DESIGN AND ANALYSIS OF ALGORITHMS

CS 4120/5120

PSEUDOCODE CONVENTIONS

AGENDA

- Pseudocode conventions
- Practice
- Breakout session

PSEUDOCODE CONVENTIONS

SYNTAX

- No semicolon ~~;~~
- Indentation to indicate block structure ~~{…}~~

```
INSERTION-SORT(A)
  for j = 2 to A.length
      key = A[j]
      // Insert A[j] into the sorted sequence
             A[1..j-1]
      i = j – 1
      while i > 0 and A[i] > key
          A[i+1] = A[i]
          i = i – 1
      A[i+1] = key
```

PSEUDOCODE CONVENTIONS
KEYWORDS

- Conditional statement
  - **if .. else**
  - **if .. elseif**
- Keywords for loop
  - **for**
  - **while**
  - **repeat-until**,
- Loop counters
  - **to**, **downto**, **by**

```
INSERTION-SORT(A)
    for j = 2 to A.length
        key = A[j]
        // Insert A[j] into the sorted sequence
            A[1..j-1]
        i = j – 1
        while i > 0 and A[i] > key
            A[i+1] = A[i]
            i = i – 1
        A[i+1] = key
```

PSEUDOCODE CONVENTIONS
THE EQUAL “=” SIGN

- The equal sign “=”
  - Following **if** means **testing** two statements
  - **Not** following **if** means an **assignment**
- Multiple assignments allowed
  - $i = j = k$ is equivalent to
  - $j = k$ followed by $i = j$

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence
        A[1..j-1]
    i = j – 1
    while i > 0 and A[i] > key
        A[i+1] = A[i]
        i = i – 1
    A[i+1] = key
```

PSEUDOCODE CONVENTIONS
SCOPE OF VARIABLES

- *Variables are* ***local*** except when there is explicit indication.
- Loop counter *retains its value* after exiting the loop

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence
        A[1..j-1]
    i = j – 1
    while i > 0 and A[i] > key
        A[i+1] = A[i]
        i = i – 1
    A[i+1] = key
```

PSEUDOCODE CONVENTIONS

ARRAYS

- $A[i]$ indicates the i-th element in array A.
- $A[i..j]$ (or sometimes $A[i \ldots j]$) indicates subarray of A consisting elements from $A[i]$ to $A[j]$.
- **Array indices begin with 1.**
  - 1st element $A[1]$
  - n-th element $A[n]$

```
INSERTION-SORT(A)
  for j = 2 to A.length
      key = A[j]
      // Insert A[j] into the sorted sequence
         A[1..j-1]
      i = j – 1
      while i > 0 and A[i] > key
         A[i+1] = A[i]
         i = i – 1
      A[i+1] = key
```

PSEUDOCODE CONVENTIONS
COMPOUND DATA

- Compound data are organized into ***objects***.
  - Composed of ***attributes***
- Variables representing an array or object are treated as a ***pointer***.
- Sometimes, a pointer refers to no object has value **NIL**.

```
INSERTION-SORT(A)
  for j = 2 to A.length
      key = A[j]
      // Insert A[j] into the sorted sequence
         A[1..j-1]
      i = j – 1
      while i > 0 and A[i] > key
         A[i+1] = A[i]
         i = i – 1
      A[i+1] = key
```

PSEUDOCODE CONVENTIONS
RETURN

- The **return** statement can take more than one value back to the calling procedure.

```
INSERTION-SORT(A)
    for j = 2 to A.length
        key = A[j]
        // Insert A[j] into the sorted sequence
            A[1..j-1]
        i = j – 1
        while i > 0 and A[i] > key
            A[i+1] = A[i]
            i = i – 1
        A[i+1] = key
```

PSEUDOCODE CONVENTIONS
SHORT CIRCUITING "AND" AND "OR"

- The Boolean operators "and" and "or" are ***short circuiting***.
  - Statement x and y
    - y will be evaluated only if x evaluates to TRUE
  - Statement x or y
    - y will be evaluated only if x evaluated to FALSE

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence
       A[1..j-1]
    i = j – 1
    while i > 0 and A[i] > key
       A[i+1] = A[i]
       i = i – 1
    A[i+1] = key
```

PSEUDOCODE CONVENTIONS
WRITE PSEUDOCODE

- Problem
  - Input: An array A of n distinct numbers.
  - Output: the largest number, or the *max*, and the smallest number, or the *min* of A.

Solution 1

Step 1: Sort A in increasing order.
Step 2: Output the first and last number in the sorted list as the min and max of the sequence.

Solution 1

$MAX\text{-}MIN\text{-}SORT(A)$

```
Sort array A into increasing order
return  A[n], A[1]
```

PSEUDOCODE CONVENTIONS
WRITE PSEUDOCODE

- Problem
  - Input: An array A of n <u>distinct</u> numbers.
  - Output: the largest number, or the *max*, and the smallest number, or the *min* of A.

Solution 2
Step 1: Scan array A to compute the max.
Step 2: Scan array A to compute the min.

Solution 2

$MAX\text{-}MIN\text{-}FIND(A)$

    return $FIND\text{-}MAX(A), FIND\text{-}MIN(A)$

$FIND\text{-}MAX(A)$

    $\max = A[1]$
    for $i = 2$ **to** n
        if $A[i] > \max$
            $\max = A[i]$
    return $\max$

$FIND\text{-}MIN(A)$

    $min = A[1]$
    for $i = 2$ **to** n
        if $A[i] < min$
            $min = A[i]$
    return min

PSEUDOCODE CONVENTIONS
WRITE PSEUDOCODE

- Problem
  - Input: An array A of n distinct numbers.
  - Output: the largest number, or the *max*, and the smallest number, or the *min* of A.

Solution 3

Step 1: Initialize temporary variables $curr\_min = curr\_max = A[1]$, assuming the starting index is 1.

Step 2: For each array element $A[i]$, where $i \geq 2$, do the following.

- if $A[i] > curr\_max$, $curr\_max = A[i]$
- else if $A[i] < curr\_min$, $curr\_min = A[i]$

Solution 3

```
MAX-MIN-SCAN(A)
    curr_min = curr_max = A[1]
    for i = 2 to n
        if A[i] < curr_min
            curr_min = A[i]
        elseif A[i] > curr_max
            curr_max = A[i]
    return curr_max, curr_min
```

PSEUDOCODE CONVENTIONS
WRITE PSEUDOCODE

- Problem
  - Input: An array A of n distinct numbers.
  - Output: the largest number, or the *max*, and the smallest number, or the *min* of A.

Solution 4

Step 1: Initialize two empty arrays, $SMALL$ and $LARGE$. **INITIAL**

Step 2: Compare pairs of numbers in array A. **PAIR-WISE CHECK**

- For each pair, store the smaller number in $SMALL$, the larger number in $LARGE$.
- If n is odd, compare the last number with the 2nd to the last number.

Step 3: Initialize $curr\_min = SMALL[1]$, $curr\_max = LARGE[1]$.

Step 4: For all numbers in $SMALL$, if $SMALL[i] < curr\_min$, $curr\_min = SMALL[i]$. **FIND**

Step 5: For all numbers in $LARGE$, if $LARGE[i] > curr\_max$, $curr\_max = LARGE[i]$.

- Problem
  - Input: An array A of n <u>distinct</u> numbers.
  - Output: the largest number, or the *max*, and the smallest number, or the *min* of A.

Solution 4

INITIAL

PAIR-WISE CHECK

FIND

```
MAX-MIN-DIVIDE(A)
    let SMALL[1..[n/2]] be an array
    let LARGE[1..[n/2]] be an array

    (LARGE, SMALL) = DIVIDE (A, LARGE, SMALL)

return FIND-MAX(LARGE), FIND-MIN(SMALL)
```

```
DIVIDE(A, LARGE, SMALL)
    n = A.length
    j = 1
    for i = 1 to n by 2
        if n%2 = 0    // n is even
            if A[i] < A[i + 1]
                SMALL[j] = A[i]
                LARGE[j] = A[i + 1]
            else SMALL[j] = A[i + 1]
                LARGE[j] = A[i]
        else        // n is odd
            if i = n
                if A[i] < A[i − 1]
                    SMALL[j] = A[i]
                else LARGE[j] = A[i]
            else
                if A[i] < A[i + 1]
                    SMALL[j] = A[i]
                    LARGE[j] = A[i + 1]
                else SMALL[j] = A[i + 1]
                    LARGE[j] = A[i]
        j + +
    return LARGE, SMALL
```

EXPRESSING ALGORITHMS
ENGLISH + MATH + COMPLEXITY

- Example
  - Input: An array A of n <u>distinct</u> numbers.
  - Output: the largest number, or the *max*, and the smallest number, or the *min* of A.
  - Objective: Design an algorithm that uses as few comparisons as possible.

- Take notes of the four solutions, participate in your breakout room to determine the number of comparisons in terms of n. (10 ~ 15 minutes)
  - Hint: you may determine the number of comparisons in the worst-case scenario.
  - Hint: you may consider using $\lceil \quad \rceil$ or $\lfloor \quad \rfloor$ operator.

NEXT UP
ANALYZING ALGORITHMS