Bubble, Insertion, Selection & Tree Sorts

Kuan-Yu Chen (陳冠宇)

2018/11/07 @ TR-212, NTUST

Sorting

- Sorting means arranging the elements of an array so that they are placed in some relevant order which may be either **ascending** or **descending**

- A sorting algorithm is defined as an algorithm that puts the elements of a list in a certain order, which can be either **numerical** order, **lexicographical** order, or **any user-defined** order
  - Bubble, Insertion, Selection, Tree
  - Merge, Quick, Radix, Heap, Shell

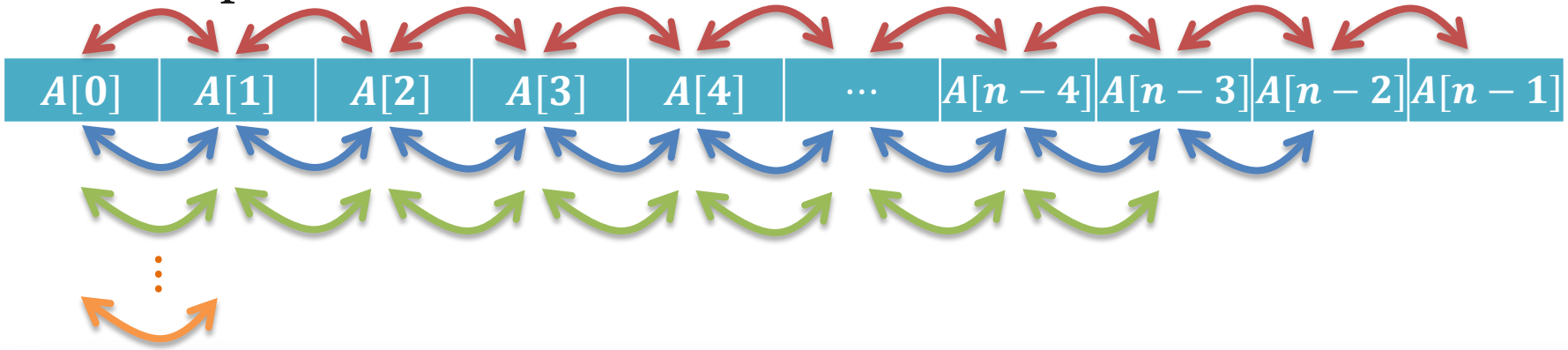
Bubble Sort.

- Bubble sort is a very simple method that sorts the array elements by repeatedly moving the largest element to the highest index position of the array segment
  - Consecutive adjacent pairs of elements in the array are compared with each other
  - If the element at the lower index is greater than the element at the higher index, the two elements are interchanged

- This procedure of sorting is called bubble sorting because elements “bubble” to the top of the list

Bubble Sort..

- The procedure!



The basic methodology of the working of bubble sort is given as follows:

(a) In Pass 1, `A[0]` and `A[1]` are compared, then `A[1]` is compared with `A[2]`, `A[2]` is compared with `A[3]`, and so on. Finally, `A[N-2]` is compared with `A[N-1]`. Pass 1 involves `n-1` comparisons and places the biggest element at the highest index of the array.

(b) In Pass 2, `A[0]` and `A[1]` are compared, then `A[1]` is compared with `A[2]`, `A[2]` is compared with `A[3]`, and so on. Finally, `A[N-3]` is compared with `A[N-2]`. Pass 2 involves `n-2` comparisons and places the second biggest element at the second highest index of the array.

(c) In Pass 3, `A[0]` and `A[1]` are compared, then `A[1]` is compared with `A[2]`, `A[2]` is compared with `A[3]`, and so on. Finally, `A[N-4]` is compared with `A[N-3]`. Pass 3 involves `n-3` comparisons and places the third biggest element at the third highest index of the array.

(d) In Pass `n-1`, `A[0]` and `A[1]` are compared so that `A[0]<A[1]`. After this step, all the elements of the array are arranged in ascending order.

Example.

- Please sort a given data array by using bubble sort

```
A[] = {30, 52, 29, 87, 63, 27, 19, 54}
```

  - Pass 1:

```
(a) Compare 30 and 52. Since 30 < 52, no swapping is done.
(b) Compare 52 and 29. Since 52 > 29, swapping is done.
    30, 29, 52, 87, 63, 27, 19, 54
(c) Compare 52 and 87. Since 52 < 87, no swapping is done.
(d) Compare 87 and 63. Since 87 > 63, swapping is done.
    30, 29, 52, 63, 87, 27, 19, 54
(e) Compare 87 and 27. Since 87 > 27, swapping is done.
    30, 29, 52, 63, 27, 87, 19, 54
(f) Compare 87 and 19. Since 87 > 19, swapping is done.
    30, 29, 52, 63, 27, 19, 87, 54
(g) Compare 87 and 54. Since 87 > 54, swapping is done.
    30, 29, 52, 63, 27, 19, 54, 87
```

Example..

- Please sort a given data array by using bubble sort

```
A[] = {30, 52, 29, 87, 63, 27, 19, 54}
```

  - Pass 1:

```
30, 29, 52, 63, 27, 19, 54, 87
```

  - Pass 2:

```
(a) Compare 30 and 29. Since 30 > 29, swapping is done.
    29, 30, 52, 63, 27, 19, 54, 87
(b) Compare 30 and 52. Since 30 < 52, no swapping is done.
(c) Compare 52 and 63. Since 52 < 63, no swapping is done.
(d) Compare 63 and 27. Since 63 > 27, swapping is done.
    29, 30, 52, 27, 63, 19, 54, 87
(e) Compare 63 and 19. Since 63 > 19, swapping is done.
    29, 30, 52, 27, 19, 63, 54, 87
(f) Compare 63 and 54. Since 63 > 54, swapping is done.
    29, 30, 52, 27, 19, 54, 63, 87
```

Example...

- Please sort a given data array by using bubble sort

```
A[] = {30, 52, 29, 87, 63, 27, 19, 54}
```

  - Pass 2:

```
29, 30, 52, 27, 19, 54, 63, 87
```

  - Pass 3:

```
(a) Compare 29 and 30. Since 29 < 30, no swapping is done.
(b) Compare 30 and 52. Since 30 < 52, no swapping is done.
(c) Compare 52 and 27. Since 52 > 27, swapping is done.
    29, 30, 27, 52, 19, 54, 63, 87
(d) Compare 52 and 19. Since 52 > 19, swapping is done.
    29, 30, 27, 19, 52, 54, 63, 87
(e) Compare 52 and 54. Since 52 < 54, no swapping is done.
```

Example….

- Please sort a given data array by using bubble sort

```
A[] = {30, 52, 29, 87, 63, 27, 19, 54}
```

  - Pass 3:

```
29, 30, 27, 19, 52, 54, 63, 87
```

  - Pass 4:

```
(a) Compare 29 and 30. Since 29 < 30, no swapping is done.
(b) Compare 30 and 27. Since 30 > 27, swapping is done.
    29, 27, 30, 19, 52, 54, 63, 87
(c) Compare 30 and 19. Since 30 > 19, swapping is done.
    29, 27, 19, 30, 52, 54, 63, 87
(d) Compare 30 and 52. Since 30 < 52, no swapping is done.
```

Example…..

- Please sort a given data array by using bubble sort

```
A[] = {30, 52, 29, 87, 63, 27, 19, 54}
```

  - Pass 4:

```
29, 27, 19, 30, 52, 54, 63, 87
```

  - Pass 5:

```
(a) Compare 29 and 27. Since 29 > 27, swapping is done.
    27, 29, 19, 30, 52, 54, 63, 87
(b) Compare 29 and 19. Since 29 > 19, swapping is done.
    27, 19, 29, 30, 52, 54, 63, 87
(c) Compare 29 and 30. Since 29 < 30, no swapping is done.
```

Example……

- Please sort a given data array by using bubble sort

```
A[] = {30, 52, 29, 87, 63, 27, 19, 54}
```

  - Pass 5:

```
27, 19, 29, 30, 52, 54, 63, 87
```

  - Pass 6:

```
(a) Compare 27 and 19. Since 27 > 19, swapping is done.
    19, 27, 29, 30, 52, 54, 63, 87
(b) Compare 27 and 29. Since 27 < 29, no swapping is done.
```

Example…….

- Please sort a given data array by using bubble sort

```
A[] = {30, 52, 29, 87, 63, 27, 19, 54}
```

  - Pass 6:

```
19, 27, 29, 30, 52, 54, 63, 87
```

  - Pass 7:

```
(a) Compare 19 and 27. Since 19 < 27, no swapping is done.
```

Bubble Sort…

```
BUBBLE_SORT(A, N)

Step 1: Repeat Step 2 For I = 0 to N
Step 2:     Repeat For J = 0 to N - I -1
Step 3:                 IF A[J] > A[J + 1]
                        SWAP A[J] and A[J+1]
            [END OF INNER LOOP]
        [END OF OUTER LOOP]
Step 4: EXIT
```

Insertion Sort.

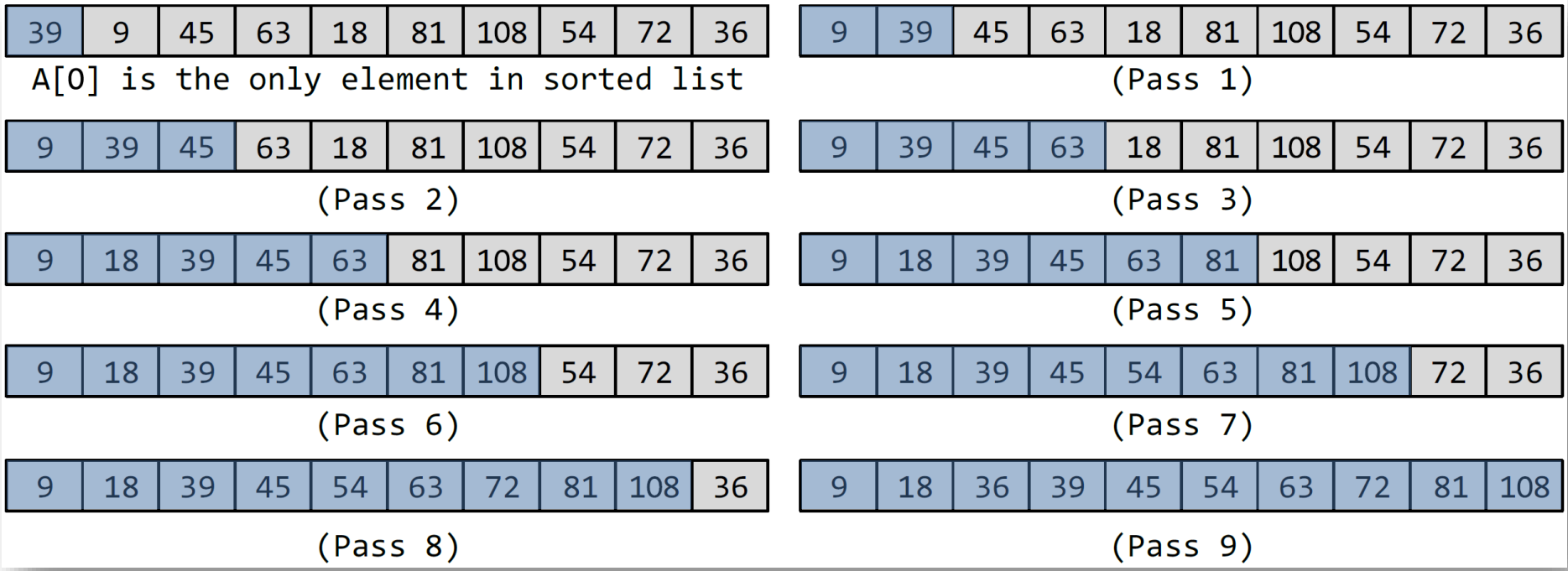
- Insertion sort is a very simple sorting algorithm in which the sorted array (or list) is built one element at a time

- The procedure!
  - The array of values to be sorted is divided into two sets
    - One stores sorted values
    - Another contains unsorted values
  - The sorting algorithm will proceed until there are no elements in the unsorted set

Example

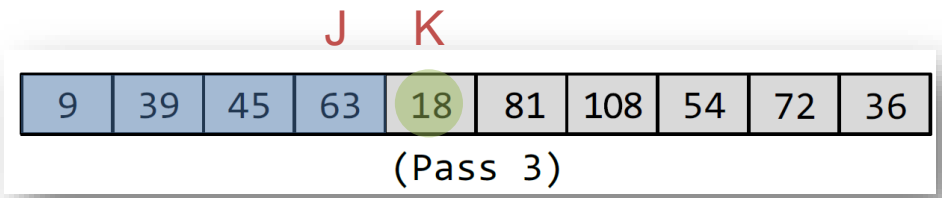
- Please sort a given data array by using insertion sort

39	9	45	63	18	81	108	54	72	36



Insertion Sort..

```
INSERTION-SORT (ARR, N)

Step 1: Repeat Steps 2 to 5 for K = 1 to N – 1
Step 2:      SET TEMP = ARR[K]
Step 3:      SET J = K - 1
Step 4:      Repeat while TEMP <= ARR[J]
                   SET ARR[J + 1] = ARR[J]
                   SET J = J - 1
             [END OF INNER LOOP]
Step 5:      SET ARR[J + 1] = TEMP
         [END OF LOOP]
Step 6: EXIT
```



Selection Sort.

- Selection sort is also a simple algorithm for sorting

- The procedure!
  - Consider an array with N elements
    - First find the smallest value in the array and place it in the first position
    - Then, find the second smallest value in the array and place it in the second position
    - Repeat this procedure until the entire array is sorted

Example

- Please sort a given data array by using selection sort

39	9	81	45	90	27	72	18

PASS	ARR[0]	ARR[1]	ARR[2]	ARR[3]	ARR[4]	ARR[5]	ARR[6]	ARR[7]
1	9	39	81	45	90	27	72	18
2	9	18	81	45	90	27	72	39
3	9	18	27	45	90	81	72	39
4	9	18	27	39	90	81	72	45
5	9	18	27	39	45	81	72	90
6	9	18	27	39	45	72	81	90
7	9	18	27	39	45	72	81	90

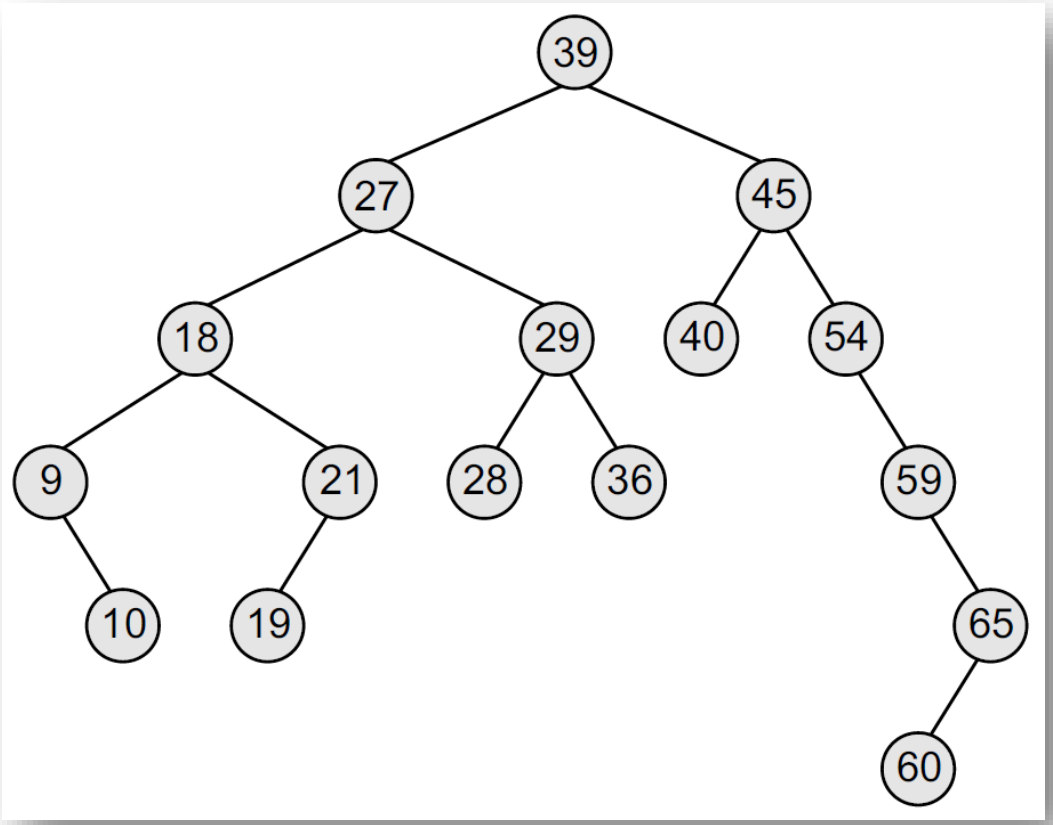
Selection Sort..

```
SMALLEST (ARR, K, N, POS)

Step 1: [INITIALIZE] SET SMALL = ARR[K]
Step 2: [INITIALIZE] SET POS = K
Step 3: Repeat for J = K+1 to N-1
            IF SMALL > ARR[J]
                  SET SMALL = ARR[J]
                  SET POS = J
            [END OF IF]
        [END OF LOOP]
Step 4: RETURN POS
```

```
SELECTION SORT(ARR, N)

Step 1: Repeat Steps 2 and 3 for K = 1
        to N-1
Step 2:     CALL SMALLEST(ARR, K, N, POS)
Step 3:     SWAP A[K] with ARR[POS]
      [END OF LOOP]
Step 4: EXIT
```

Tree Sort

- A tree sort is a sorting algorithm that sorts numbers by making use of the properties of binary search tree
  - Build a binary search tree
  - Do an in-order traversal



Questions?

kychen@mail.ntust.edu.tw