

# Computer Science 210: Data Structures

## Sorting

# Sorting

- Given a sequence of elements, sort them
- More precisely:
  - Input:
    - an array  $a = [a_0, a_1, a_2, \dots]$
    - the elements are comparable to each other (we can ask whether  $a[i] < a[j]$  )
  - Output:
    - the same array  $a$ , rearranged (permuted) so that the elements are in increasing order:  
 $a[0] \leq a[1] \leq a[2] \leq \dots$
    - OR, a new array  $b$  that contains the elements of  $a$  in increasing order:  
 $b[0] \leq b[1] \leq b[2] \dots$
- Some sorting algorithms can rearrange the input array, others need to “create” a new array
- Permuting the input is advantageous because it does not use extra memory
- A sorting algorithm that permutes the input array and does not need a new output array is called “in-place”

# Sorting algorithms

- Bubble-sort

- idea: a pass through the array swaps the elements that are out of order

- Insertion sort

- idea: think of sorting a deck of cards. take one element at a time, and insert it “in hand” in the right place.

- Selection sort

- select the smallest, put it in position 0; select the smallest of the remaining elements, put it in position 1; and so on.

- Mergesort

- divide in half, sort each half recursively, and merge

- Quicksort

- pick an element and call it pivot. re-arrange the input so that all elements smaller than the pivot are to its left, and all the elements larger than the pivot are to its right. sort each part recursively. no need of merging.

details in 23 I (Algorithms)



# Bubble-sort

- Idea: a pass through the array swaps the elements that are out of order

```
//assume an array a of n elements:  a[0], ...a[n-1]
```

```
for i=0 to n-2
```

```
    if (a[i] > a[i+1]) then swap a[i], a[i+1]
```

- Is the array sorted after one pass?
- What is true after one pass?
- How many passes do you need?
- Find a worst-case example
- Sketch the (pseudo)code for bubblesort
- Analyze it

# Bubblesort

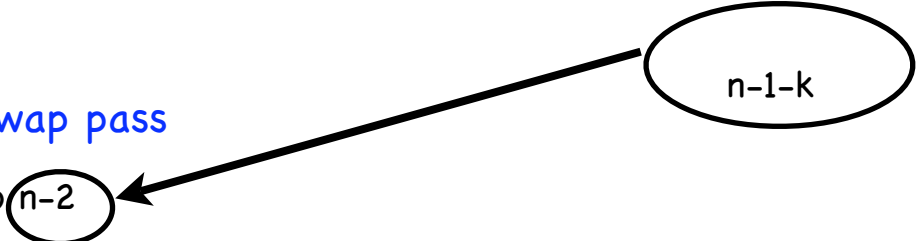
```
//assume an array a of n elements: a[0], ...,a[n-1]
```

```
for k=1 to n-1
```

```
  //do a swap pass
```

```
  for i=0 to n-2
```

```
    if (a[i] > a[i+1]) then swap a[i], a[i+1]
```



n-1-k

- This always performs n-1 passes
- Can be refined to exit early if the array is sorted

# Selection sort

- Idea: select the smallest, put it in position 0; select the smallest of the remaining elements, put it in position 1; and so on.

- *//assume an array a of n elements*
- for i=0 to n-2
  - let k = the index of the smallest element among  $a[i], a[i+1], \dots, a[n-1]$
  - swap  $a[i]$  with  $a[k]$

- Analysis
  - Worst-case?
  - Best-case?

# Insertion sort

- Idea: think of sorting a deck of cards. Take one element at a time, and insert it “in hand” in the right place.

```
for i=1 to n-1
```

```
    //invariant: a[0]...a[i-1] is sorted
```

```
    shift a[i] to its correct place so that a[0]...a[i] is sorted
```

- Analysis
  - Best case?
  - Worst-case?

# Sorting summary

- We looked at 3 sorting algorithms:
  - Bubble sort
  - Selection sort
  - Insertion sort
- Each one does two nested loops
  - number of instruction: order  $n^2$
- Best-cases are different
- We'll analyze their complexity formally next week
  - big-oh notation