Data Structures and Algorithms
(ES221)

Insertion and Selection Sort

Dr. Zubair Ahmad

- **Attendance?**

  - Active Attendance
  - **Dead Bodies.**
  - **Active Minds**
  - Mobiles in hands -> Mark as absent
  - 80% mandatory

Insertion Sort

- On the *i*th pass we "insert" the *i*th element *A*[i] into its rightful place among *A*[1],*A*[2],…*A*[i-1] which were placed in sorted order.

- After this insertion *A*[1],*A*[2],…*A*[i] are in sorted order.

-

3

Insertion Sort

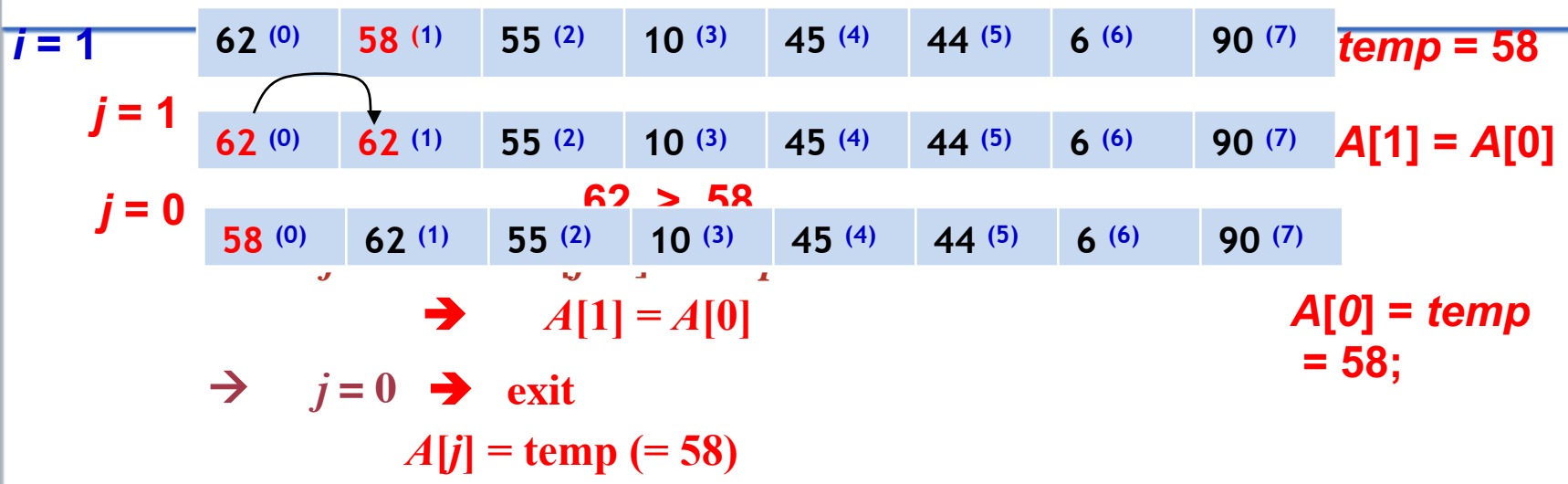
```
for (int i =1, i  <  n, i++){
        temp = A[i];
        for (int j = i, j>0  &&  A[j-1] > temp, j--)
              A[j] = A[j-1]
        A[j] = temp;
        }// end outer for
```

Complexity ?

$O(N^2)$

4

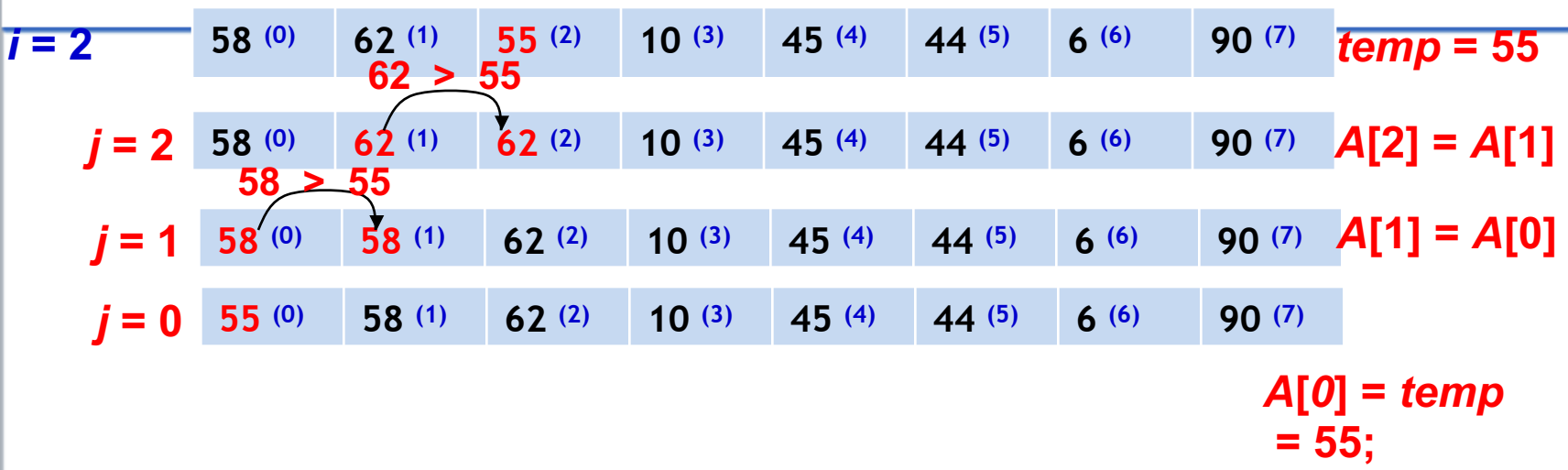
Insertion Sort Example (First Pass)




```
for (int i =1, i  <  n, i++){
    temp = A[i];
        for (int j = i, j > 0  &&  A[j-1] > temp, j--)
            A[j] = A[j-1]
    A[j] = temp;
    }// end outer for
```


5

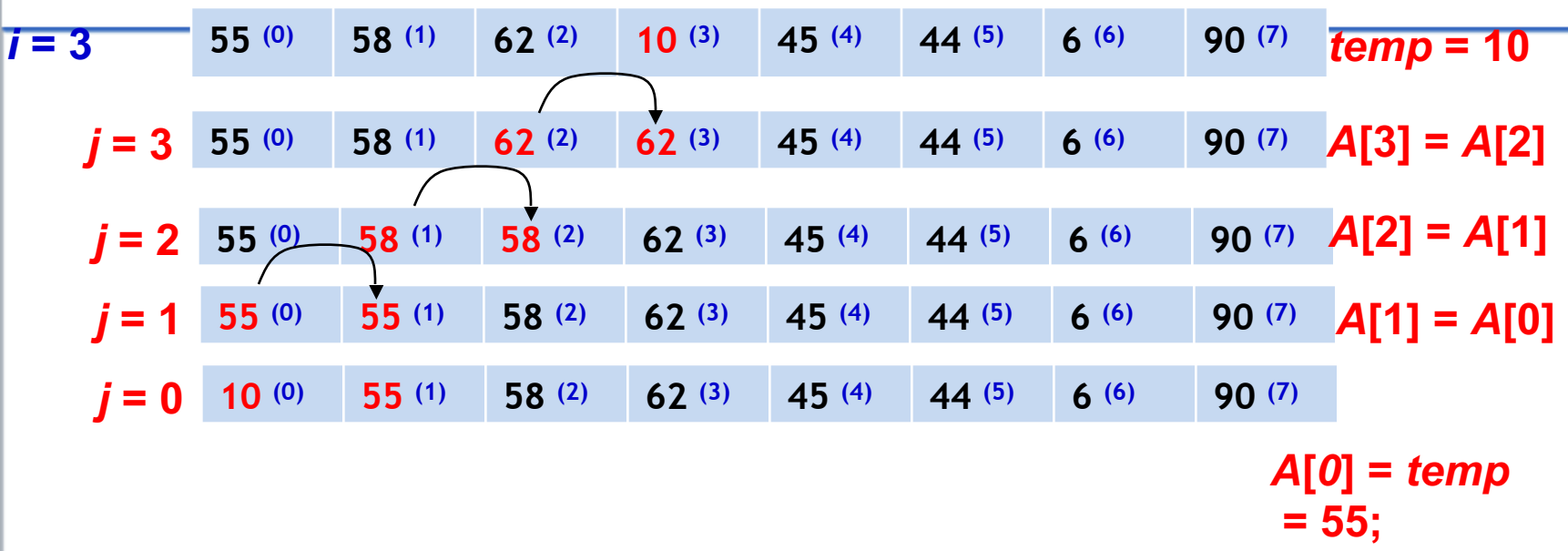
Insertion Sort Example (Second Pass)




```
for (int i =1, i < n, i++){
    temp = A[i];
        for (int j = i, j > 0 && A[j-1] > temp, j--)
            A[j] = A[j-1]
    A[j] = temp;
}// end outer for
```

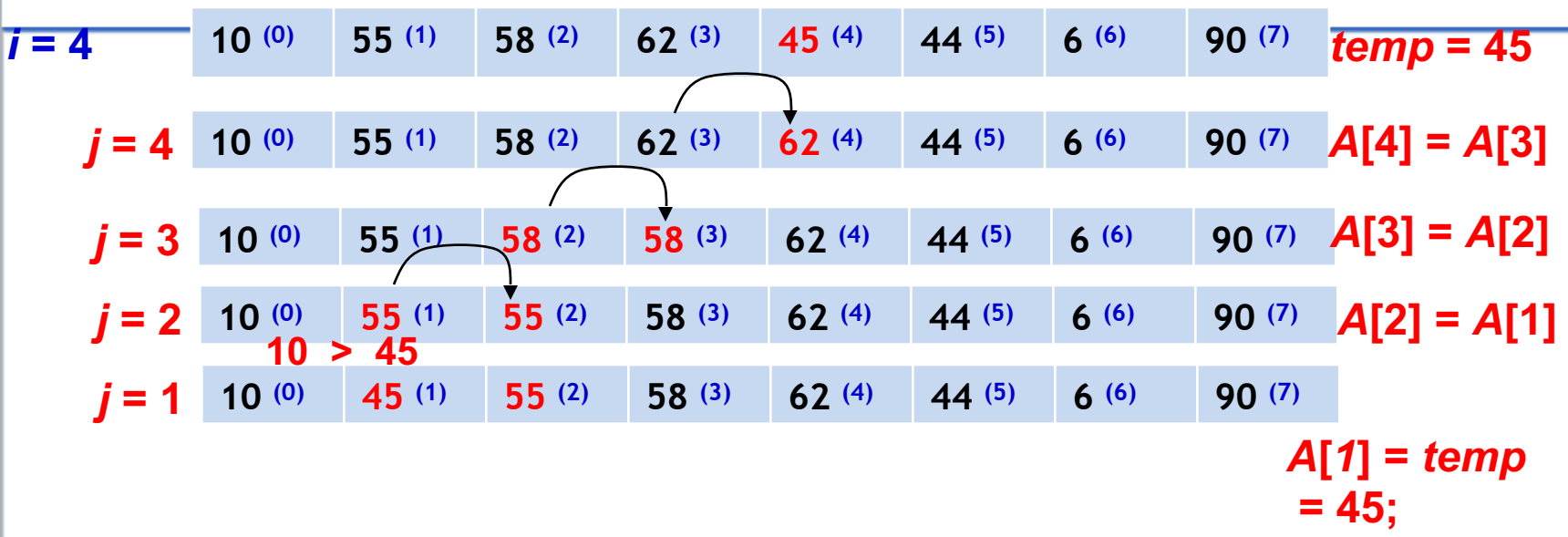

6

Insertion Sort Example (Third Pass)




```
for (int i =1, i < n, i++){
    temp = A[i];
        for (int j = i, j > 0 && A[j-1] > temp, j--)
            A[j] = A[j-1]
    A[j] = temp;
}// end outer for
```

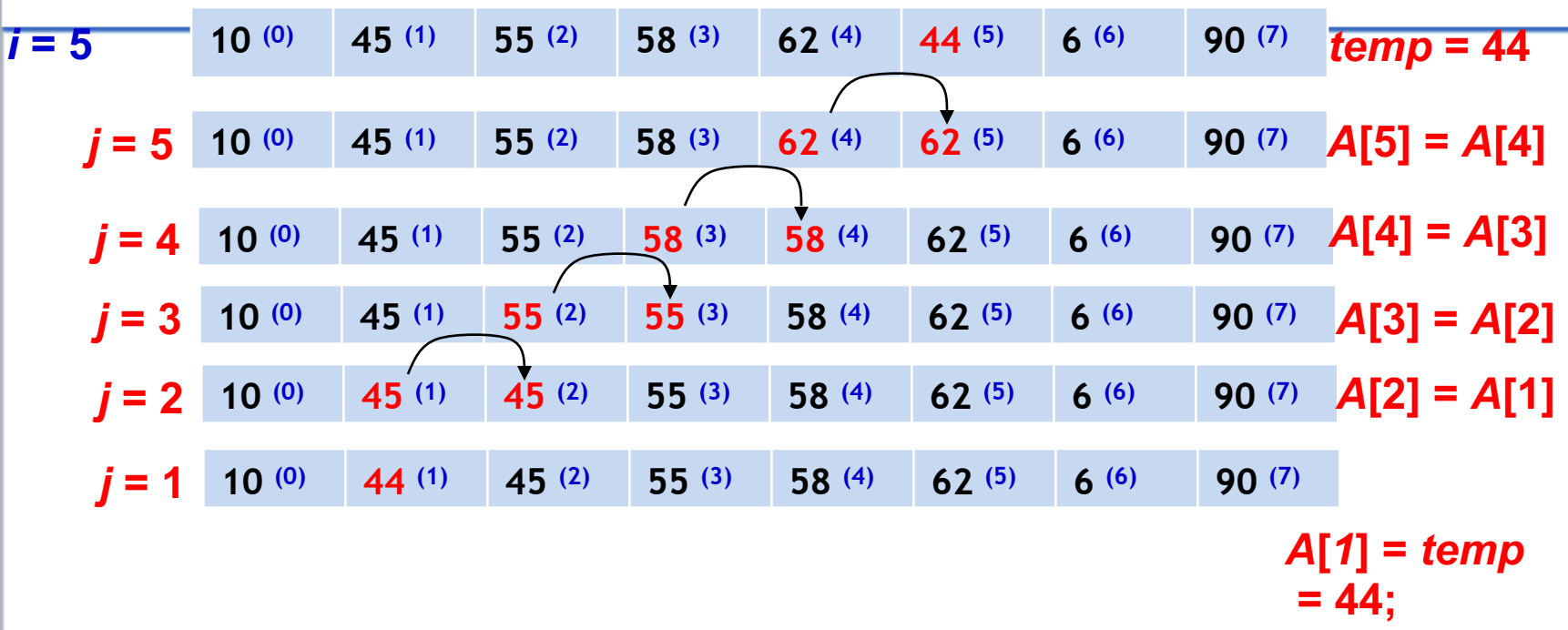
Insertion Sort Example (Fourth Pass)




```
for (int i =1, i < n, i++){
    temp = A[i];
        for (int j = i, j > 0 && A[j-1] > temp, j--)
            A[j] = A[j-1]
    A[j] = temp;
}// end outer for
```


8

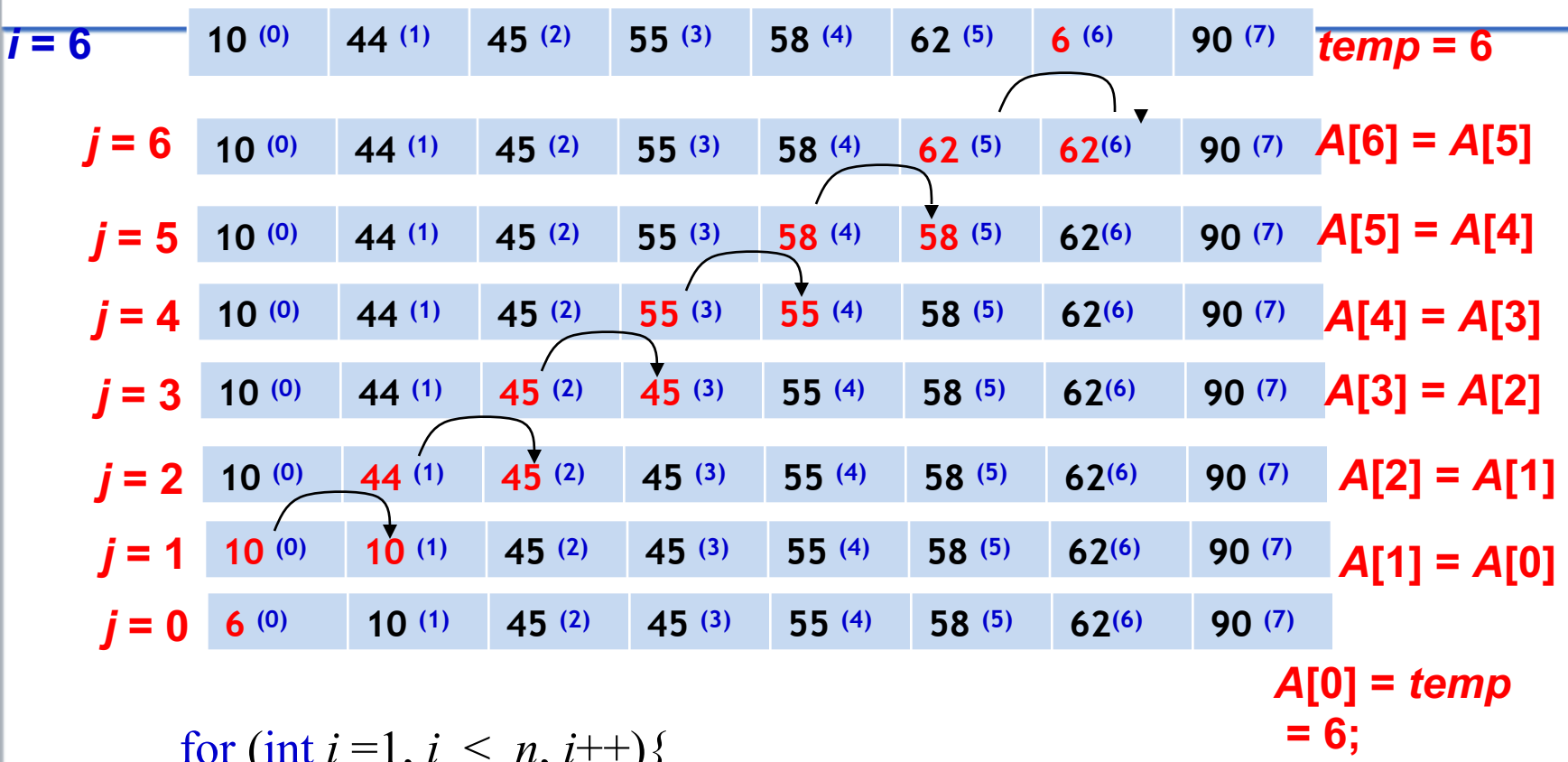
Insertion Sort Example (Fifth Pass)




```
for (int i =1, i  <  n, i++){
    temp = A[i];
        for (int j = i, j > 0  &&  A[j-1] > temp, j--)
            A[j] = A[j-1]
     A[j] = temp;
    }// end outer for
```


9

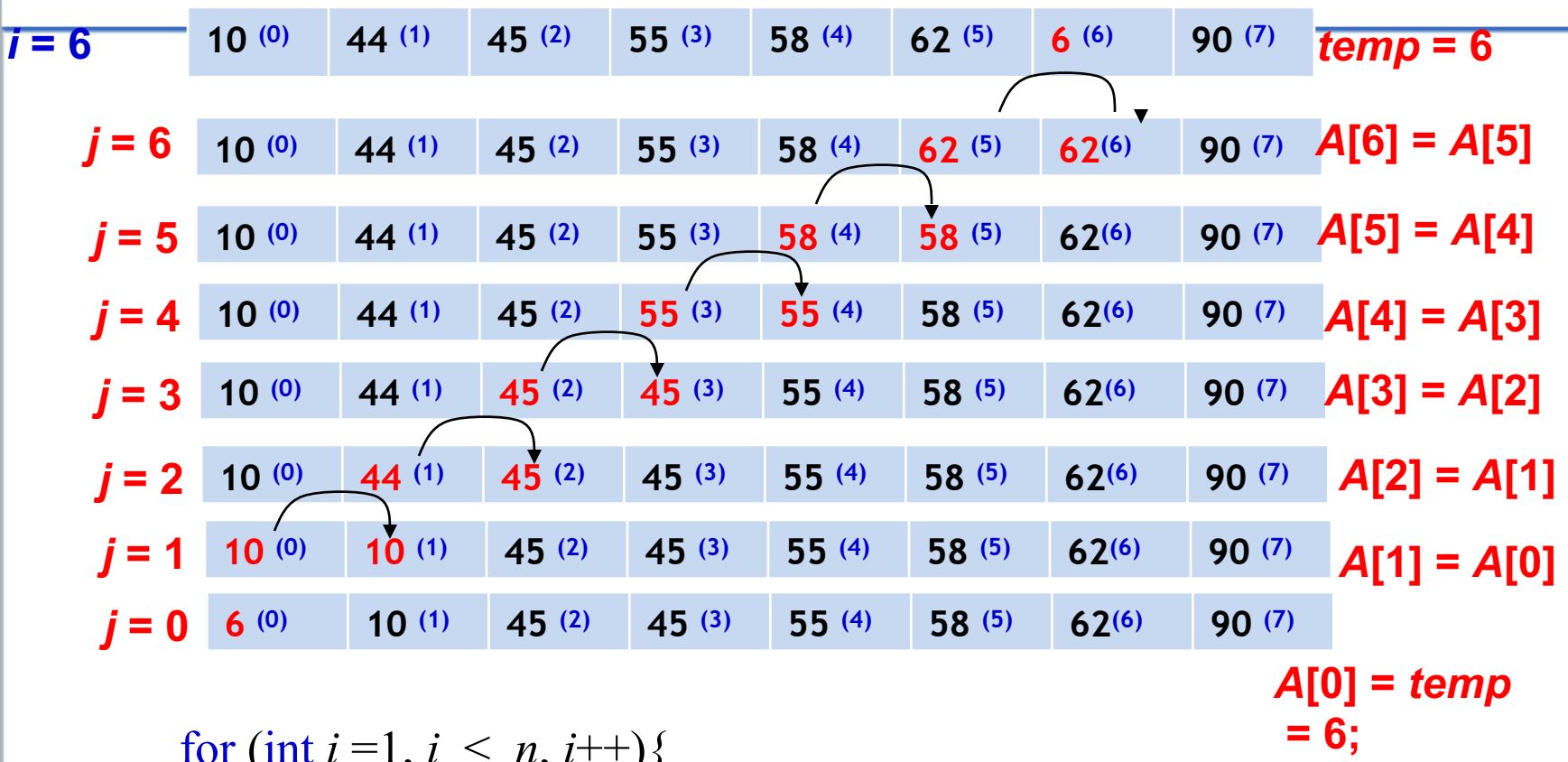
Insertion Sort Example (Sixth Pass)




```
for (int i =1, i < n, i++){
    temp = A[i];
        for (int j = i, j > 0 && A[j-1] > temp, j--)
            A[j] = A[j-1]
    A[j] = temp;
}// end outer for
```


10

Insertion Sort Example (Sixth Pass)




```
for (int i =1, i < n, i++){
    temp = A[i];
        for (int j = i, j > 0 && A[j-1] > temp, j--)
            A[j] = A[j-1]
    A[j] = temp;
}// end outer for
```


11

Insertion Sort Example (Sixth Pass)

i = 7

6 (0)	10 (1)	45 (2)	45 (3)	55 (4)	58 (5)	62(6)	90 (7)

temp = 90

A[j-1] > temp?
No
Quit

```
for (int i =1, i  <  n, i++){
    temp = A[i];
        for (int j = i, j > 0  &&  A[j-1] > temp, j--)
            A[j] = A[j-1]
     A[j] = temp;
    }// end outer for
```


12

Insertion Sort Example (Sixth Pass)

i = 7

6 (0)	10 (1)	45 (2)	45 (3)	55 (4)	58 (5)	62(6)	90 (7)

temp = 90

A[j-1] > temp?
No
Quit

```
for (int i =1, i  <  n, i++){
    temp = A[i];
        for (int j = i, j > 0  &&  A[j-1] > temp, j--)
            A[j] = A[j-1]
     A[j] = temp;
    }// end outer for
```


13

Analysis of Insertion Sort

- Because of the nested loops, each of which can take ***n*** iterations, insertion sort is $O(n^2)$.
- Furthermore, this bound is tight, because input in reverse order can actually achieve this bound.
- A precise calculation shows that the test at line 3 can be executed at most ***i*** times for each value of ***i***. Summing over all ***i*** gives a total of

$$\sum_{i=1}^{n-1} i = 1 + 2 + \ldots + n - 1 = \Theta(n^2)$$

- If the input is presorted, the running time is $O(n)$
  - because the test in the inner *for* loop always fails immediately
- The average running time also $O(n^2)$

14

Selection Sort

- Find the minimum value in the list
- Swap it with the value in the first position
- Repeat the steps above for the remainder of the list (starting at the second position and advancing each time)

15

Selection Sort

```
for (int i =0, i  <  n, i++){
      min = i;
      for (int j = i+1, j < n , j++){
            if (     A[j] < A[min]){
                  min = j
            } // end if
       }// end inner for
       swap(i, min)
 }// end outer for
```

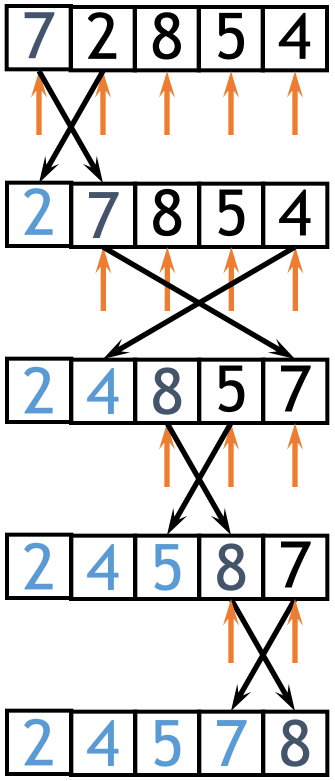
Complexity ?

$O(N^2)$

```
// Swap function assumes that
// A[n] is a globally declared array
swap(i, min) {
 int temp = A[i];
A[i] = A[min];
A[min] = temp;
}
```


16

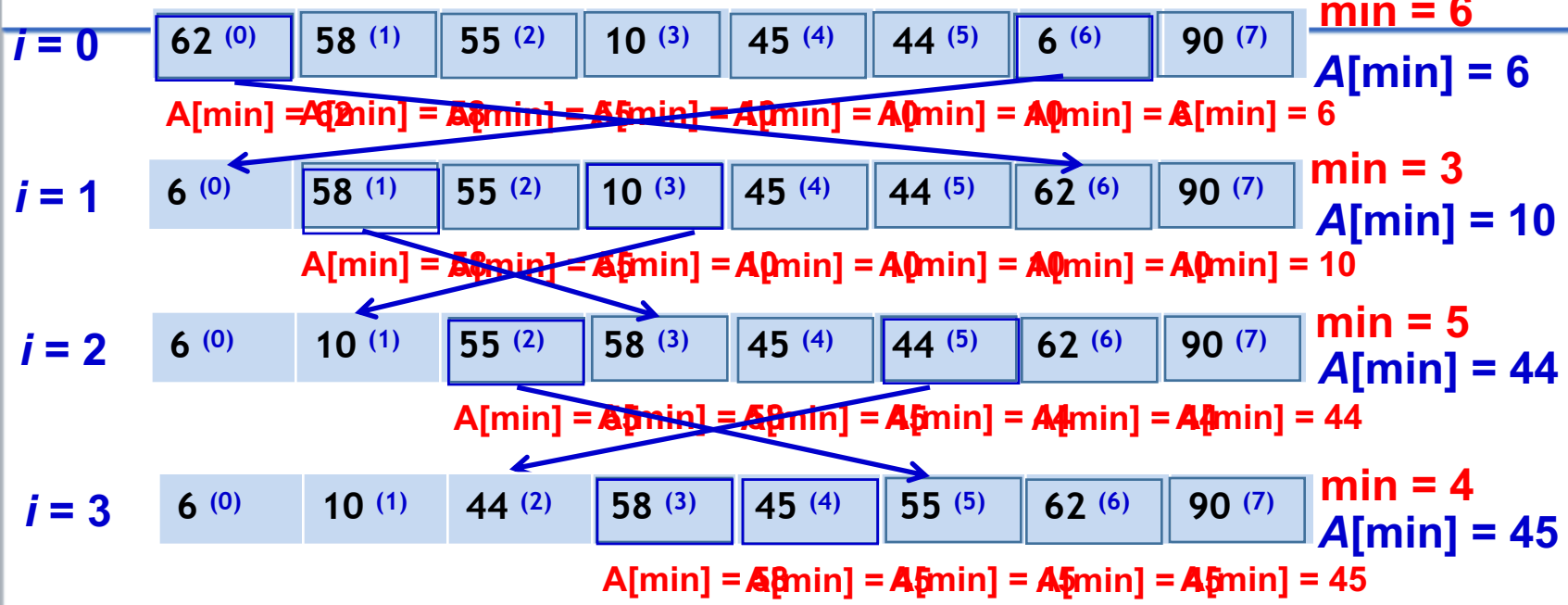
Selection Sort - Example



- The Selection Sort might swap an array element with itself--this is harmless.

```
for (int i =0, i  <  n, i++){
      min = i;
            for (int j = i+1, j < n , j++){
                  if (       A[j] < A[min]){
                        min = j
          } // end if
       }// end inner for
            swap(i, min)
  }// end outer for
```

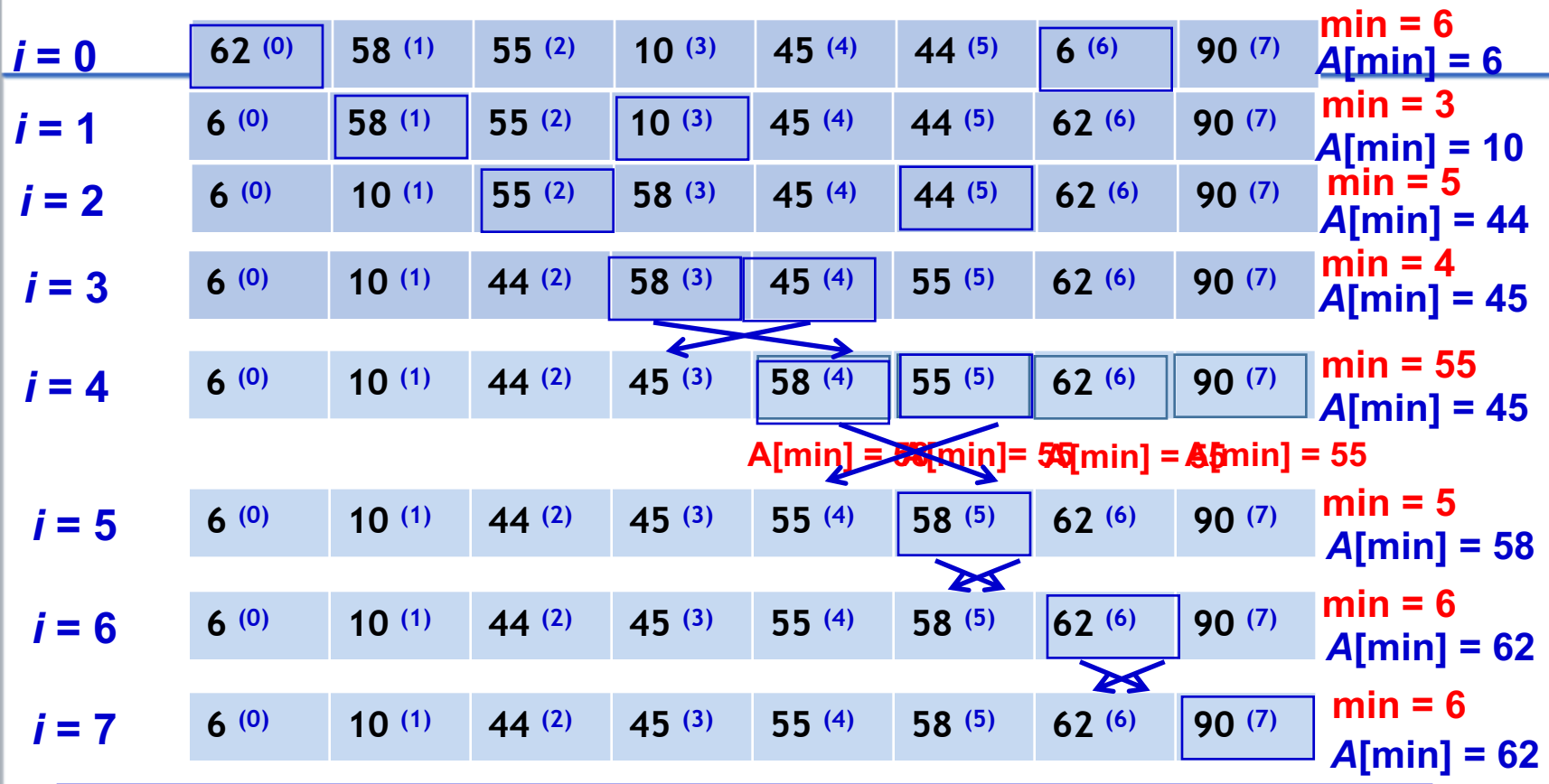
Selection Sort Example




```
for (int i =0, i  <  n, i++){
      min = i;
            for (int j = i+1, j < n , j++){
                  if (       A[j] < A[min]){
                        min = j
        } // end if
      }// end inner for
            swap(i, min)
 }// end outer for
```


18

Selection Sort Example - continued




```
for (int i =0, i  <  n, i++){
      min = i;
                  for (int j = i+1, j < n , j++){
                              if (            A[j] < A[min]){
                                    min = j
        } // end if
      }// end inner for
                  swap(i, min)
 }// end outer for
```


19

Selection Sort vs Insertion Sort

- Selection sort's advantage is that
  - While insertion sort typically makes fewer comparisons than selection sort,
  - Insertion sort requires more writes than the selection sort because the inner loop of the insertion sort can require shifting large sections of the sorted portion of the array.
    - In general, insertion sort will write to the array $O(n^2)$ times
    - Whereas selection sort will write/swap only $O(n)$ times
  - For this reason selection sort may be preferable in cases where writing to memory is significantly more expensive than reading,
    - such as with EPROM or flash memory

20

Comparisons of different sorting algorithms

Bubble Sort	Insertion Sort	Selection Sort
$\Theta(n^2)$ comparisons	$\Theta(n^2)$ comparisons	$\Theta(n^2)$ comparisons
$\Theta(n^2)$ swaps	$\Theta(n^2)$ writes	$\Theta(n)$ swaps
Adaptive: O(n) running time when nearly sorted (Best case running time)	Adaptive: O(n) running time when nearly sorted (Best case running time)	Not adaptive $\Theta(n^2)$ running time when nearly sorted (Best case running time)

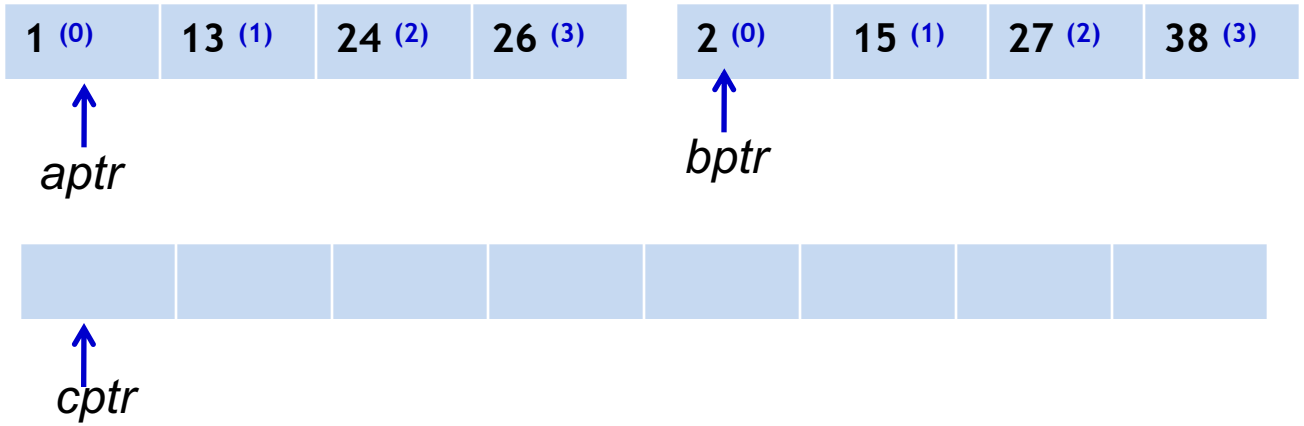
21

Merge Sort

- The fundamental operation in this algorithm is merging two sorted lists.
- Because the lists are sorted, this can be done in one pass through the input, if the output is put in a third list.
- The basic merging algorithm takes
  - two input arrays: *a* and *b*,
  - an output array: *c*
  - three counters: *aptr*, *bptr*, and *cptr*,
    - which are initially set to the beginning of their respective arrays.
- The smaller of *a*[*aptr*] and *b*[*bptr*] is copied to the next entry in *c*, and the appropriate counters are advanced.
- When either input list is exhausted, the remainder of the other list is copied to *c*.

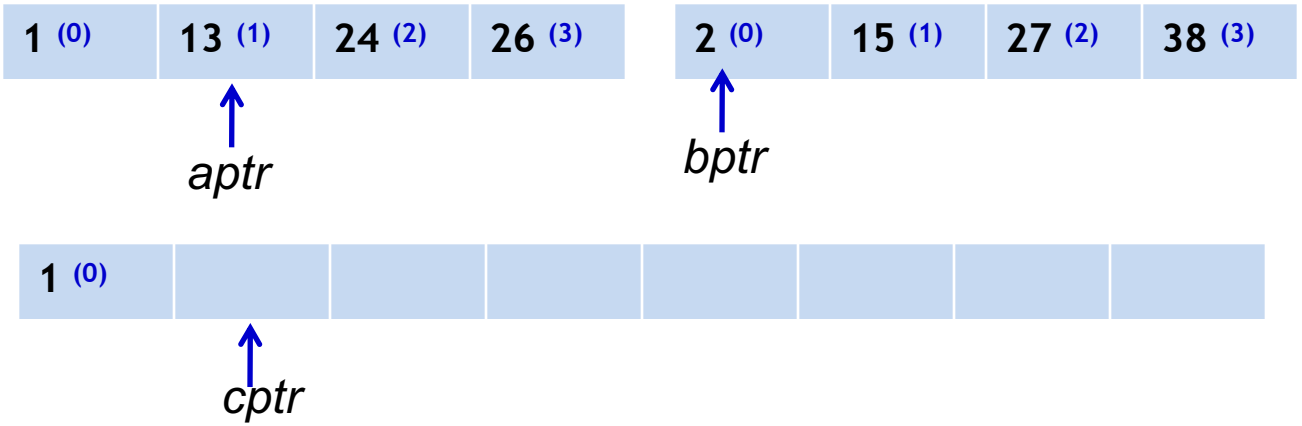
22

Merge Sort : Example



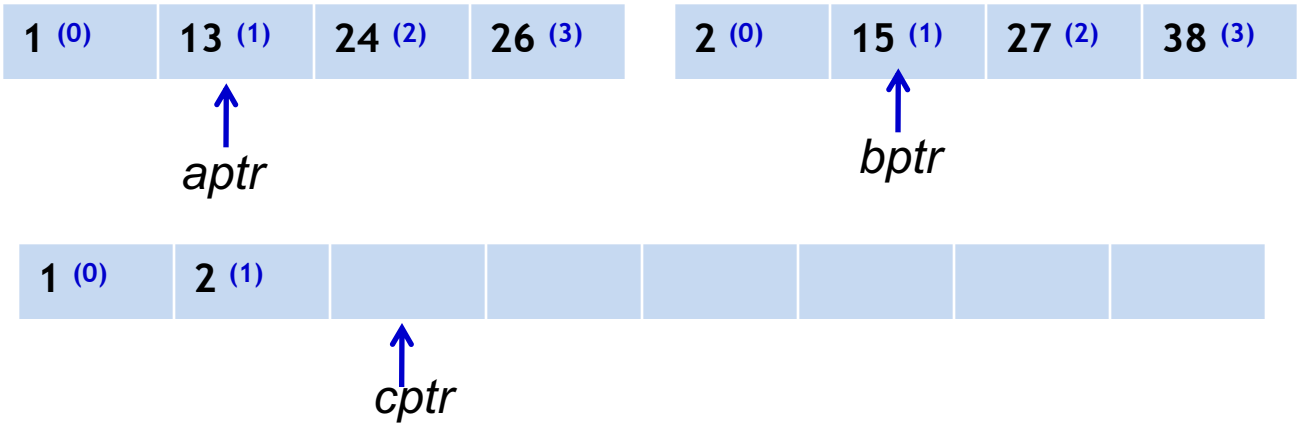
23

Merge Sort : Example



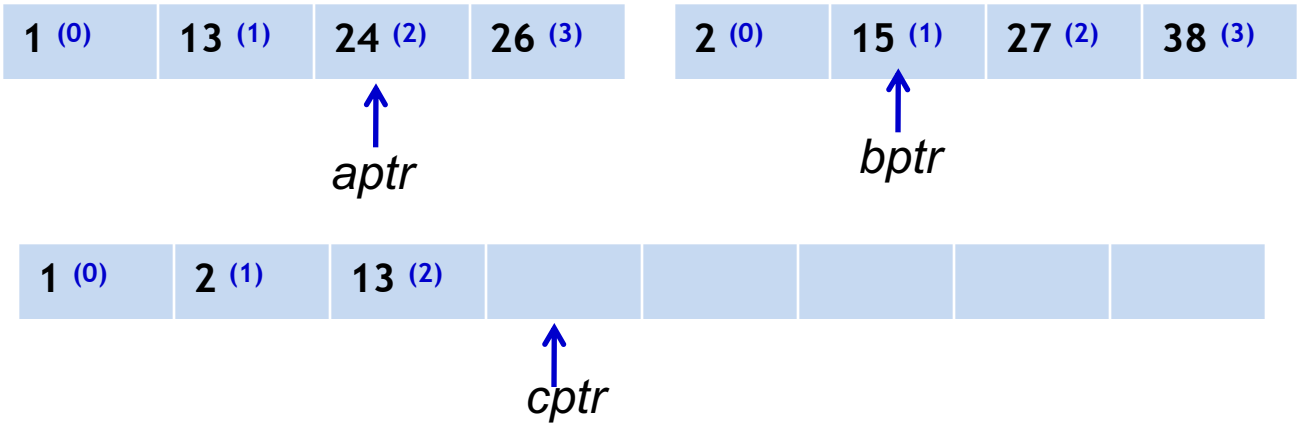
24

Merge Sort : Example



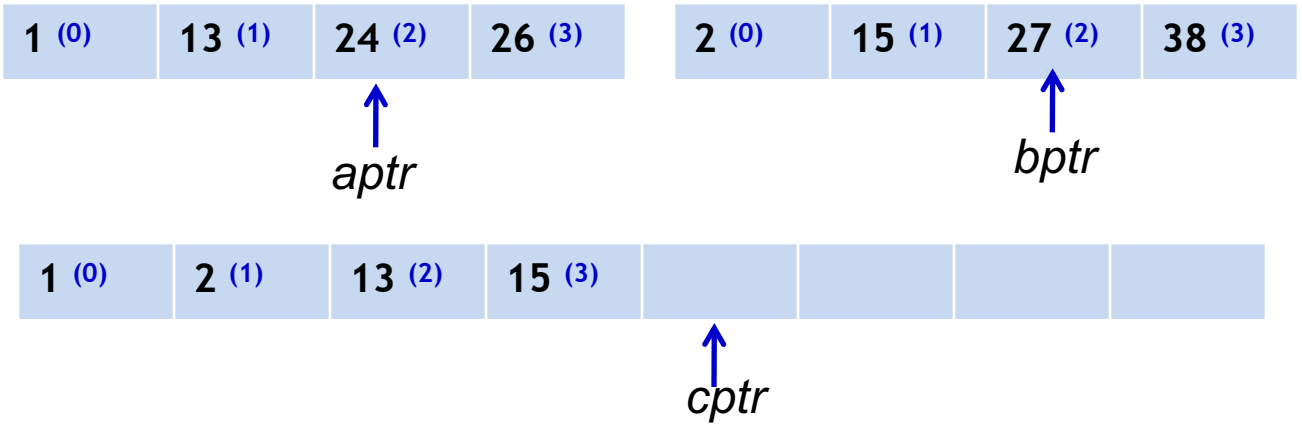
25

Merge Sort : Example



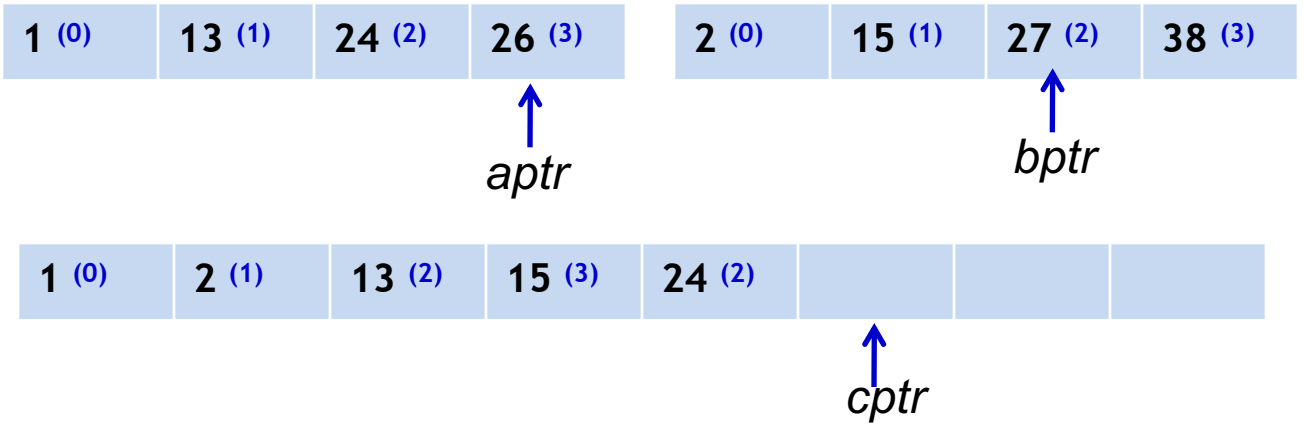
26

Merge Sort : Example



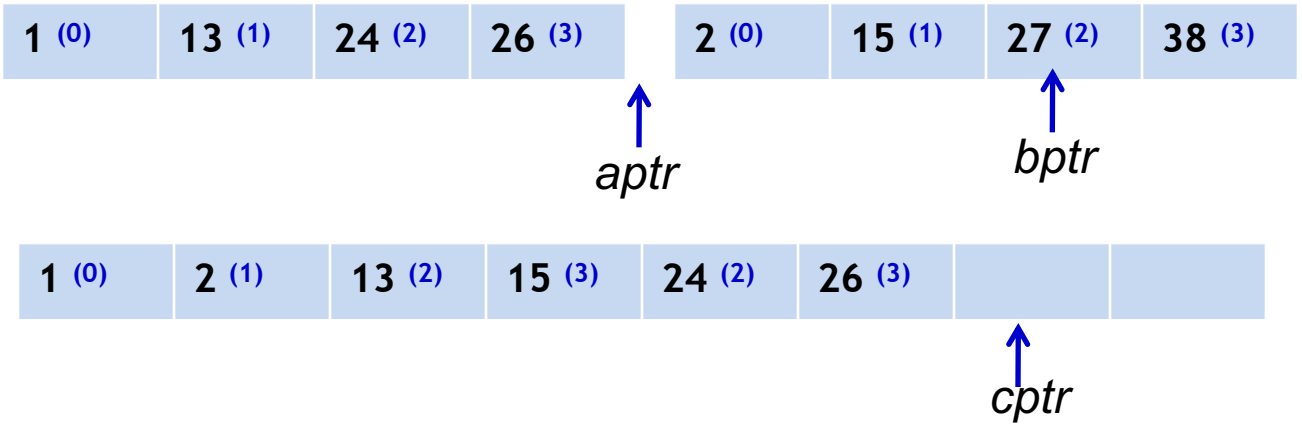
27

Merge Sort : Example



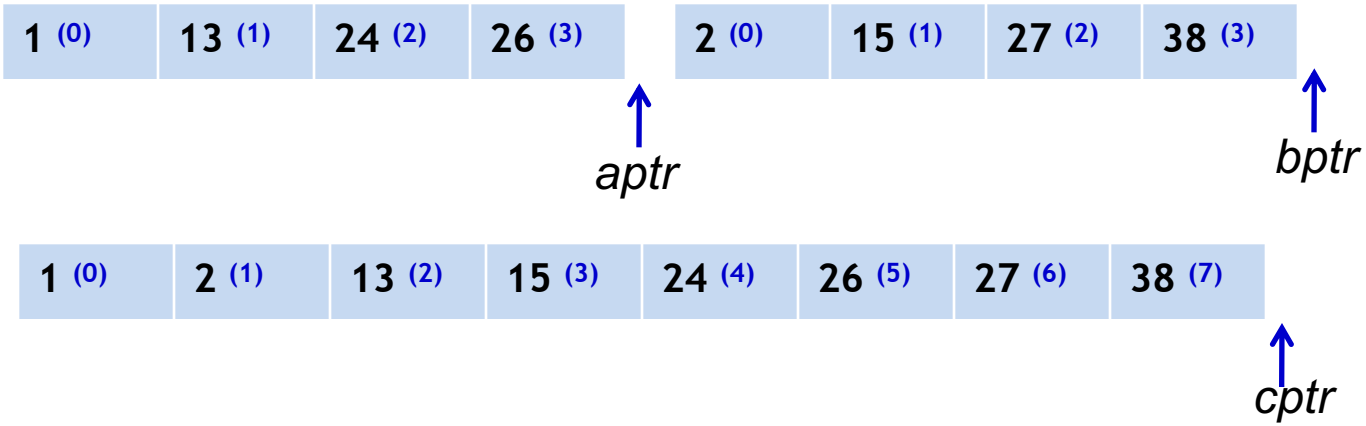
28

Merge Sort : Example



29

Merge Sort : Example



30

Merge Sort

```
void m_sort( input_type a[], input_type tmp_array[ ], int left, int right )
{
int center;
if( left < right )
   {
   center = (left + right) / 2;
   m_sort( a, tmp_array, left, center );
   m_sort( a, tmp_array, center+1, right );
   merge( a, tmp_array, left, center+1, right );
   }
}
```

Calculate the **centre** index of the input list

Recursively call the **m_sort** procedure for t

Recursively call the **m_sort** procedure for the **right-half** of the input data

Merge the two sorted lists

31

Questions?

zahmaad.github.io