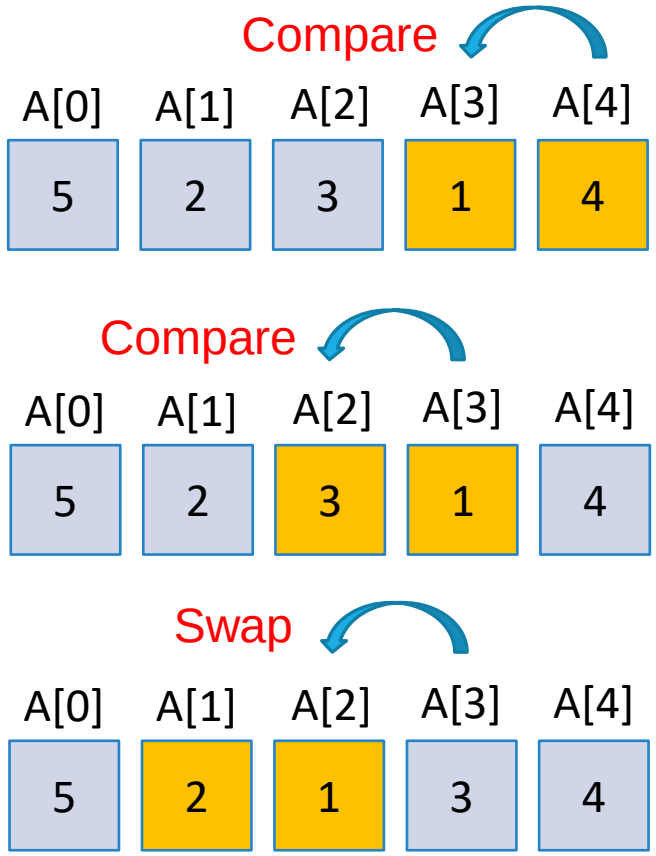
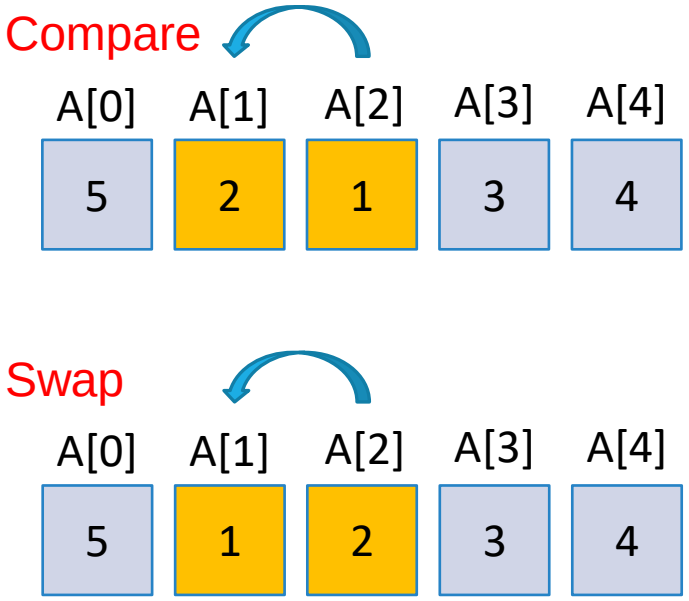
CSCI2100C Data Structures
Tutorial 06 – Bubble Sort, Insertion Sort

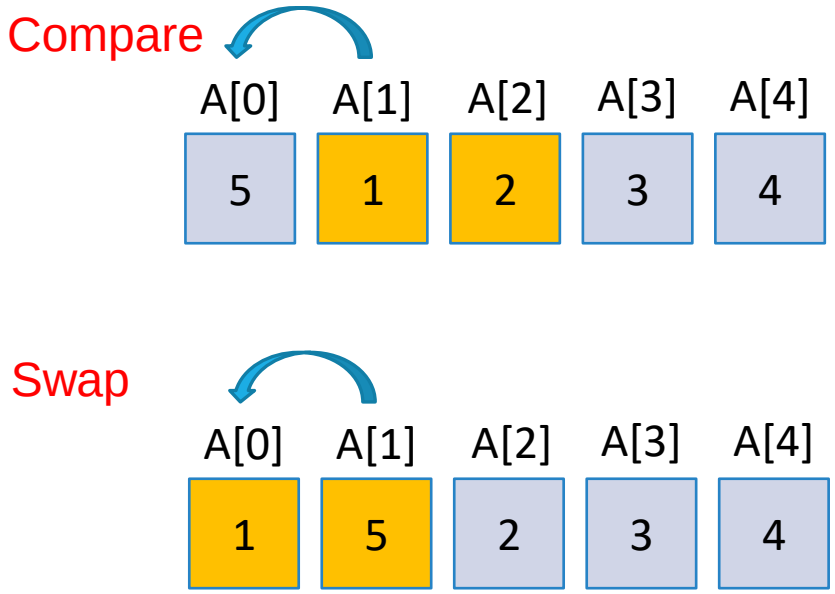
LI Muzhi
mzli@cse.cuhk.edu.hk

Bubble Sort (泡沫排序 / 冒泡排序) – Pass #1

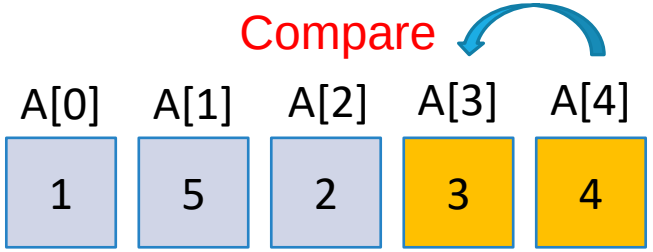
- Bubble sort iteratively swap adjacent items in wrong order.

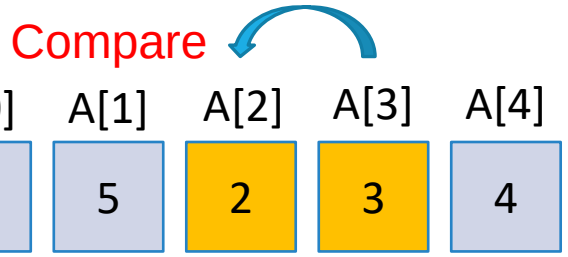


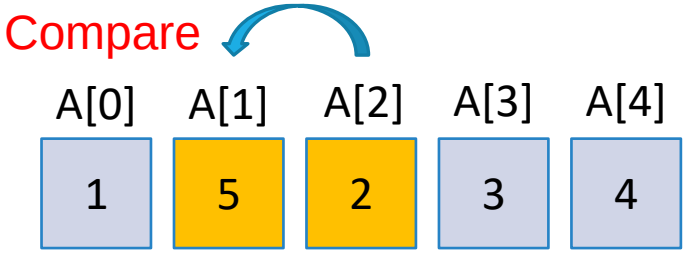


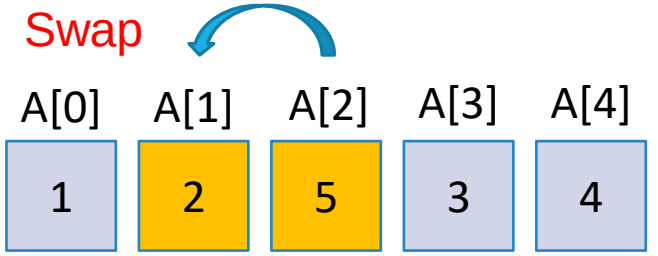


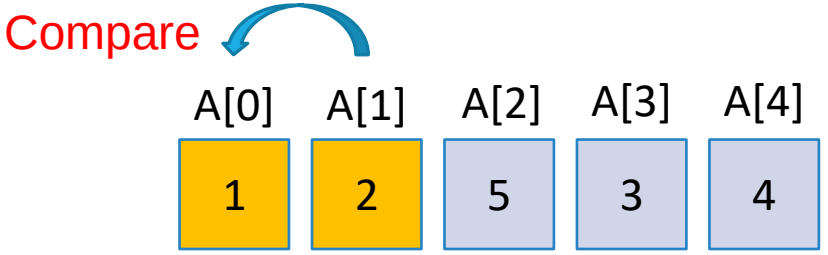
Bubble Sort – Pass #2



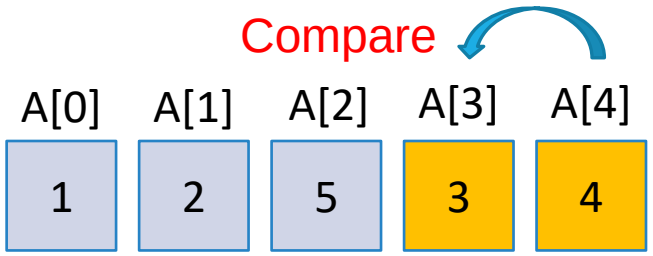


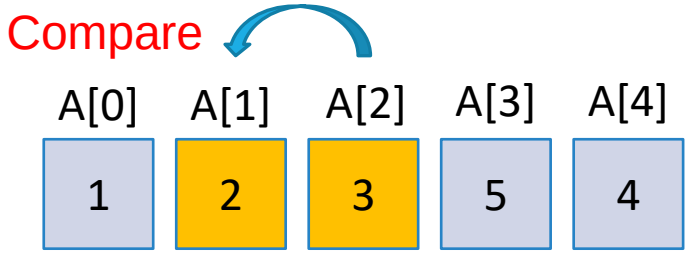


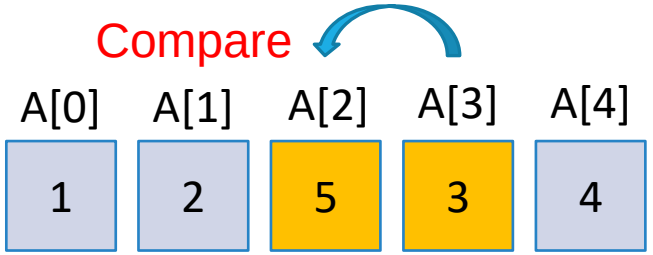


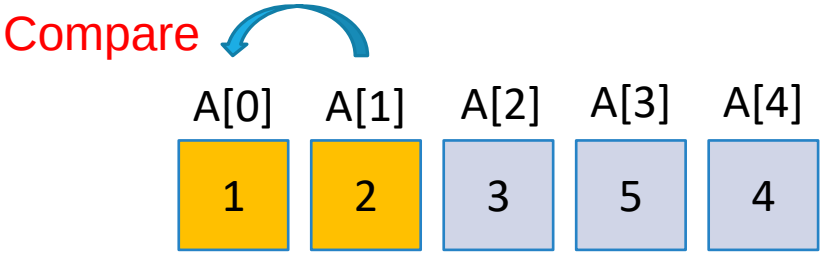


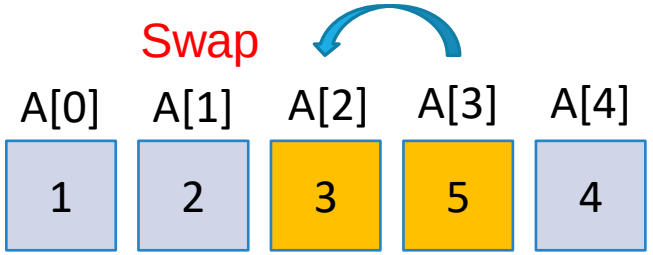
Bubble Sort – Pass #3



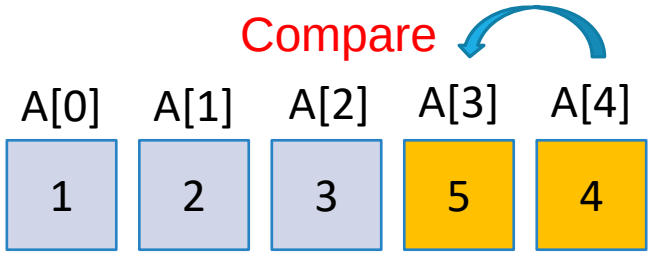


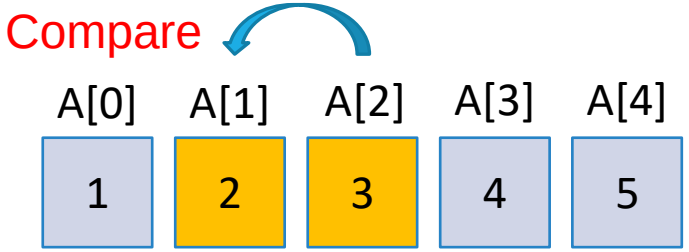


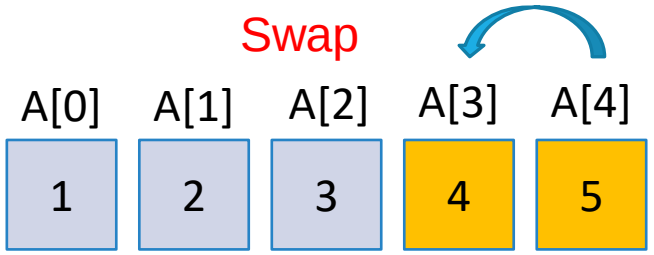


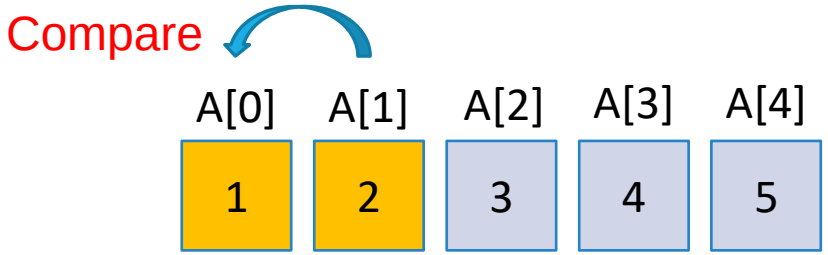


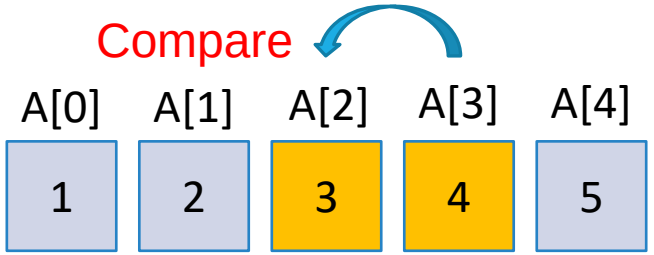
Bubble Sort – Pass #4





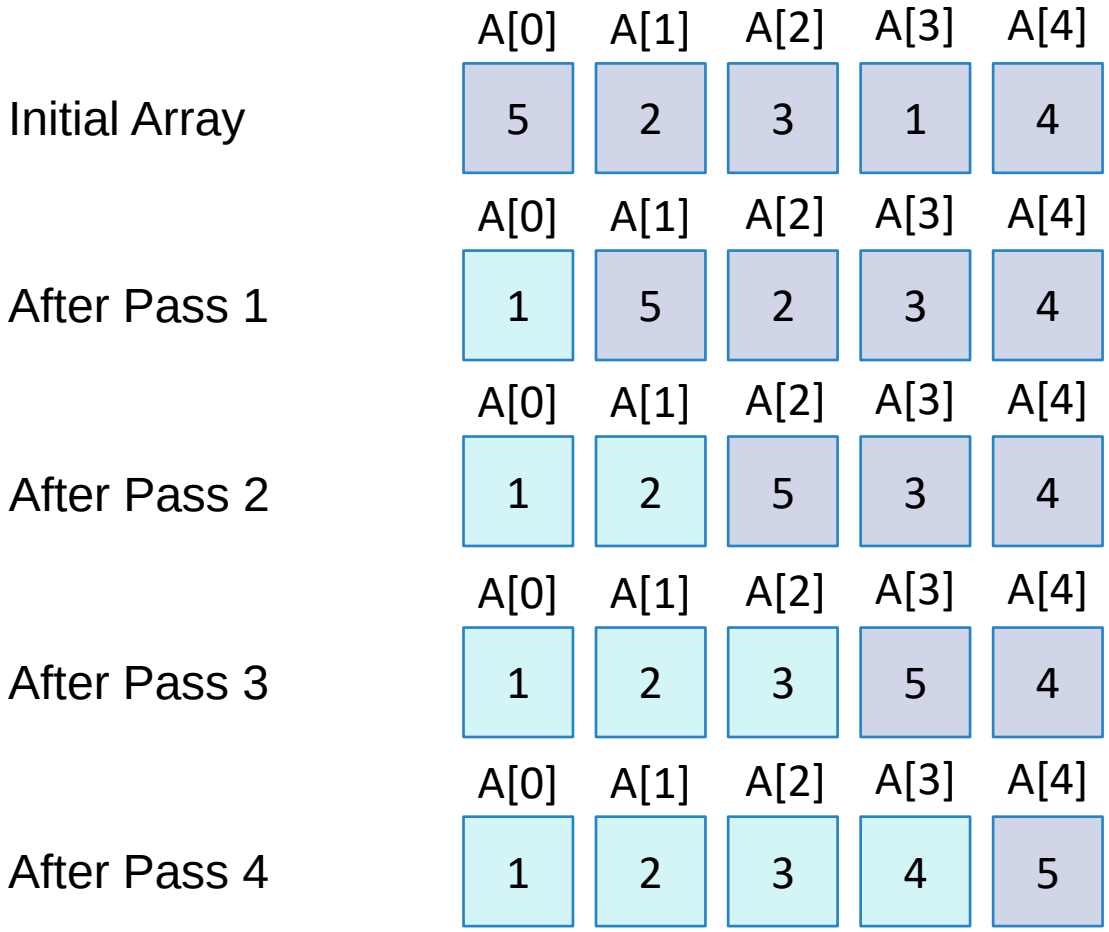






- It seems that the array is already sorted.

Bubble Sort – Observation



- After pass i, the first ___i___ elements of the array are sorted.
- Therefore, use bubble sort to sort an array of 5 elements, (at most) ___4___ iterations are required.
- Use bubble sort to sort an array of n elements, (at most) ___$n-1$___ passes are required.

Bubble Sort Ver. 1.0 – Implementation

```
void bubble_sort(int arr[], int len) {



}
```

- Use bubble sort to sort an array of n elements, (at most) $n-1$ passes are required.
- After iteration i, the first i elements of the array are sorted.

Bubble Sort – Analysis

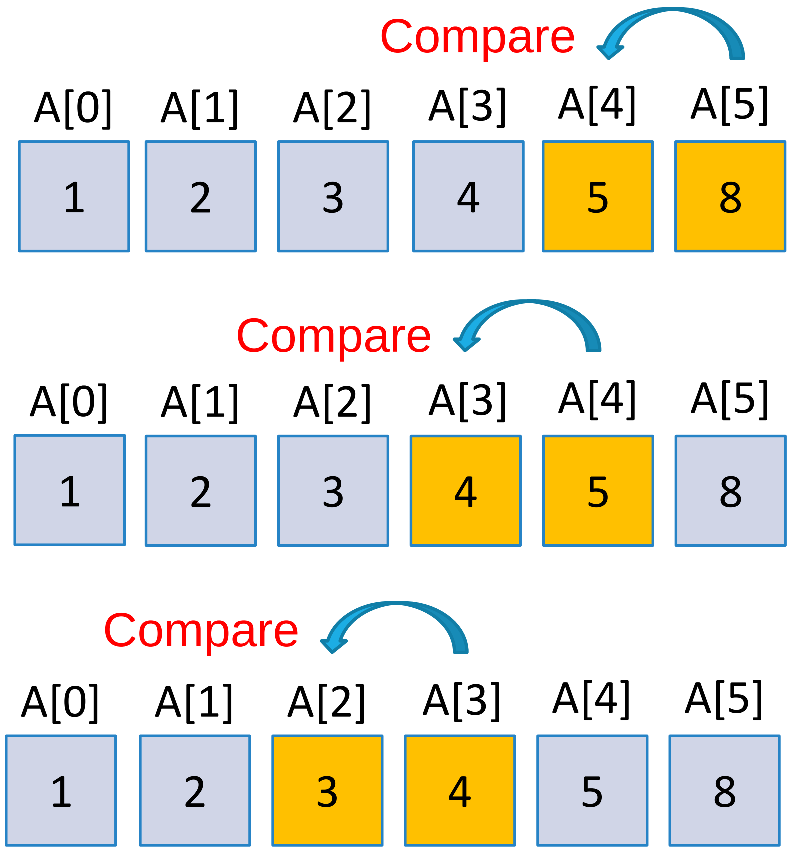
```
void bubble_sort(int arr[], int len) {
    for (int i = 0; i < len - 1; i++) {
        for (int j = len - 1; j > i; j--) {
            if (arr[j] < arr[j - 1]) {
                swap(&arr[j], &arr[j-1]);
            }
        }
    }
}
```

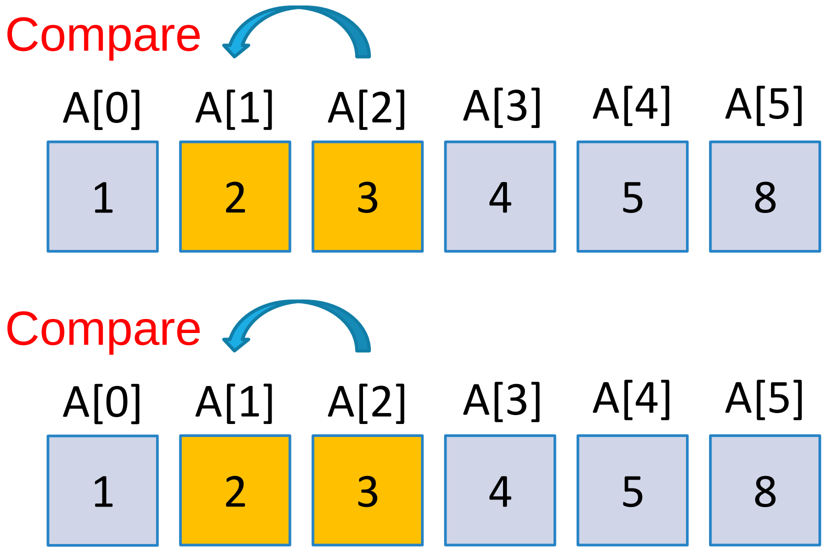
```
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;

}
```

- At pass #1 ($i = 0$), we perform $n - 1$ comparisons.
- At pass #2 ($i = 1$), we perform $n - 2$ comparisons.
- At pass #m ($i = m - 1$), we perform $n - m$ or $n - i - 1$ item comparisons
- Therefore, the total number of running time at the worst case is roughly proportional to $(n - 1) + (n - 2) + \cdots + 1 = \frac{n^2 - n}{2}$

Bubble Sort – Early Stop





- If there's no swap required in one pass, we can terminate the sorting algorithm in advance

Bubble Sort – Ver. 1.1

- If there's no swap required in one pass, one can terminate the sorting algorithm in advance

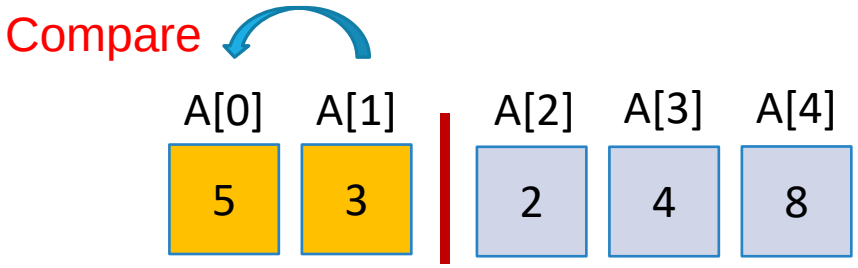
```
void bubble_sort(int arr[], int len) {
    int swap_flag = 1;
    for (int i = 0; i < len - 1 && swap_flag; i++) {
        swap_flag = 0;
        for (int j = len - 1; j > i; j--) {
            if (arr[j] < arr[j - 1]) {
                swap(&arr[j], y: &arr[j-1]);
                swap_flag = 1;
            }
        }
    }
}
```

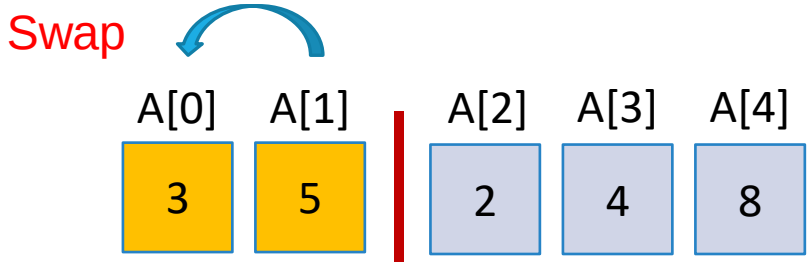
```
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;

}
```

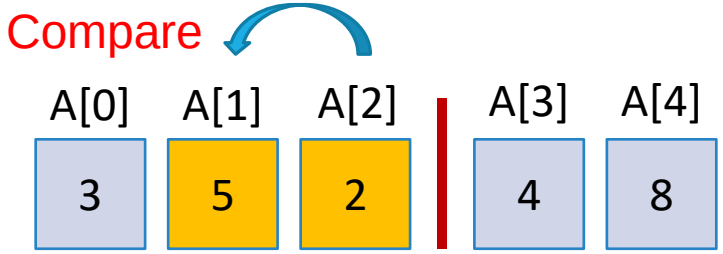
Insertion Sort (插入排序) – Pass 1 & 2

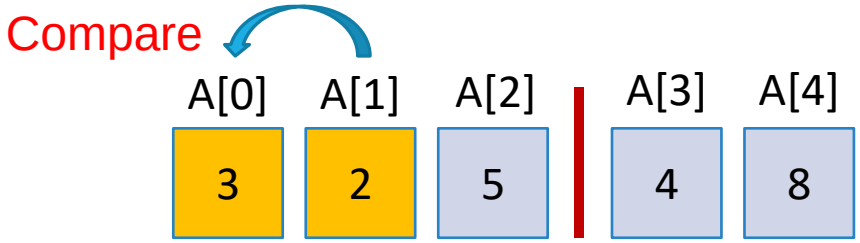
- Pass #1

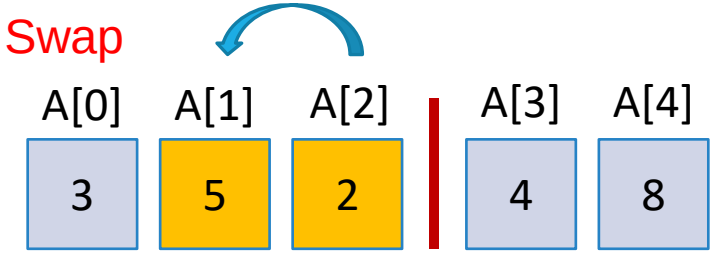


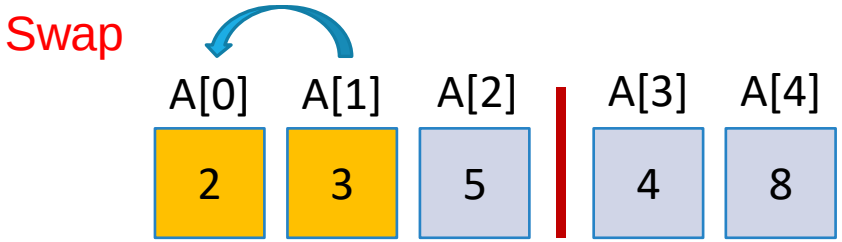


- Pass #2



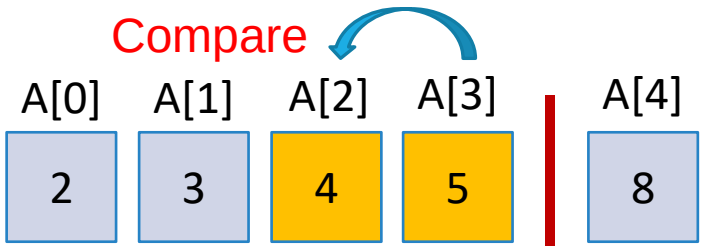


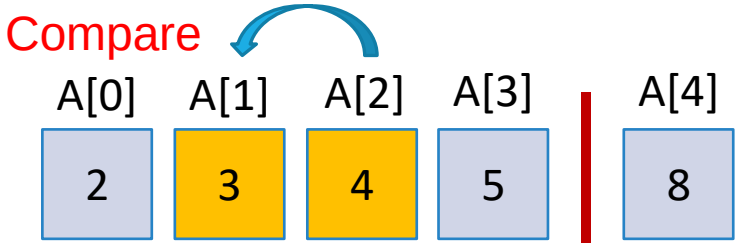


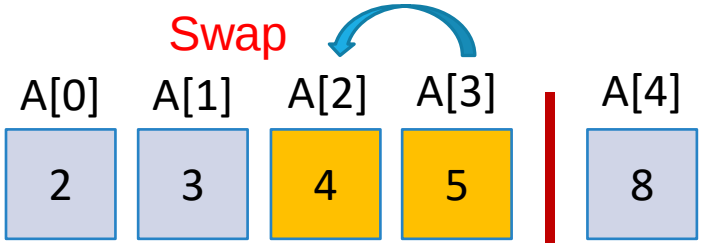


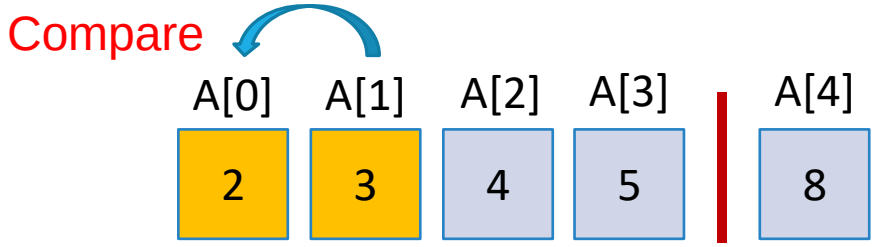
Insertion Sort (插入排序) – Pass 3

- Pass #3



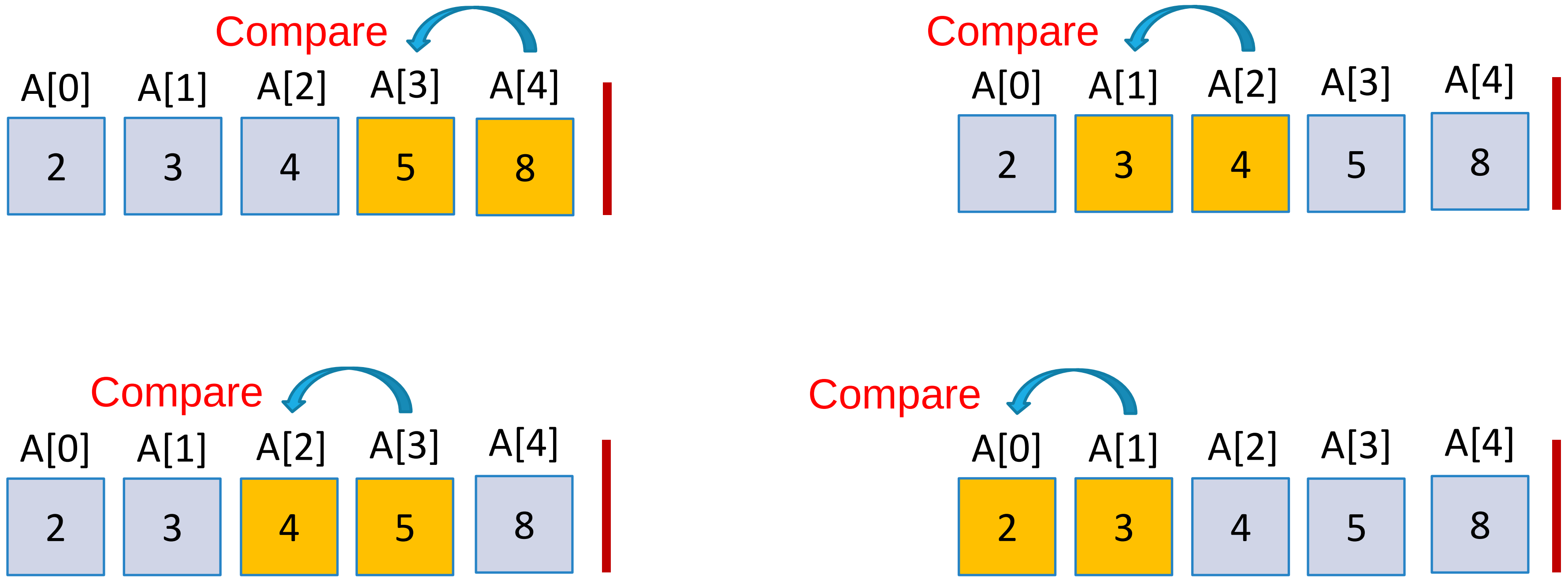




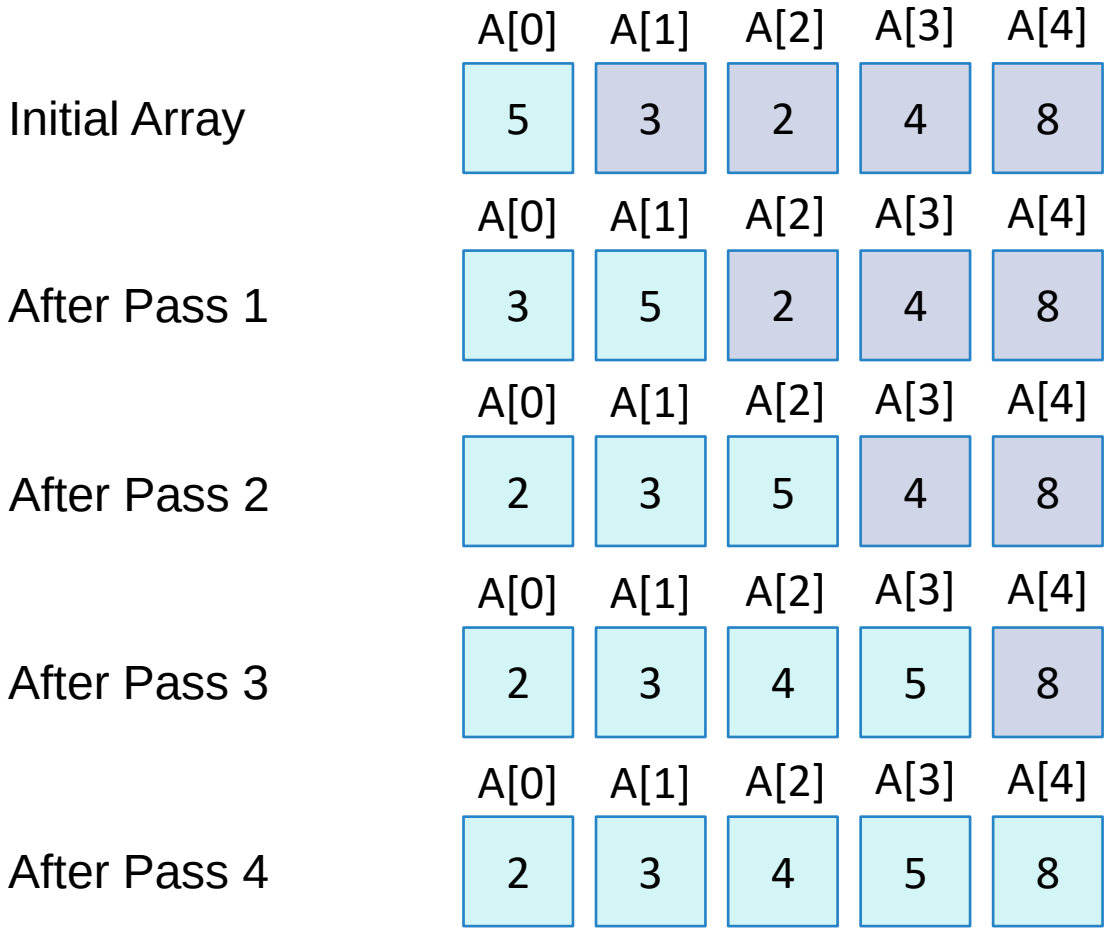


Insertion Sort (插入排序) – Pass 4

- Pass #4



Insertion Sort – Observation



- After iteration i, the first i elements of the array are sorted.
- Therefore, use insertion sort to sort an array of 5 elements, (at most) 4 iterations are required.
- Use bubble sort to sort an array of n elements, (at most) $n-1$ passes are required.

Insertion Sort Ver. 1.0 – Implementation

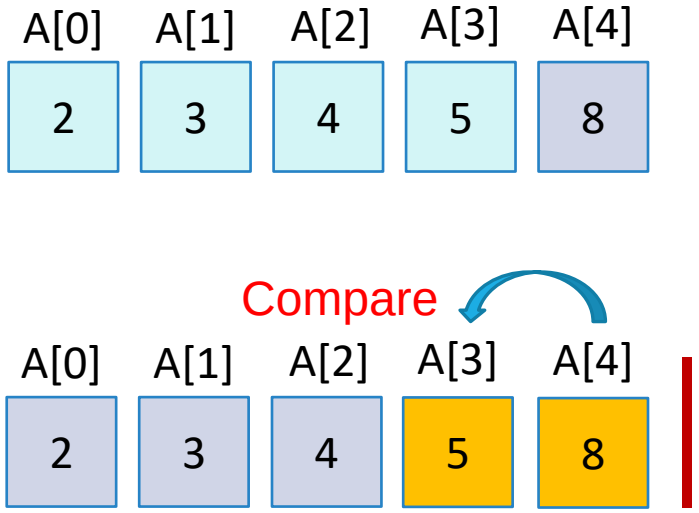
```
void insertion_sort(int arr[], int len) {
   for (int i = 1; i < len; i++) {
      for (int j = i; j > 0; j--) {
         if (arr[j] < arr[j-1]){
            swap(&arr[j], &arr[j-1]);
         }
      }
   }
}
```

```
void swap(int *x, int *y) {
   int temp = *x;
   *x = *y;
   *y = temp;
}
```

Insertion Sort – Early Stop (for each pass)

- Pass #4

After Pass 3



- At pass 4, the first 4 elements are sorted, then the 4^{th} element is the largest element among first 4 elements.
- As the 5^{th} element $8 > 5$, we know that the 5^{th} element is larger than the largest element of first 4 elements. Then we can stop this pass and enter the next one.
- More general, if one single comparison shows that no swapping is required, one can stop the current pass and enter the next one. Why?

Insertion Sort Ver. 1.1 – Implementation

```
void insertion_sort(int arr[], int len) {
   for (int i = 1; i < len; i++) {
      for (int j = i; j > 0 && arr[j] < arr[j-1]; j--) {
         swap(&arr[j], &arr[j-1]);
      }
   }
}
```

- At Pass i, the first i elements are sorted, then the i^{th} element is the largest element among first i elements.
- if one single comparison shows that no swapping is required, one can stop the current pass and enter the next one.

Insertion Sort – Analysis

```
void insertion_sort(int arr[], int len) {
  for (int i = 1; i < len; i++) {
    for (int j = i; j > 0 && arr[j] < arr[j-1]; j--) {
      swap(&arr[j], &arr[j-1]);
    }
  }
}
```

- At pass #1 ($i = 1$), we perform at most ___1___ comparisons.
- At pass #2 ($i = 2$), we perform at most ___2___ comparisons.
- At pass #m ($i = m$), we perform ___m or i___ item comparisons
- Therefore, the total number of running time at the worst case to sort a n element array is roughly proportional to ___$1 + 2 + \cdots + (n-1) = \frac{n^2-n}{2}$___

Thanks