Today's Material

- Iterative Sorting Algorithms
  - Sorting - Definitions
  - Bubble Sort
  - Selection Sort
  - Insertion Sort

Sorting - Definitions

- Input: You are given an array *A* of data records, each with a key (which could be an integer, character, string, etc.).
  - There is an *ordering* on the set of possible keys
  - You can compare any two keys using <, >, ==
- For simplicity, we will assume that *A*[i] contains only one element – the key
- Sorting Problem: Given an array *A*, output *A* such that:
  - For any i and j, if i < j then *A*[i] <= *A*[j] (ascending order)
  - For any i and j, if i < j then *A*[i] >= *A*[j] (descending order)
- *Internal sorting*: all data in main memory
- *External sorting:* data on disk

Why Sort?

- Sorting algorithms are among the most frequently used algorithms in computer science
  - Crucial for efficient retrieval and processing of large volumes of data, e.g., Database systems
  - Typically a first step in some more complex algorithm
    - An initial stage in organizing data for faster retrieval
- Allows binary search of an N-element array in O(log N) time
- Allows O(1) time access to k^{th} largest element in the array for any *k*
- Allows easy detection of any duplicates

Sorting – Things to consider

- Space: Does the sorting algorithm require extra memory to sort the collection of items?
  - Do you need to copy and temporarily store some subset of the keys/data records?
  - An algorithm which requires O(1) extra space is known as an **in place** sorting algorithm

- Stability: Does it rearrange the order of input data records which have the same key value (duplicates)?
  - E.g. Given: Phone book sorted by name. Now sort by district – Is the list still sorted by name within each county?
  - Extremely important property for databases – next slide
  - A **stable sorting algorithm** is one which does not rearrange the order of duplicate keys

Stability – Why?

- Consider the following data records sorted by name. Second field is the student's class (1st year, 2nd year)

  (Ali, 1), (Mehmet, 2) (Nazan, 1), (Selim, 1), (Zeynep, 2)

- Now sort this set with respect to class

  (Ali, 1), (Nazan, 1), (Selim, 1), (Mehmet, 2) (Zeynep, 2)
  - The set is sorted with respect to class
  - And students are sorted with respect to name within each class.
  - This is the output by a STABLE sorting algorithm

  (Nazan, 1), (Ali, 1), (Selim, 1), (Zeynep, 2) (Mehmet, 2)
  - The set is sorted with respect to class
  - But students are NOT sorted with respect to name within the class
  - This is the output by a UNSTABLE sorting algorithm

Basic Sorting Algorithms

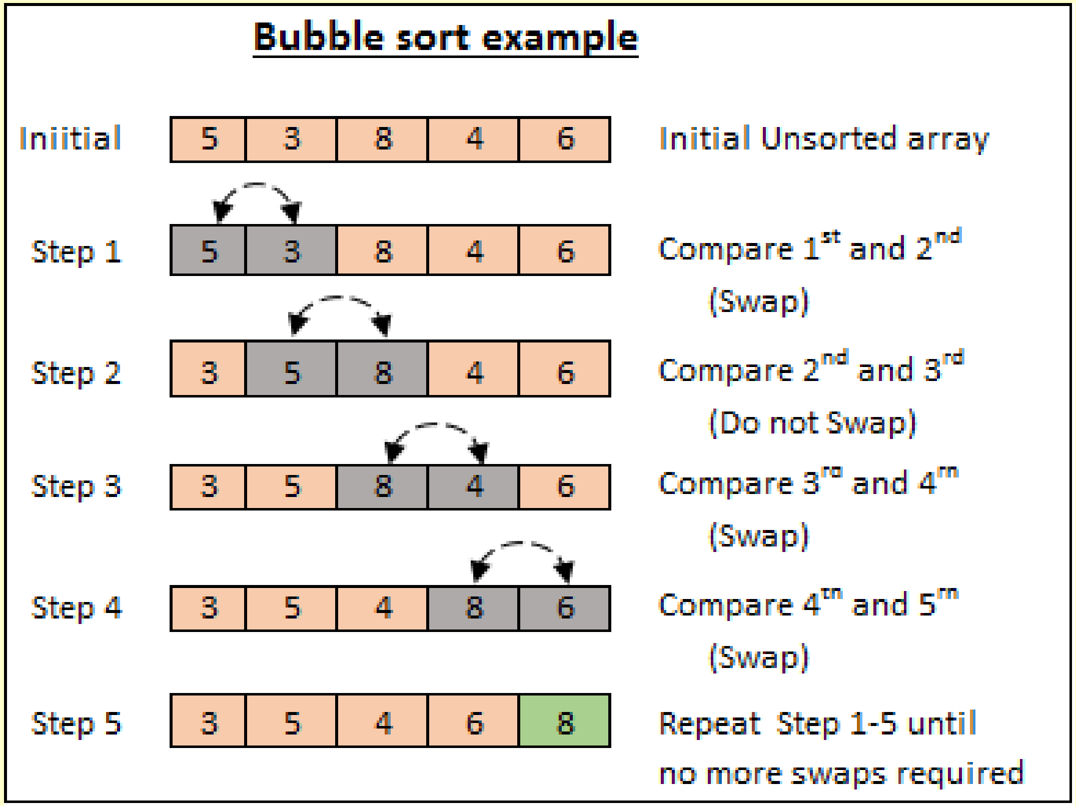
- Bubble Sort
- Selection Sort
- Insertion Sort

Bubble Sort

- Idea: “Bubble” larger elements to end of array by comparing elements i and i+1, and swapping if A[i] > A[i+1]
  - Repeat from first to end of unsorted part
- Example: Sort the following input sequence:
  - 21, 33, 7, 25
  - Start: 21 33 7 25 // Initial array
  - Iteration1: 21 7 25 33 // Moved the max. element to the end (4th slot)
  - Iteration2: 7 21 25 33 // Moved the 2nd max. element to (3rd slot)
  - Iteration3: 7 21 25 33 // Moved the 3rd max. element to (2nd slot)
  - Iteration4: 7 21 25 33 // Moved the 4th max. element to (1st slot)

Bubble Sort

- Example: Sort the following input sequence:
  - 5, 3, 8, 4, 6
  - Start: 5, 3, 8, 4, 6 // Initial array
  - Iteration1:



Bubble Sort

```
/* Bubble sort pseudocode for integers
 * A is an array containing N integers */
BubleSort(int A[], int N){
    for(int i=0; i<N; i++) {
        /* From start to the end of unsorted part */
        for(int j=1; j<(N-i); j++) {
            /* If adjacent items out of order, swap */
            if( A[j-1] > A[j] ) SWAP(A[j-1],A[j]);
        } //end-for-inner
    } //end-for-outer
} //end-BubbleSort
```

- Stable? Yes
- In place? Yes
- Running time = $O(N^2)$

Bubble Sort (Exercise 1)

- Exercise: Sort the following input sequence:
  - 30, 1, 4, 9, 16, 3, 2
  - Start: 30, 1, 4, 9, 16, 3, 2 // Initial array
  - (What are the outputs of Iteration 1 and Iteration 2?)
  - Iteration1:
  - Iteration2:

Bubble Sort (Exercise 1)

- Exercise: Sort the following input sequence:
  - 30, 1, 4, 9, 16, 3, 2
  - Start: 30, 1, 4, 9, 16, 3, 2 // Initial array
  - (What are the outputs of Iteration 1 and Iteration 2?)
  - Iteration1: 1, 4, 9, 16, 3, 2, 30
  - Iteration2: 1, 4, 9, 3, 2, 16, 30

Bubble Sort (Exercise 2)

- Exercise: Sort the following input sequence:
  - 9, 2, 7, 3, 5, 20, 16, 4
  - Start: 9, 2, 7, 3, 5, 20, 16, 4 // Initial array
  - (What are the outputs of Iteration 1 and Iteration 2?)
  - Iteration1:
  - Iteration2:

Bubble Sort (Exercise 2)

- Exercise: Sort the following input sequence:
  - 9, 2, 7, 3, 5, 20, 16, 4
  - Start: 9, 2, 7, 3, 5, 20, 16, 4 // Initial array
  - (What are the outputs of Iteration 1 and Iteration 2?)
  - Iteration1: 2, 7, 3, 5, 9, 16, 4, 20
  - Iteration2: 2, 3, 5, 7, 9, 4, 16, 20

Selection Sort

- Bubblesort is stable and in place, but O(N^2) – can we do better by moving items more than 1 slot per step?

- Idea: Scan array and select smallest key, swap with A[1]; scan remaining keys, select smallest and swap with A[2]; repeat until last element is reached.

- Example: Sort the following input sequence:
  - 21, 33, 7, 25

- Is selection sort stable?
  - Suppose you had another 33 instead of 7→ NOT STABLE
- In place?→ Yes.
  - O(1) space for a Temp variable for swapping keys
- Running time?
  - $N + N-1 + N-2 + \ldots 1 = N*(N+1)/2 = O(N^2)$

Selection Sort

- Suppose the array is
  5 3 4 5 2 6 8

  Let's distinguish the two 5's as 5(a) and 5(b) .

- So our array is:
  5(a) 3 4 5(b) 2 6 8

  After iteration 1:
  2 will be swapped with the element in 1st position:

  So our array becomes:
  2 3 4 5(b) 5(a) 6 8

- So we can clearly see that selection sort is **<u>not stable</u>**.

Selection Sort (Exercise 1)

- Exercise: Sort the following input sequence:
  - 30, 1, 4, 9, 16, 3, 2
  - Start: 30, 1, 4, 9, 16, 3, 2 // Initial array
  - (What are the outputs of Iteration 1 and Iteration 2?)
  - Iteration1:
  - Iteration2:

Selection Sort (Exercise 1)

- Exercise: Sort the following input sequence:
  - 30, 1, 4, 9, 16, 3, 2
  - Start: 30, 1, 4, 9, 16, 3, 2 // Initial array
  - (What are the outputs of Iteration 1 and Iteration 2?)
  - Iteration1: 1, 30, 4, 9, 16, 3, 2
  - Iteration2: 1, 2, 4, 9, 16, 3, 30

Selection Sort (Exercise 2)

- Exercise: Sort the following input sequence:
  - 9, 2, 7, 3, 5, 20, 16, 4
  - Start: 9, 2, 7, 3, 5, 20, 16, 4 // Initial array
  - (What are the outputs of Iteration 1 and Iteration 2?)
  - Iteration1:
  - Iteration2:

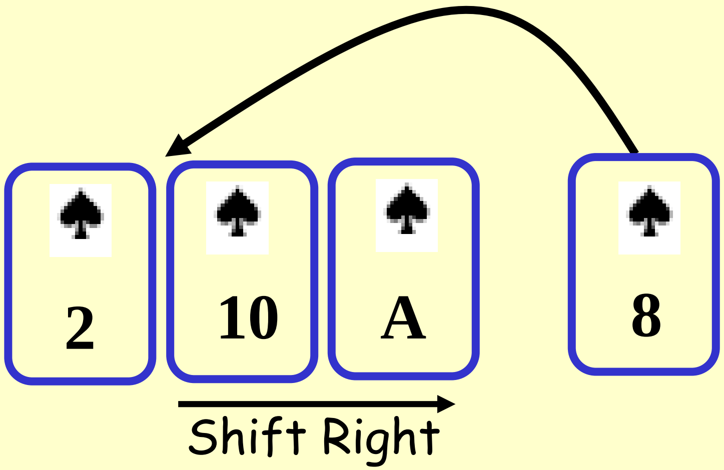
Selection Sort (Exercise 2)

- Exercise: Sort the following input sequence:
  - 9, 2, 7, 3, 5, 20, 16, 4
  - Start: 9, 2, 7, 3, 5, 20, 16, 4 // Initial array
  - (What are the outputs of Iteration 1 and Iteration 2?)
  - Iteration1: 2, 9, 7, 3, 5, 20, 16, 4
  - Iteration2: 2, 3, 7, 9, 5, 20, 16, 4

Insertion Sort

- What if first *k* elements of array are already sorted?
  - E.g. 4, 7, 12, 5, 19, 16

- Idea: Can *insert* next element into proper position and get k+1 sorted elements, *insert* next and get k+2 sorted etc.
  - 4, 5, 7, 12, 19, 16
  - 4, 5, 7, 12, 19, 16
  - 4, 5, 7, 12, 16, 19 Done!
  - Overall, N-1 passes needed
  - Similar to card sorting:
    - Start with empty hand
    - Keep inserting...



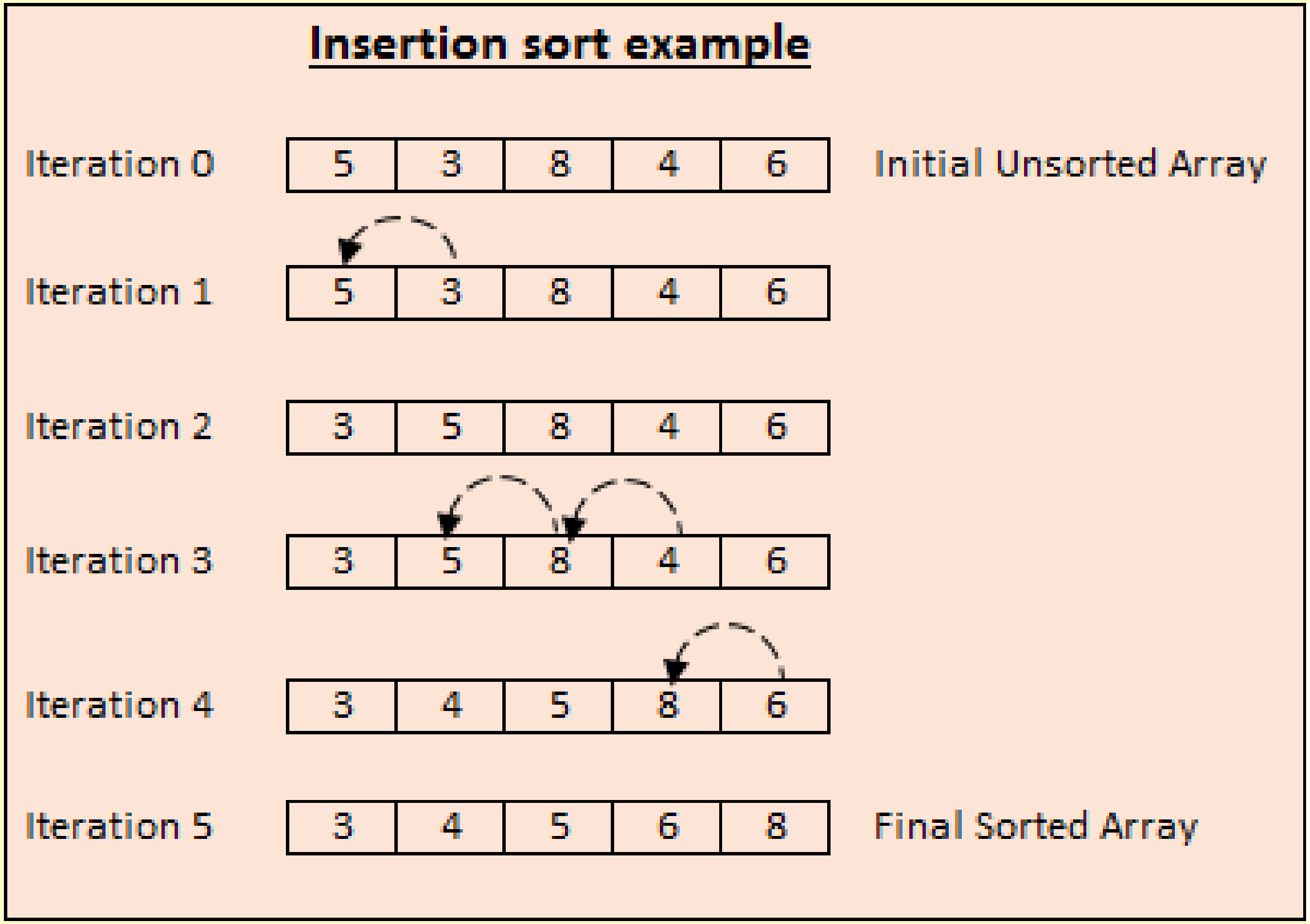
Insertion Sort

```
/* Insertion sort pseudocode for integers
   A is an array containing N integers */
InsertionSort(int A[], int N){
      int j, P, Tmp;
      for(P = 1; P < N; P++ ) {
            Tmp = A[ P ];
            for(j = P; j > 0 && A[ j - 1 ] > Tmp; j-- ){
                        A[ j ] = A[ j - 1 ]; //Shift A[j-1] to right
            } //end-for-inner
            A[ j ] = Tmp; // Found a spot for A[P] (= Tmp)
      } //end-for-outer
} //end-InsertionSort
```

- In place (O(1) space for Tmp) and stable
- Running time:
  - Worst case is reverse order input = $O(N^2)$
  - Best case is input already sorted = $O(N)$.

Insertion Sort (Example)

- Exercise: Sort the following input sequence:



Insertion Sort (Exercise 1)

- Exercise: Sort the following input sequence:
  - 30, 1, 4, 9, 16, 3, 2
  - Start: 30, 1, 4, 9, 16, 3, 2 // Initial array
  - (What are the outputs of Iteration 1, Iteration 2, and Iteration 3?)
  - Iteration1:
  - Iteration2:
  - Iteration3:

Insertion Sort (Exercise 1)

- Exercise: Sort the following input sequence:
  - 30, 1, 4, 9, 16, 3, 2
  - Start: 30, 1, 4, 9, 16, 3, 2 // Initial array
  - (What are the outputs of Iteration 1, Iteration 2, and Iteration 3?)
  - Iteration1: 1, 30, 4, 9, 16, 3, 2
  - Iteration2: 1, 4, 30, 9, 16, 3, 2
  - Iteration3: 1, 4, 9, 30, 16, 3, 2

Insertion Sort (Exercise 2)

- Exercise: Sort the following input sequence:
  - 9, 2, 7, 3, 5, 20, 16, 4
  - Start: 9, 2, 7, 3, 5, 20, 16, 4 // Initial array
  - (What are the outputs of Iteration 1, Iteration 2, and Iteration 3?)
  - Iteration1:
  - Iteration2:
  - Iteration3:

Insertion Sort (Exercise 2)

- Exercise: Sort the following input sequence:
  - 9, 2, 7, 3, 5, 20, 16, 4
  - Start: 9, 2, 7, 3, 5, 20, 16, 4 // Initial array
  - (What are the outputs of Iteration 1, Iteration 2, and Iteration 3?)
  - Iteration1: 2, 9, 7, 3, 5, 20, 16, 4
  - Iteration2: 2, 7, 9, 3, 5, 20, 16, 4
  - Iteration3: 2, 3, 7, 9, 5, 20, 16, 4

Summary of Simple Sorting Algos

- Simple Sorting choices:
  - BubbleSort - $O(N^2)$
  - Selection Sort - $O(N^2)$
  - Insertion Sort - $O(N^2)$

  - Insertion sort gives the best practical performance for small input sizes (~20)