CSC 211: Object Oriented Programming
Basic Sorting Algorithms

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Fall 2019

THINK BIG WE DO℠

Sorting

- Given an input sequence of **n** elements that can be compared to each other according to a **total order** relation
  - we want to rearrange them in non-increasing/non-decreasing order
- Example (sorting in non-decreasing order):
  - **input**: array $A = [k_0, k_1, \ldots, k_{n-1}]$
  - **output**: array B (permutation of A), s.t. $B[0] \leq \ldots \leq B[n-1]$

`Central problem in computer science`

Bubble-Sort

- Basic sorting algorithm
  - yet too slow in practice

- Scan the input sequence from left-to-right
  - compare all adjacent elements and swap them if they are in the wrong order
- Repeat the scan until the list is sorted

`After every pass (iteration), the smaller/larger element bubbles up to the end of the sequence`

(animation)

Algorithm

```
void swap(int& v1, int& v2) {
    int temp = v1;
    v1 = v2;
    v2 = temp;
}

void bubble(int A[], int n_elem) {
    bool sorted = false;
    while (! sorted) {
        sorted = true;
        for (int i = 0 ; i < (n_elem-1) ; i++) {
            if (A[i] > A[i+1]) {
                sorted = false;
                swap(A[i], A[i+1]);
            }
        }
    }
}

int main() {
    int array[] = {15, 12, 13, 24, 5};
    bubble(array, 5);
}
```

Selection-Sort

- Basic sorting algorithm
  - yet too slow in practice

- Keep two parts: **left part** is already sorted and **right part** is to be sorted
  - initially, the sorted part is empty and the unsorted part is the input sequence
- At every iteration, find the smallest (or largest) element in the unsorted part and swap it with the leftmost unsorted element
  - then move the boundary between parts one element to the right

`At every iteration we select the minimum/maximum`

(animation)

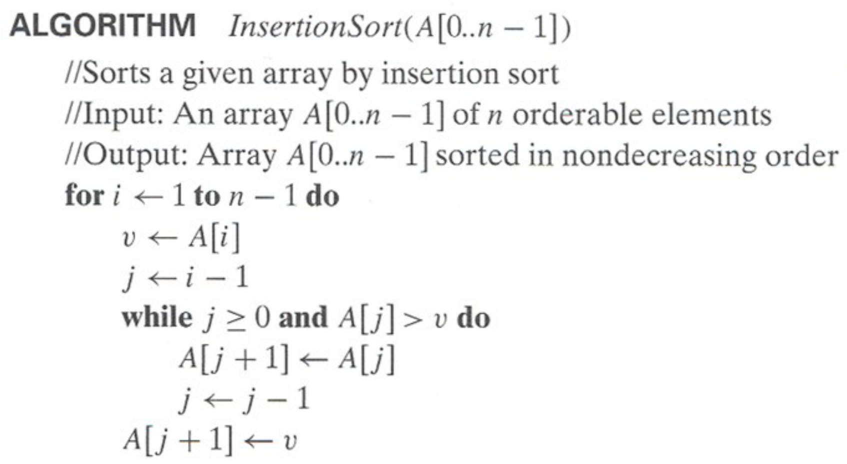
Algorithm

```
void swap(int& v1, int& v2) {
    int temp = v1;
    v1 = v2;
    v2 = temp;
}

int find_min(int A[], int start, int last) {
    int min = start;
    for (int i = start + 1 ; i < last ; i++) {
        if (A[i] < A[min]) {
            min = i;
        }
    }
    return min;
}

void selection(int A[], int n_elem) {
    for (int i = 0, j ; i < (n_elem-1) ; i ++) {
        j = find_min(A, i, n_elem);
        swap(A[i], A[j]);
    }
}

int main() {
    int array[] = {15, 12, 13, 24, 5};
    selection(array, 5);
}
```

Insertion-Sort

- Basic sorting algorithm
  - slightly faster than bubble-sort and selection-sort

- Keep two parts: **left part** is already sorted and **right part** is to be sorted
  - initially, the sorted part contains the first element in the array and the unsorted part is the remaining elements
- At every iteration, the first element of the unsorted part is selected, and the algorithm finds the location it belongs within the sorted part, and inserts it there
  - then move the boundary between parts one element to the right
  - repeat until no elements remain in the unsorted part



```
ALGORITHM  InsertionSort($A[0..n-1]$)
    //Sorts a given array by insertion sort
    //Input: An array $A[0..n-1]$ of $n$ orderable elements
    //Output: Array $A[0..n-1]$ sorted in nondecreasing order
    for $i \leftarrow 1$ to $n-1$ do
        $v \leftarrow A[i]$
        $j \leftarrow i-1$
        while $j \geq 0$ and $A[j] > v$ do
            $A[j+1] \leftarrow A[j]$
            $j \leftarrow j-1$
        $A[j+1] \leftarrow v$
```

(animation)