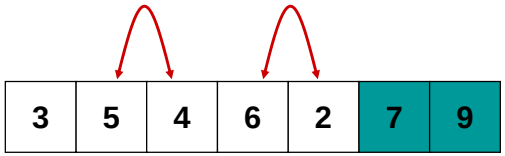
```
public static void selectionSort3(int[] data){
	selection(data, data.length-1);
}

public static void selection(int[] data, int n){
	if(n>0){
		int max = getMax(data, n);
		swap(data, n, max);
		selection(data, n-1);
	}
}

public static int getMax(int[] data, int n){
	if(n==0)
		return 0;
	else{
		int max = getMax(data, n-1);
		return (data[max] > data[n]) ? max : n;
	}
}
```

Bubblesort

Swap adjacent values if they are out of order

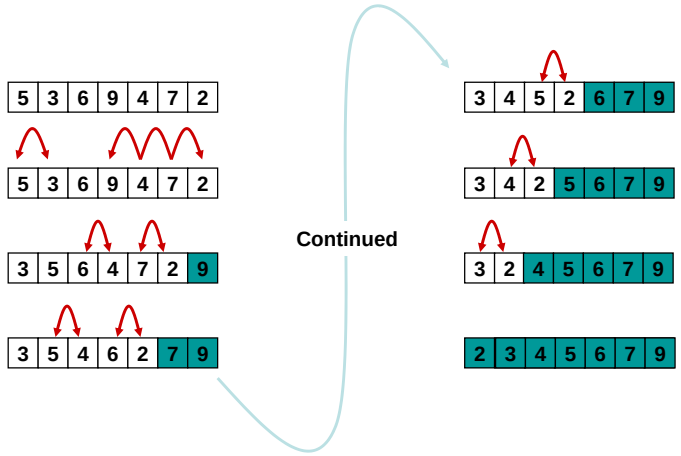
Imagine a deck of cards

- start at one end
- compare adjacent cards
- swap if they are out of order
- the largest value will "bubble" up to the end



Bubblesort

.



Bubble sort

while entire array hasn't been sorted
for each element in unsorted section
swap with adjacent value if required

```
public static void bubbleSort(int[] data){
	for(int i=data.length-1; i>0; i--)
		bubbleUp(data, i);
}


public static void bubbleUp(int data[], int n){
	for(int j=0;j<n; j++)
		if(data[j] > data[j+1])
			swap(data, j, j+1);
}
```


Bubblesort

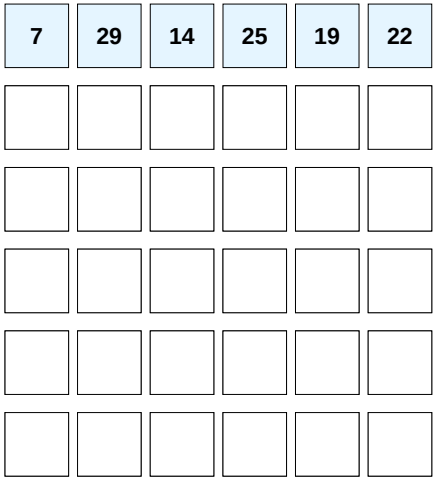
What is the time complexity of this method (use big-O)?

How could we improve the algorithm in case the data is sorted?

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

Exercise: Bubble sort

Show the state of the array after each iteration of the outer loop:



Recursive Bubblesort

.

```
public void bubbleSort(int data[]){
        bubbleS(data, data.length-1);
}

public void bubbleS(int[] data, int n){
        if(n>0){
                bubbleUp(data, n);
                bubbleS(data, n-1);
        }
}

public void bubbleUp(int[] data, int n){
        if(n>0){
                bubbleUp(data, n-1);
                if(data[n-1] > data[n])
                        swap(data, n-1, n);
        }
}
```


Debugging code

Print out the contents of the array

- Before sorting
- During sorting
- After sorting

```
public static void printArray(int[] data){
        for(int i=0; i<data.length; i++)
                out.print(data[i] + " ");
        System.out.println();
}
```