Sorting Algorithms

- The following pages present several sorting algorithms:
- Two implementations of insertion sort: one that uses 2 stacks and the other that is in-place (this means that the algorithm does not use any auxiliary data structures).
- An in-place implementation of selection sort.
- An implementation of quicksort using 3 auxiliary arrays

Algorithm insertionSort (A,n) // Uses two stacks

In: Array A storing n elements

Out: Sorted array

```
sorted = empty stack
temp = empty stack
for i = 0 to n-1 do {
    while (sorted is not empty) and (sorted.peek() < A[i]) do
        temp.push (sorted.pop())
    sorted.push (A[i])
    while temp is not empty do
        sorted.push (temp.pop())
}
for i = 0 to n-1 do
    A[i] = sorted.pop()
```

```
// In-Place insertion sort: does not use auxiliary data
// structures
Algorithm insertionSort (A,n)
In: Array A storing n values
Out: {Sort A in increasing order}
for i = 1 to n-1 do {
    temp = A[i]
    j = i – 1
    // Insert A[i] into the sorted subarray A[0] ... A[i-1]
    while (j >= 0) and (A[j] > temp) do {
        A[j+1] = A[j]
        j = j – 1
    }
    A[j+1] = temp
}
```

```
// In-Place selection sort
Algorithm selectionSort (A,n)
In: Array A storing n values
Out: {Sort A in increasing order}
for i = 0 to n-2 do {
    // Find the smallest value in A[i] … A[n-1]
    smallest = i
    for j = i + 1 to n - 1do {
        if A[j] < A[smallest] then
            smallest = j
    }
    temp = A[smallest]
    A[smallest] = A[i]
    A[i] = temp
}
```

```
Algorithm quicksort(A,n)
In: Array A storing n values
Out: {Sort A in increasing order}
If n > 1 then {
    smaller, equal, larger = new arrays of size n
    n_s = n_e = n_l = 0
    pivot = A[0]
    for i = 0 to n-1 do  // Partition the values
        if A[i] = pivot then equal[n_e++] = A[i]
        else if A[i] > pivot then larger[n_l++] = A[i]
        else smaller[n_s++] = A[i]
    quicksort(smaller,n_s)  // Sort smaller and larger
    quicksort(larger,n_l)
    i = 0                   // Copy data back to original array
    for j = 0 to n_s do A[i++] = smaller[j]
    for j = 0 to n_e do A[i++] = equal[j]
    for j = 0 to n_l do A[i++] = larger[j]
}
```