SORT

Insertion Sort

Sort

INSERTION SORT

Insertion Sort

คือการจัดเรียงด้วยการแทรกตัวเองไปอยู่หลังตัวที่มีค่าน้อยกว่าตัวเอง

Code of Insertion sort

```
static void insertionSort(int[] arr){
  int n = arr.length;
  for (int curIndex = 1; curIndex < n; curIndex++) {
    int current = arr[curIndex];          <------   ตัวปัจจุบันที่กำลังเช็ค
    int i = curIndex-1;
    while (i>= 0 && arr[i] > current) {
      arr[i+1] = arr[i];
      i--;
    }
  }
  arr[i+1] = current;                     <------   แทรกตัวเองลงไปแทน
}
```

} Loop สลับตำแหน่งไปเรื่อยๆ

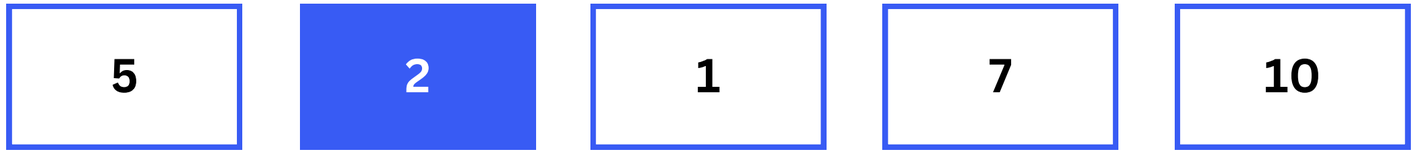
Insertion Sort

Loop รอบ แรก

arr =

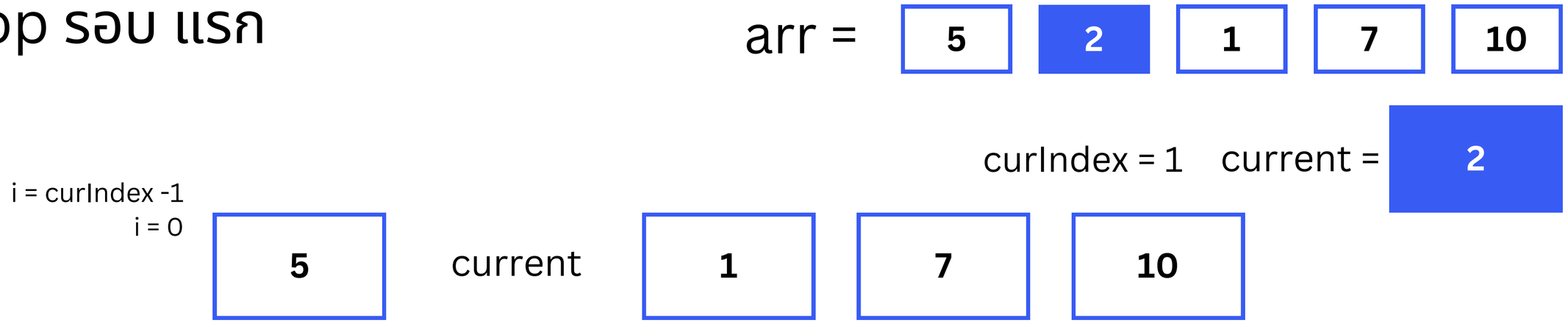
curIndex = 1 current = arr[curIndex]

i = curIndex -1
i = 0



Insertion Sort

Loop รอบ แรก



Insertion Sort

Loop รอบ แรก

arr = | 5 | 2 | 1 | 7 | 10 |

curIndex = 1 current = 2

i = curIndex -1
i = 0

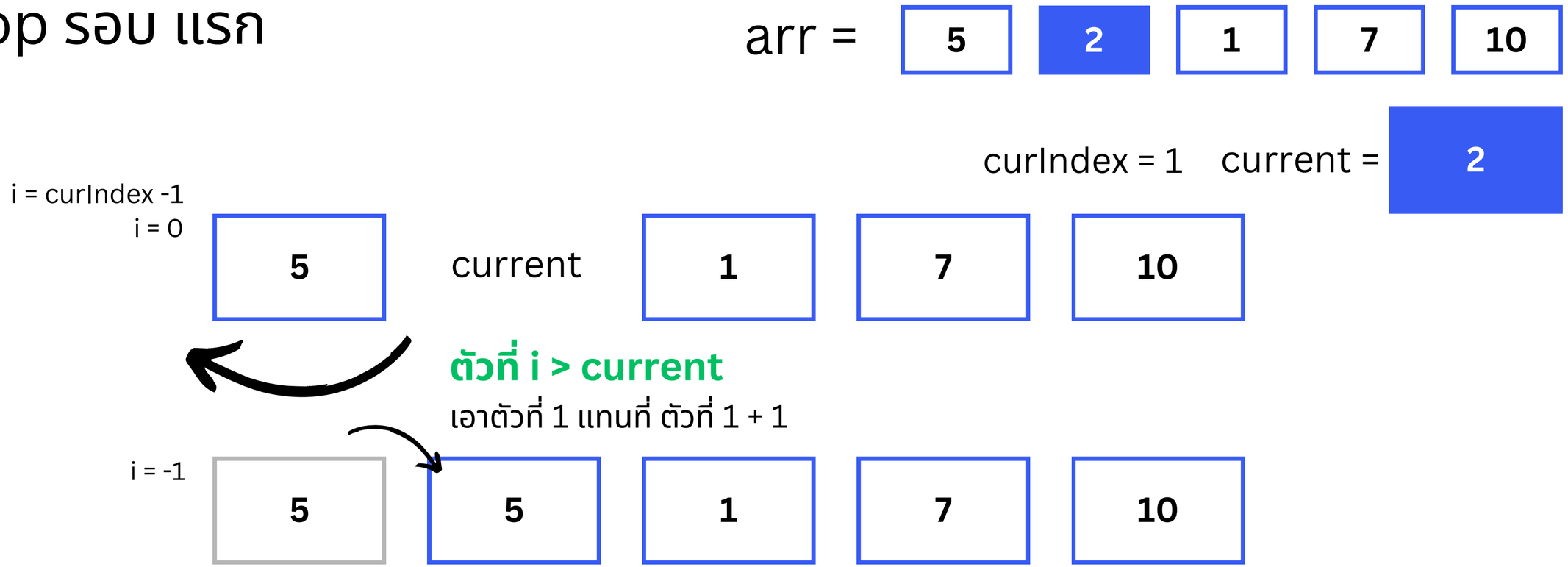
| 5 | current | 1 | 7 | 10 |

ตัวที่ i > current

เอาตัวที่ 1 แทนที่ ตัวที่ 1 + 1

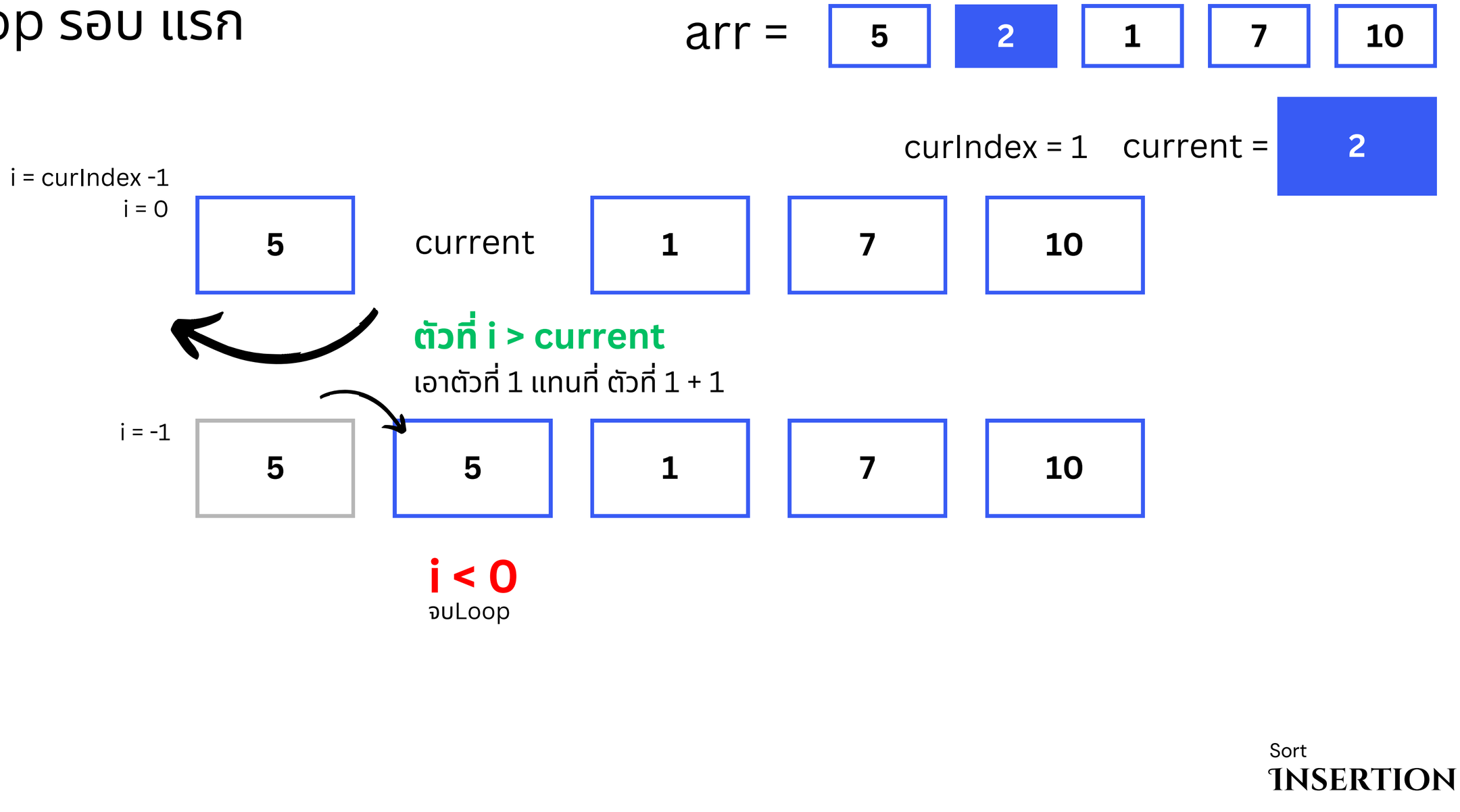
Insertion Sort

Loop รอบ แรก



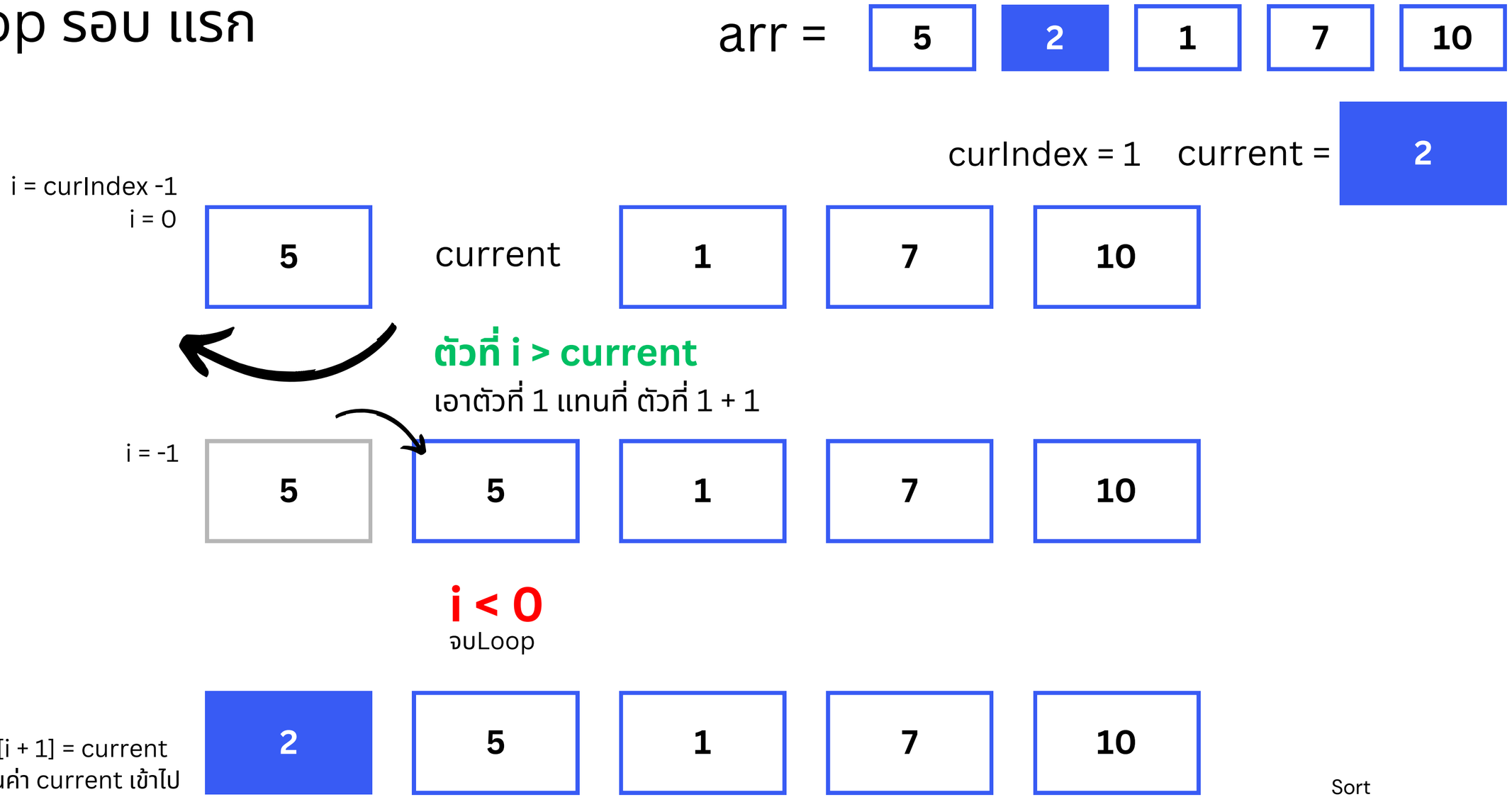
Insertion Sort

Loop รอบ แรก



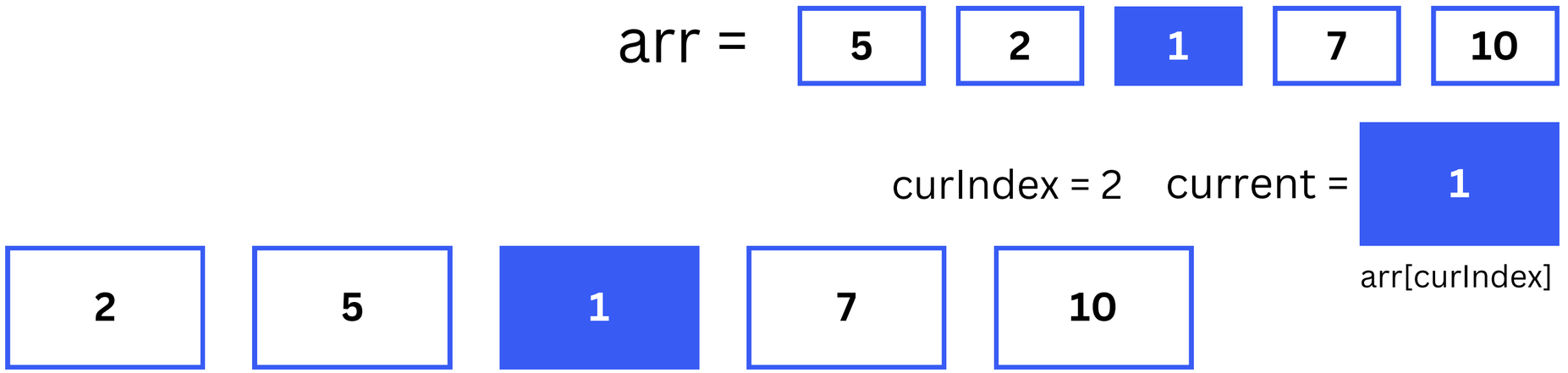
Insertion Sort

Loop รอบ แรก



Insertion Sort

Loop รอบ 2



Insertion Sort

Loop รอบ 2

arr = | 2 | 5 | 1 | 7 | 10 |

curIndex = 2 current = 1

arr[curIndex]

i = curIndex -1

i = 1

| 2 | 5 | current | 7 | 10 |

Insertion Sort

Loop รอบ 2

arr = | 2 | 5 | 1 | 7 | 10 |

curIndex = 2 current = 1

arr[curIndex]

i = curIndex -1

i = 1

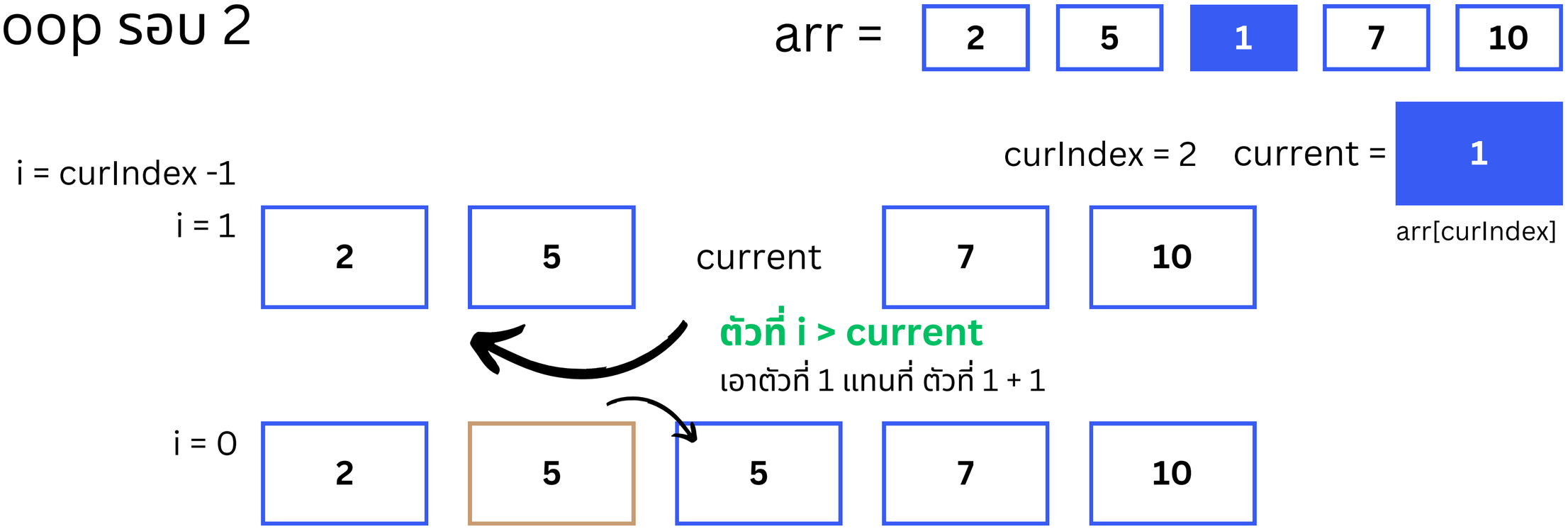
| 2 | 5 | current | 7 | 10 |

ตัวที่ i > current

เอาตัวที่ 1 แทนที่ ตัวที่ 1 + 1

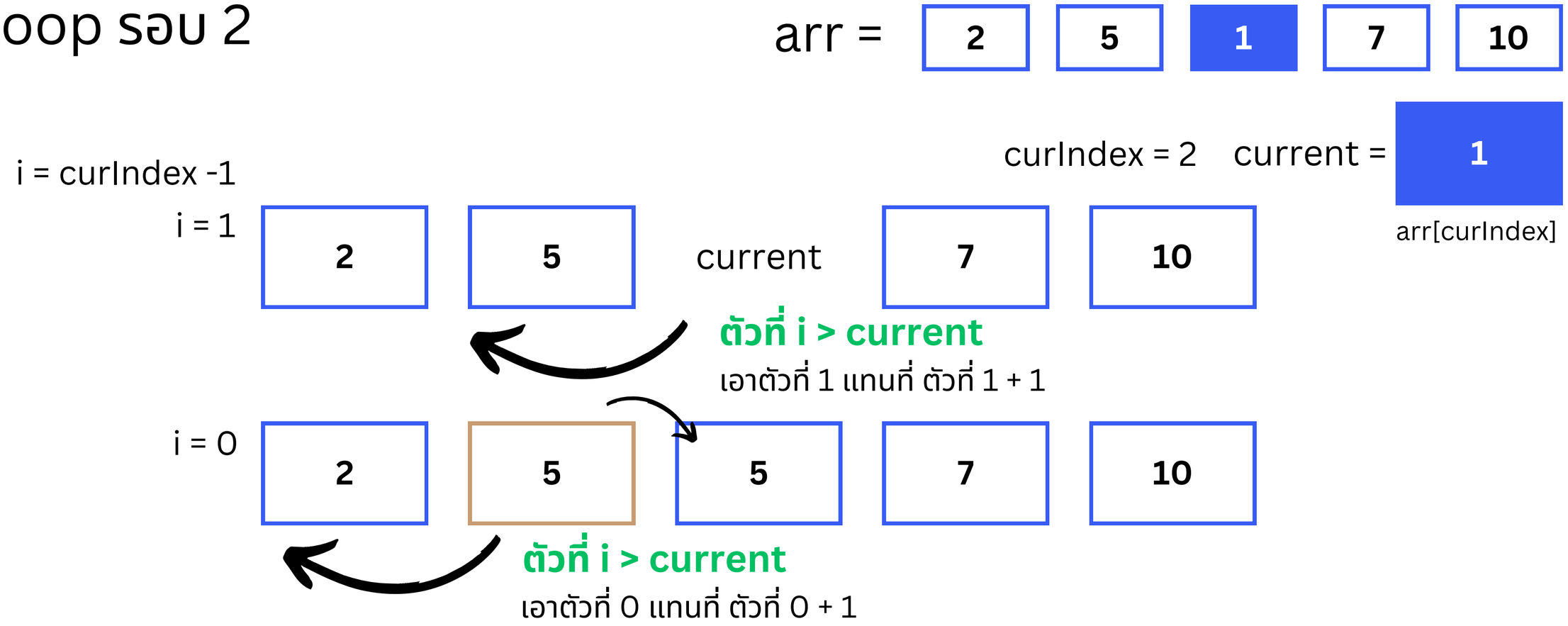
Insertion Sort

Loop รอบ 2



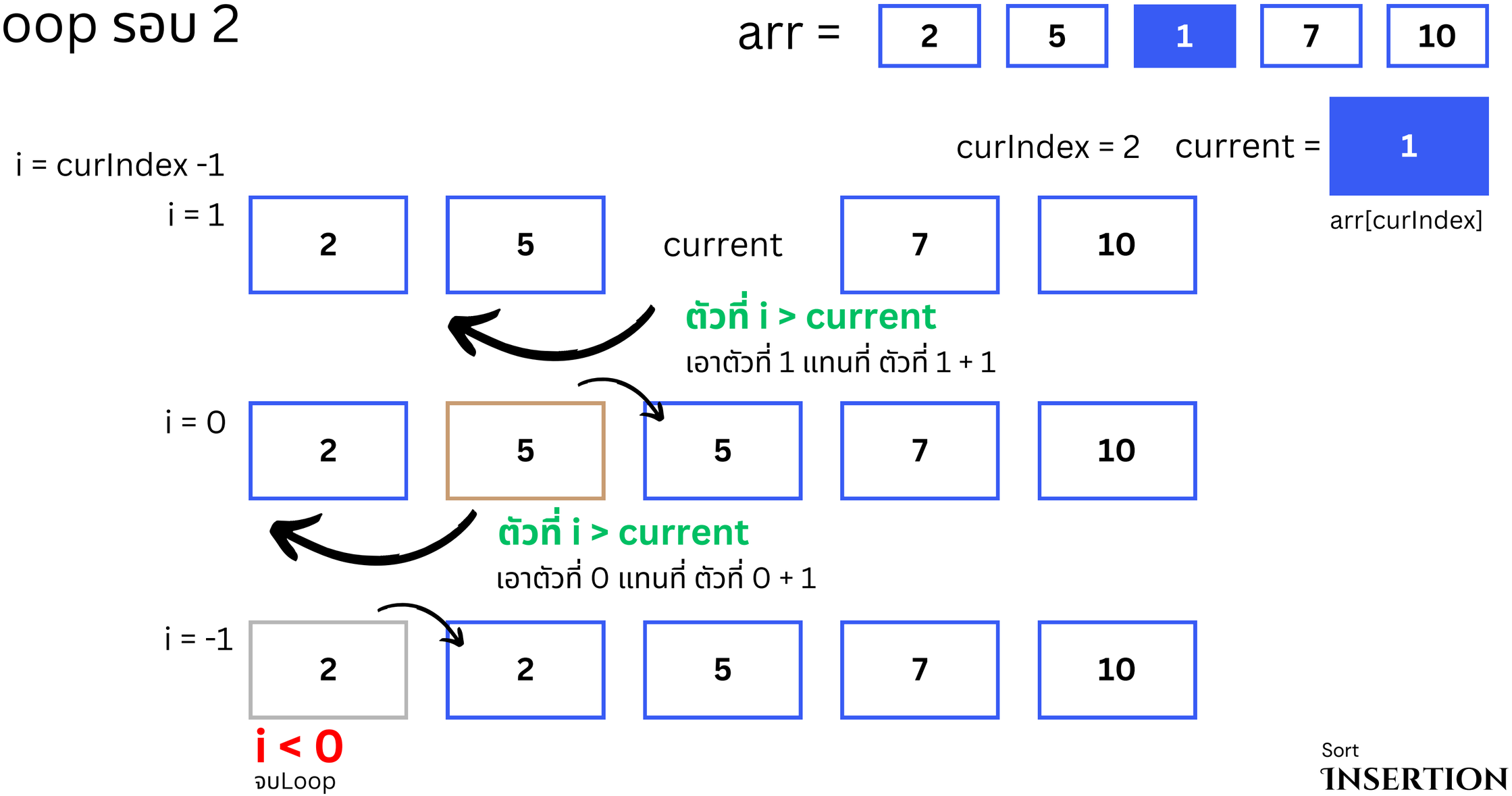
Insertion Sort

Loop รอบ 2



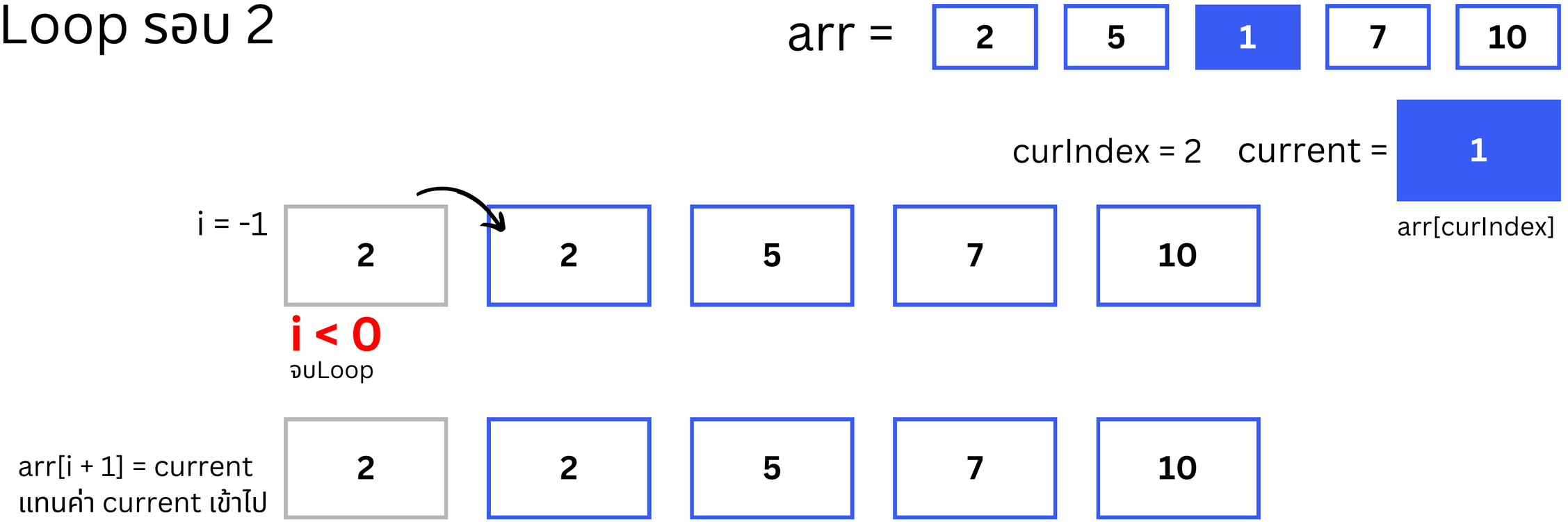
Insertion Sort

Loop รอบ 2



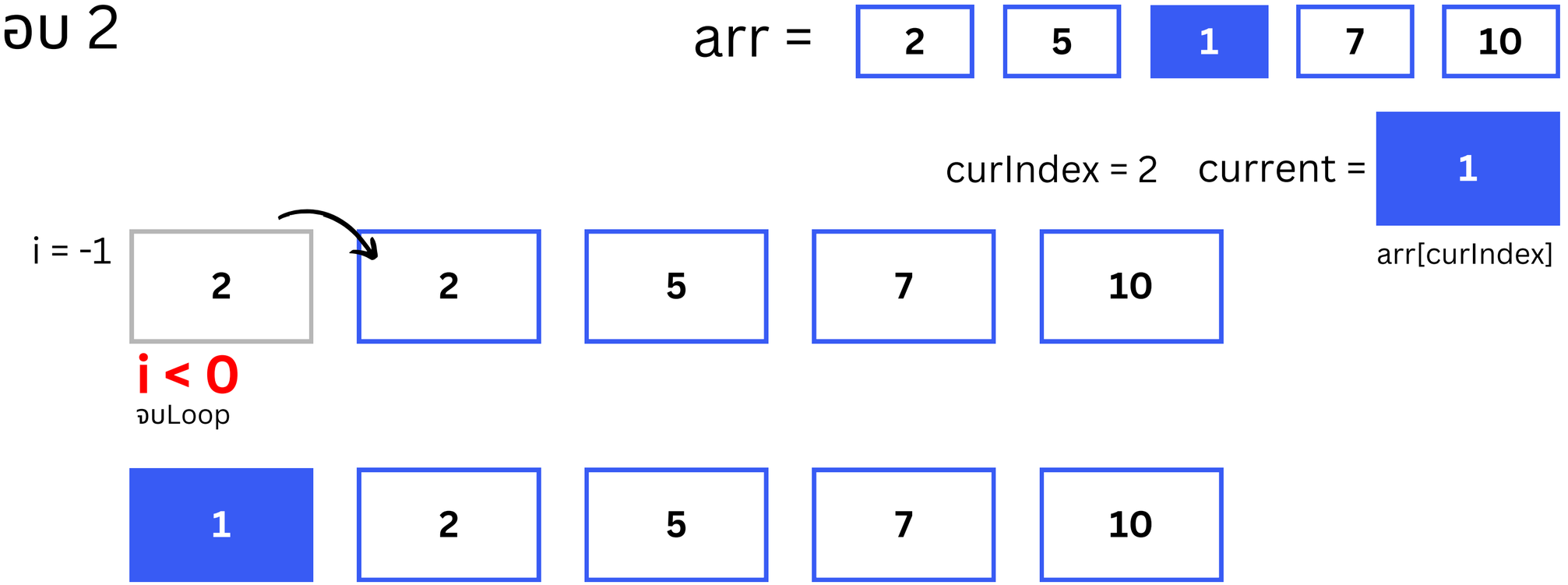
Insertion Sort

Loop รอบ 2



Insertion Sort

Loop รอบ 2



ทำจนจบ

Quick Sort

คือการหาตำแหน่งที่ถูกต้องของสมาชิกแต่ละตัวใน array โดยใช้หลักการของ **Divide and Conquer** ในการแยกส่วน array

Quick Sort

Divide and Conquer

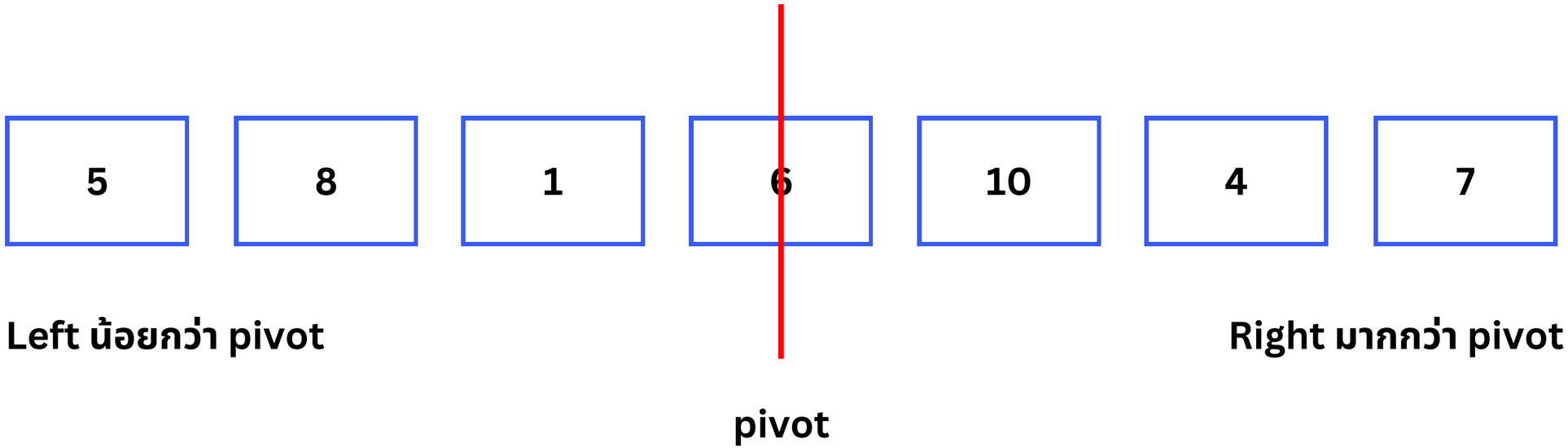
Divide : การแบ่งปัญหาออกมาเป็นส่วนย่อยๆก่อนการแก้ปัญหา

Conquer : แก้ปัญหาในแต่ละส่วนย่อยๆนั้นด้วย Recursive จนสามารถแบ่งปัญหานั้นจนเล็กที่สุด (Base case)

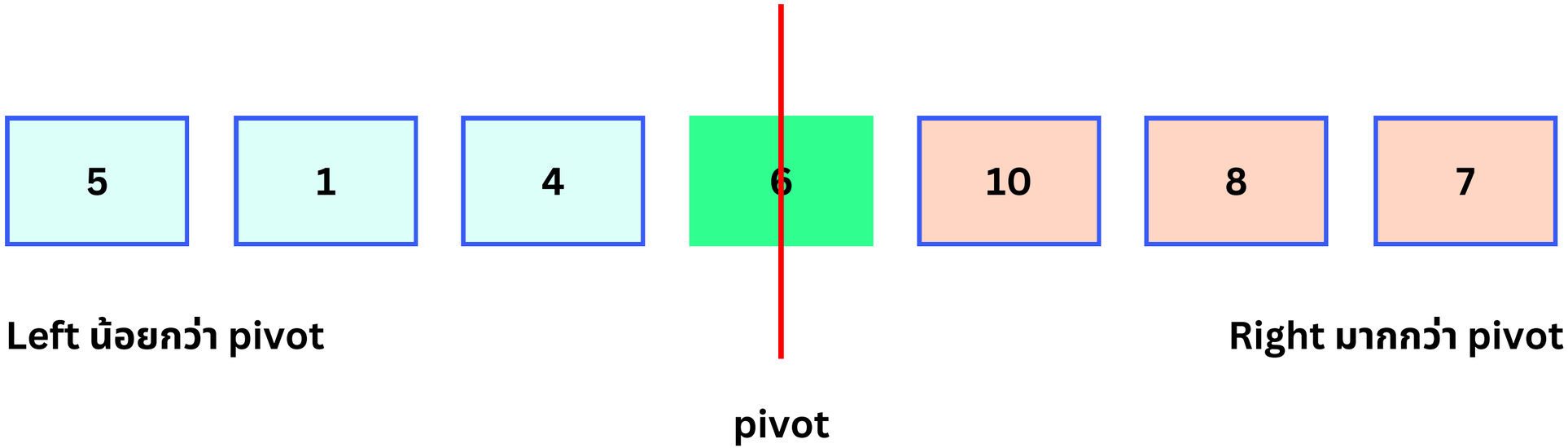
Combine : เอาปัญหาที่แก้ทั้งหมดมารวมกัน

ref : https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm/

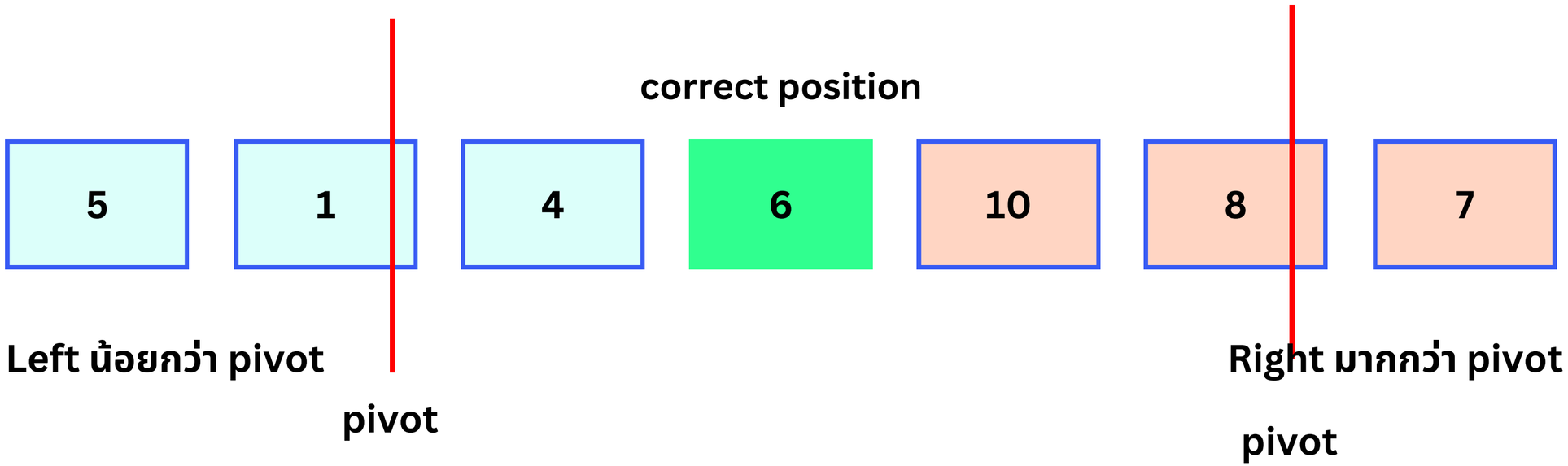
Quick Sort



Quick Sort



Quick Sort


correct position
5
1
4
6
10
8
7
Left น้อยกว่า pivot
pivot
Right มากกว่า pivot
pivot

Quick Sort

Code จะแบ่งเป็น 3 ส่วน

```
static int partition(int[] arr, int low, int high){
  int pivot = arr[high];
  int i = (low - 1);
  for (int j = low; j <= high - 1; j++) {
    if (arr[j] < pivot) {
      i++;
      swap(arr, i, j);
    }
  }
  swap(arr, i + 1, high);
  return (i + 1);
}
```

```
static void swap(int[] arr, int i, int j)
 {
   int temp = arr[i];
   arr[i] = arr[j];
   arr[j] = temp;
 }
```

```
static void quickSort(int[] arr, int low, int high)
 {
   if (low < high) {
     int pi = partition(arr, low, high);

     quickSort(arr, low, pi - 1);
     quickSort(arr, pi + 1, high);
   }
 }
```

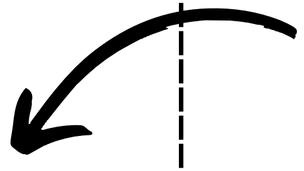
Quick Sort

```
static void quickSort(int[] arr, int low, int high)
 {
   if (low < high) {
     int pi = partition(arr, low, high);

     quickSort(arr, low, pi - 1);
     quickSort(arr, pi + 1, high);
   }
 }
```

Quick Sort

quickSort(arr, 0, arr.length - 1);

```
static void quickSort(int[] arr, int low, int high)
 {
   if (low < high) {
     int pi = partition(arr, low, high);

     quickSort(arr, low, pi - 1);
     quickSort(arr, pi + 1, high);
   }
 }
```

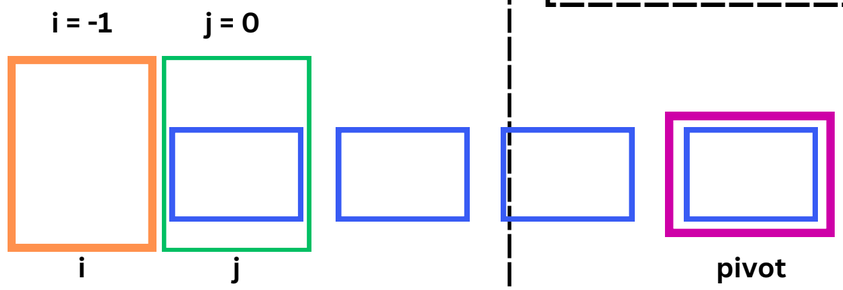


recursion

Quick Sort

Partition

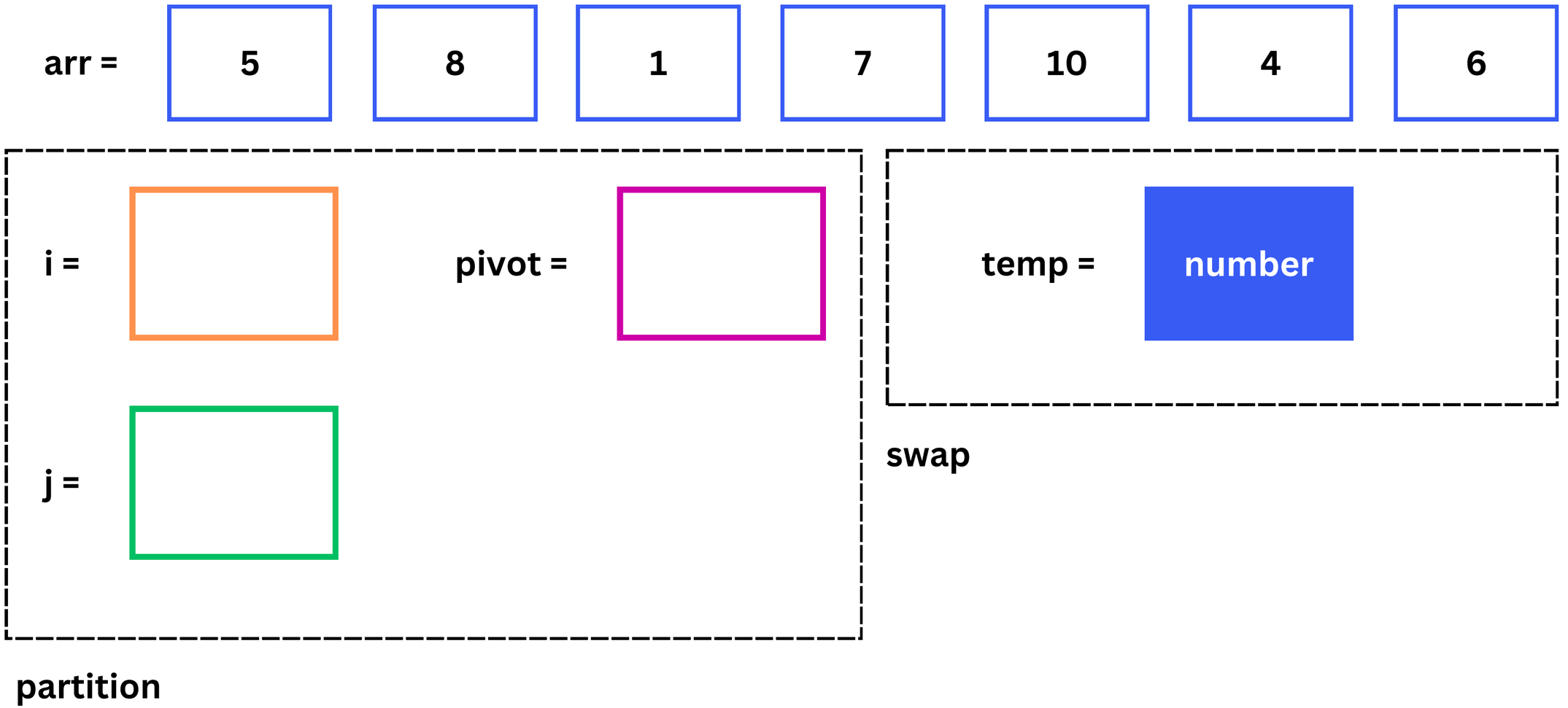
```
static int partition(int[] arr, int low, int high){
  int pivot = arr[high];
  int i = (low - 1);
  for (int j = low; j <= high - 1; j++) {
    if (arr[j] < pivot) {
      i++;
      swap(arr, i, j);
    }
  }
  swap(arr, i + 1, high);
  return (i + 1);
}
```



Swap

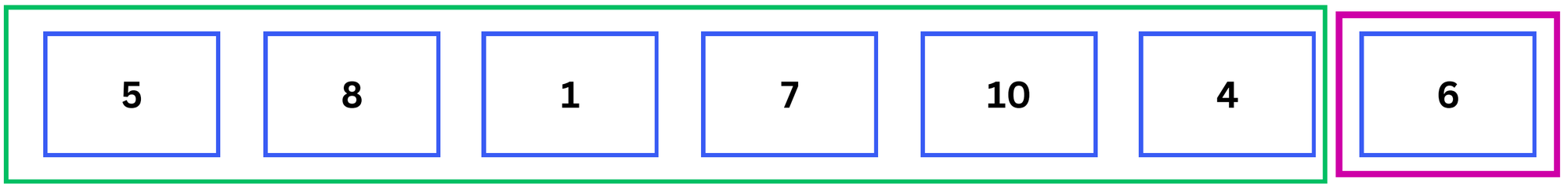
```
static void swap(int[] arr, int i, int j)
 {
   int temp = arr[i];
   arr[i] = arr[j];
   arr[j] = temp;
 }
```

Quick Sort

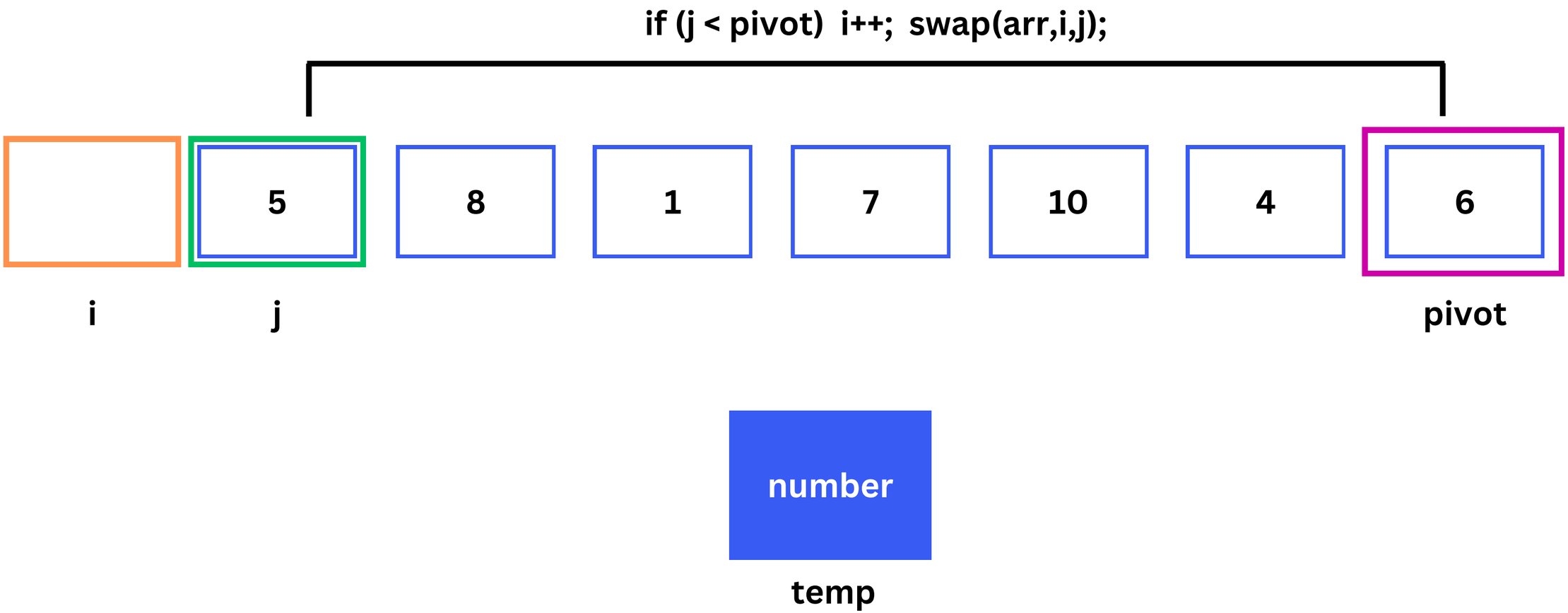


Quick Sort

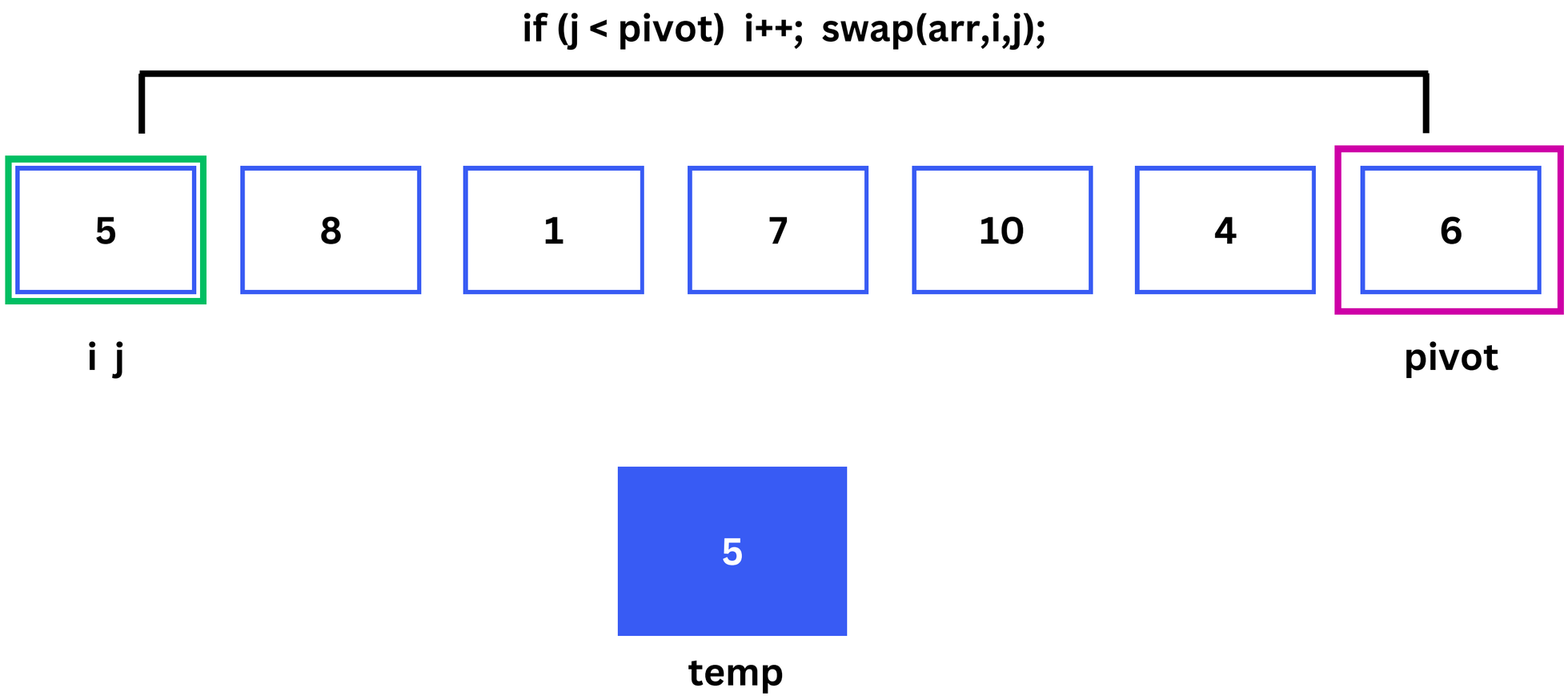
สลับที่โดยให้ฝั่งซ้ายเป็นค่าน้อยกว่า pivotและฝั่งขวาเป็นค่ามากกว่า pivot



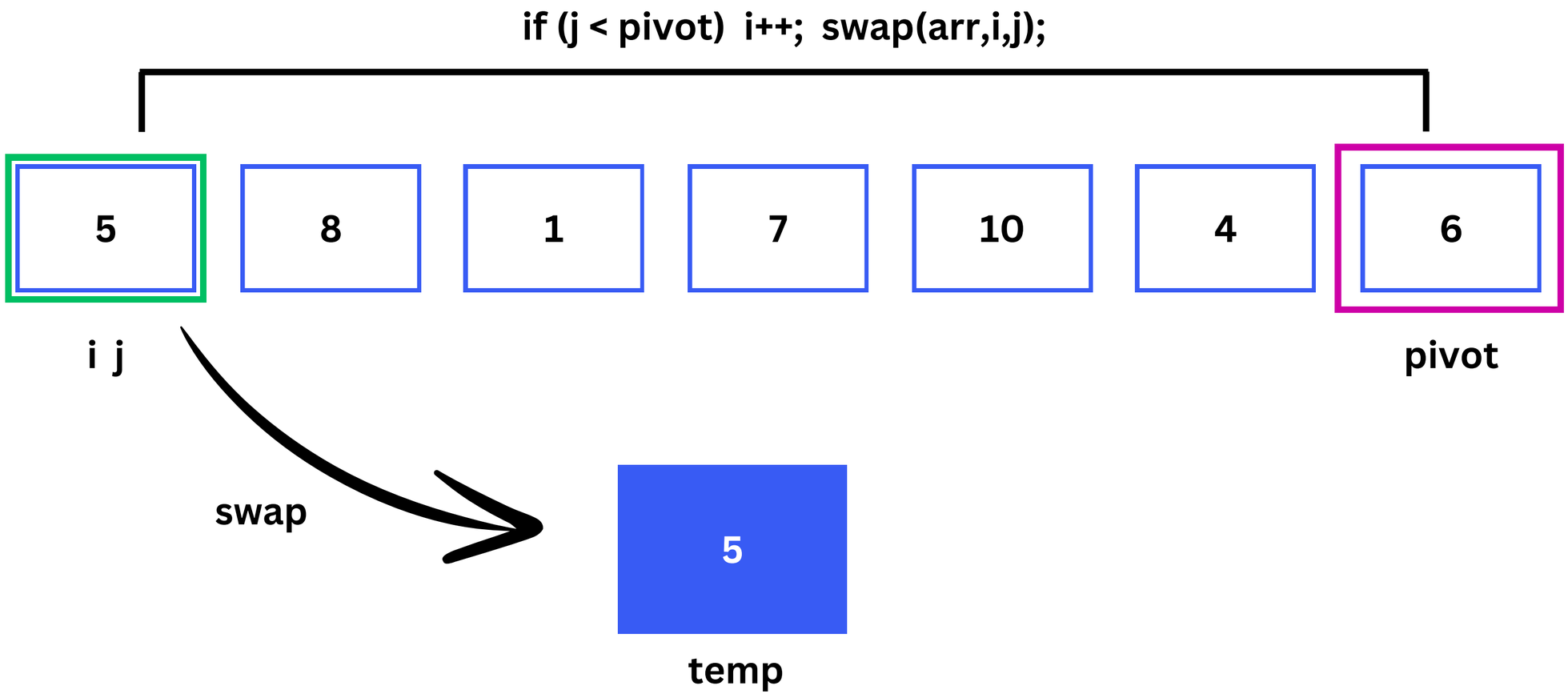
Quick Sort



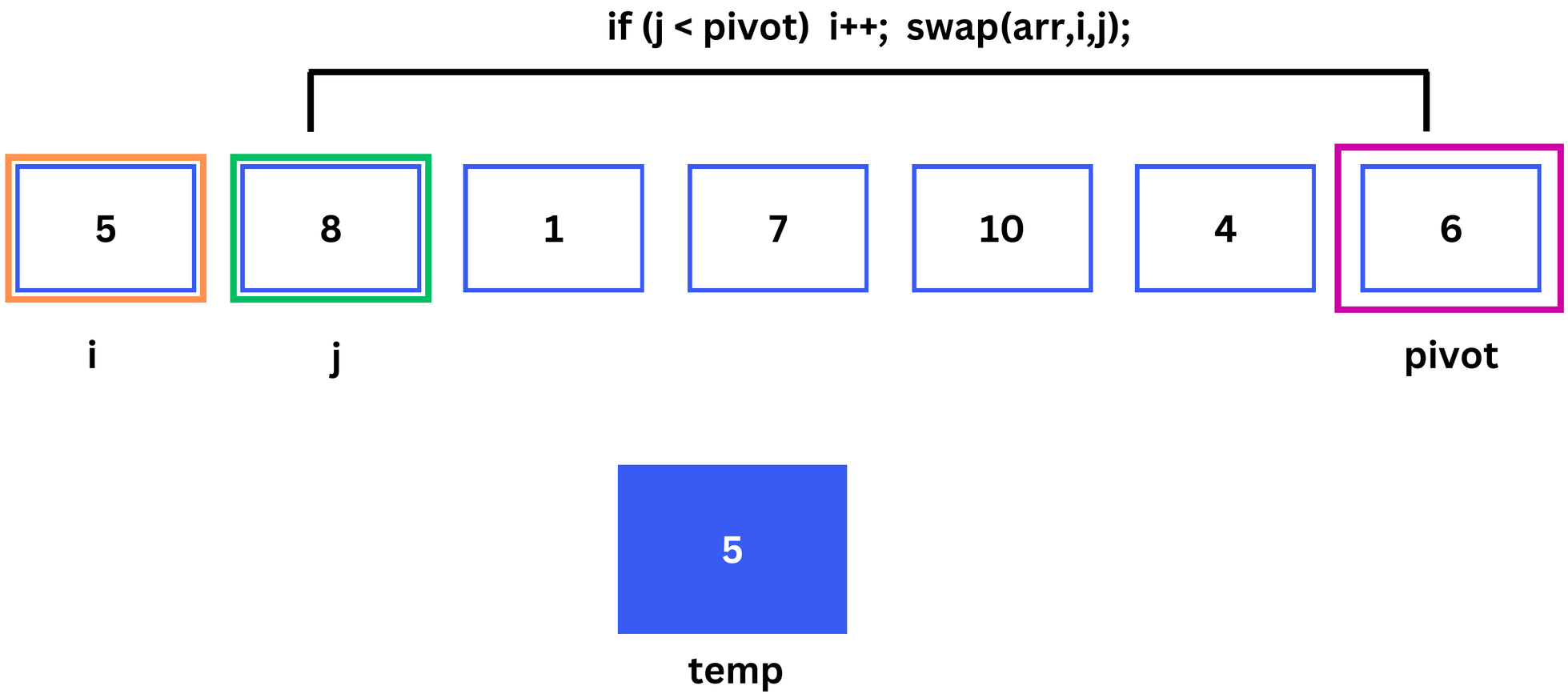
Quick Sort



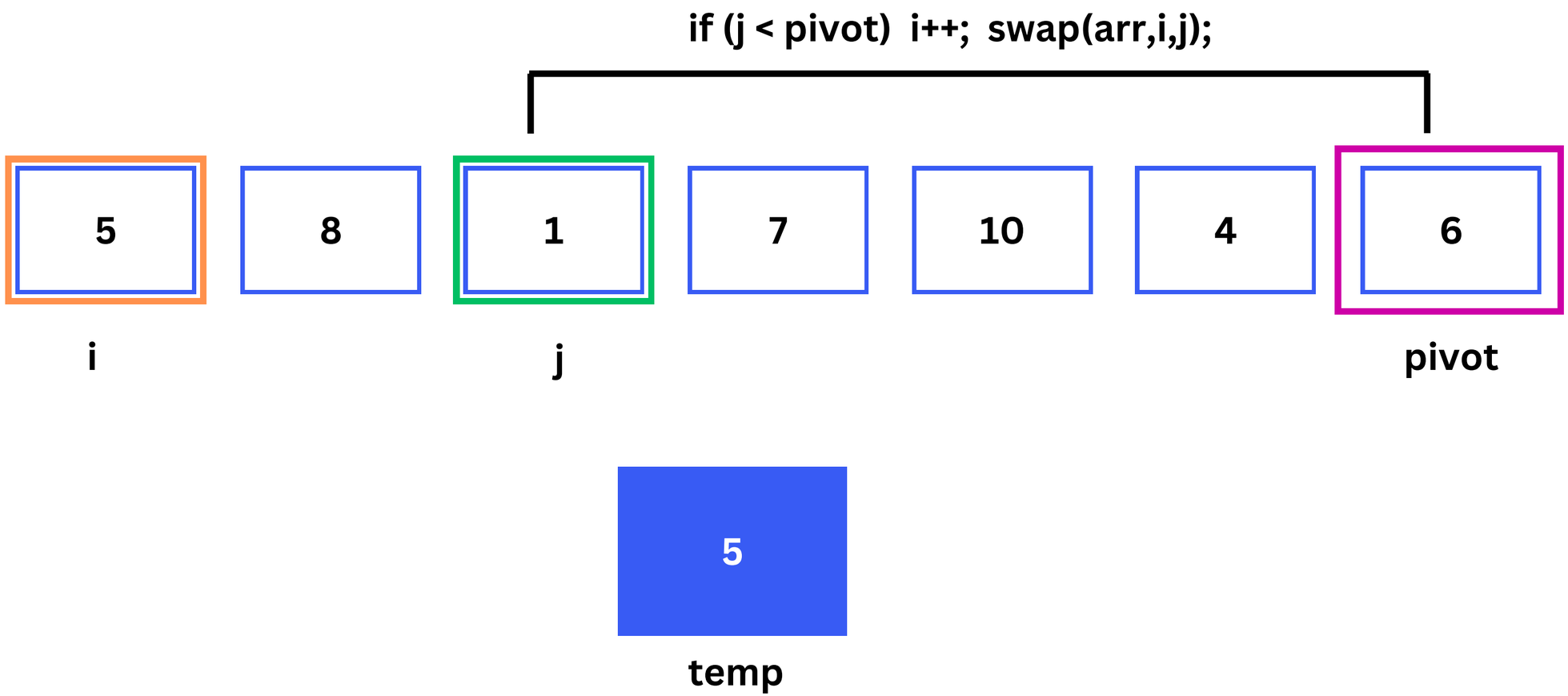
Quick Sort



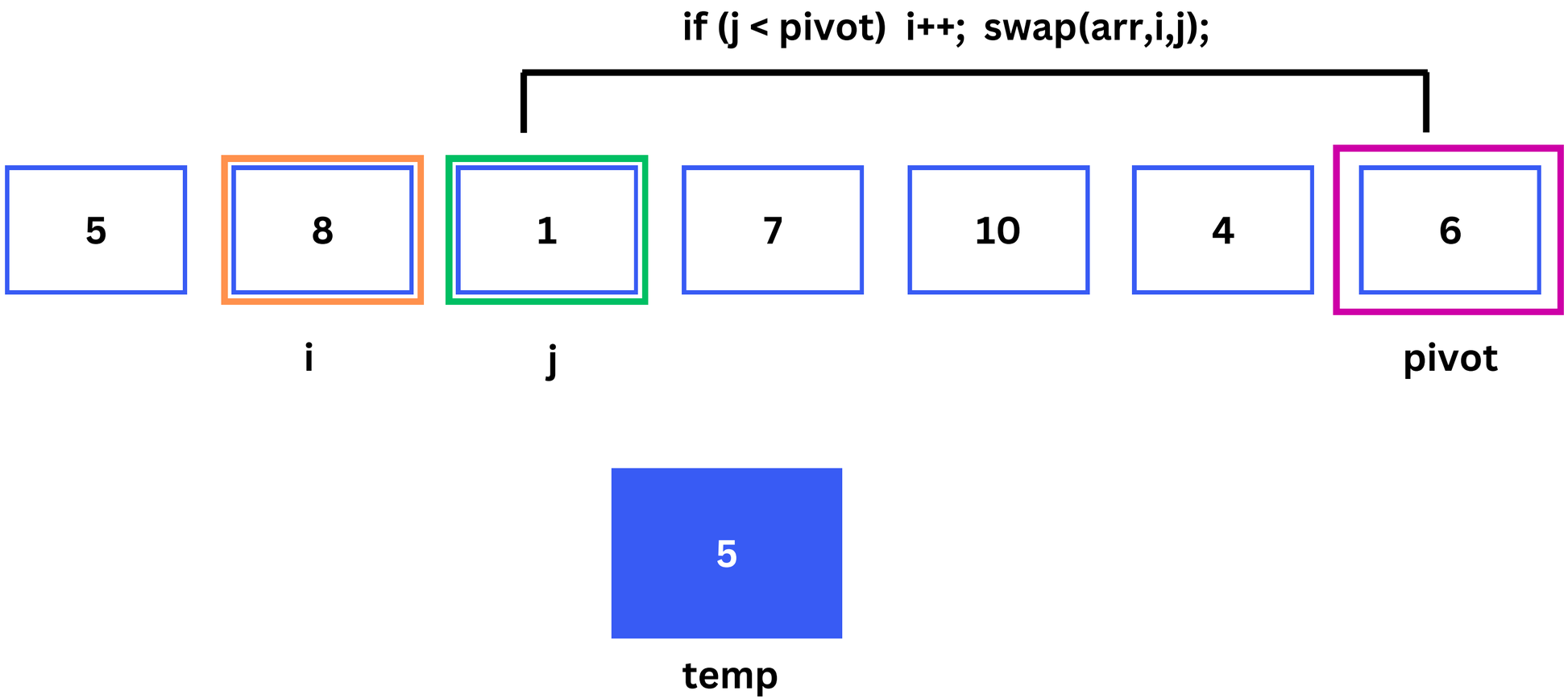
Quick Sort



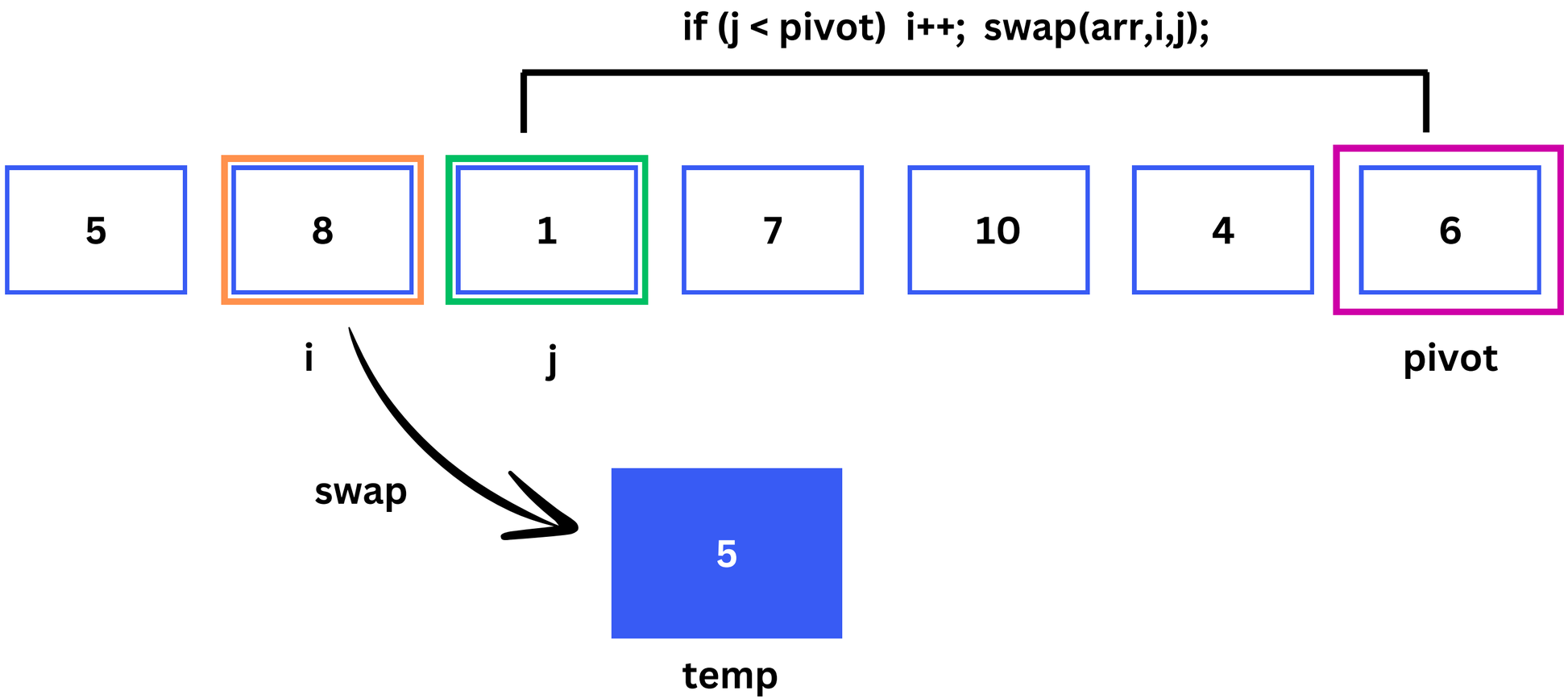
Quick Sort



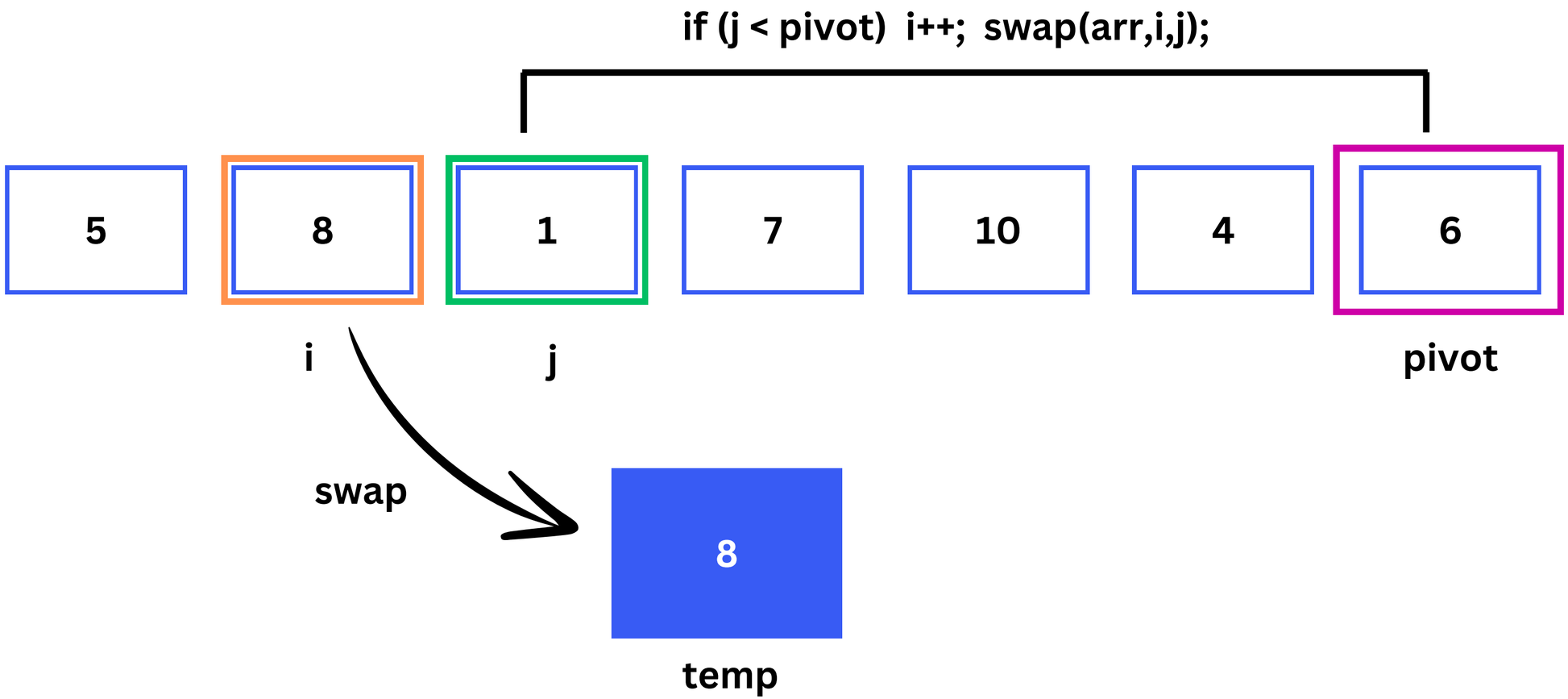
Quick Sort



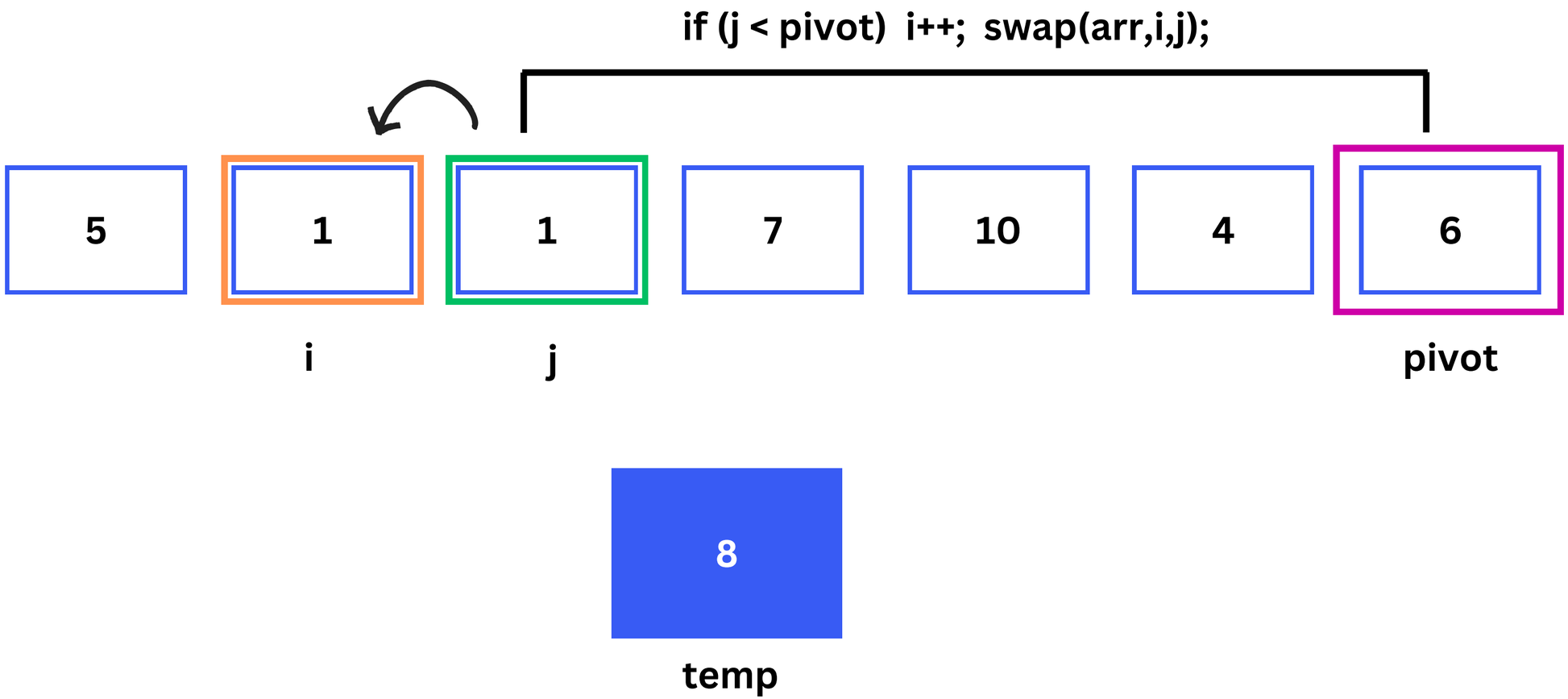
Quick Sort



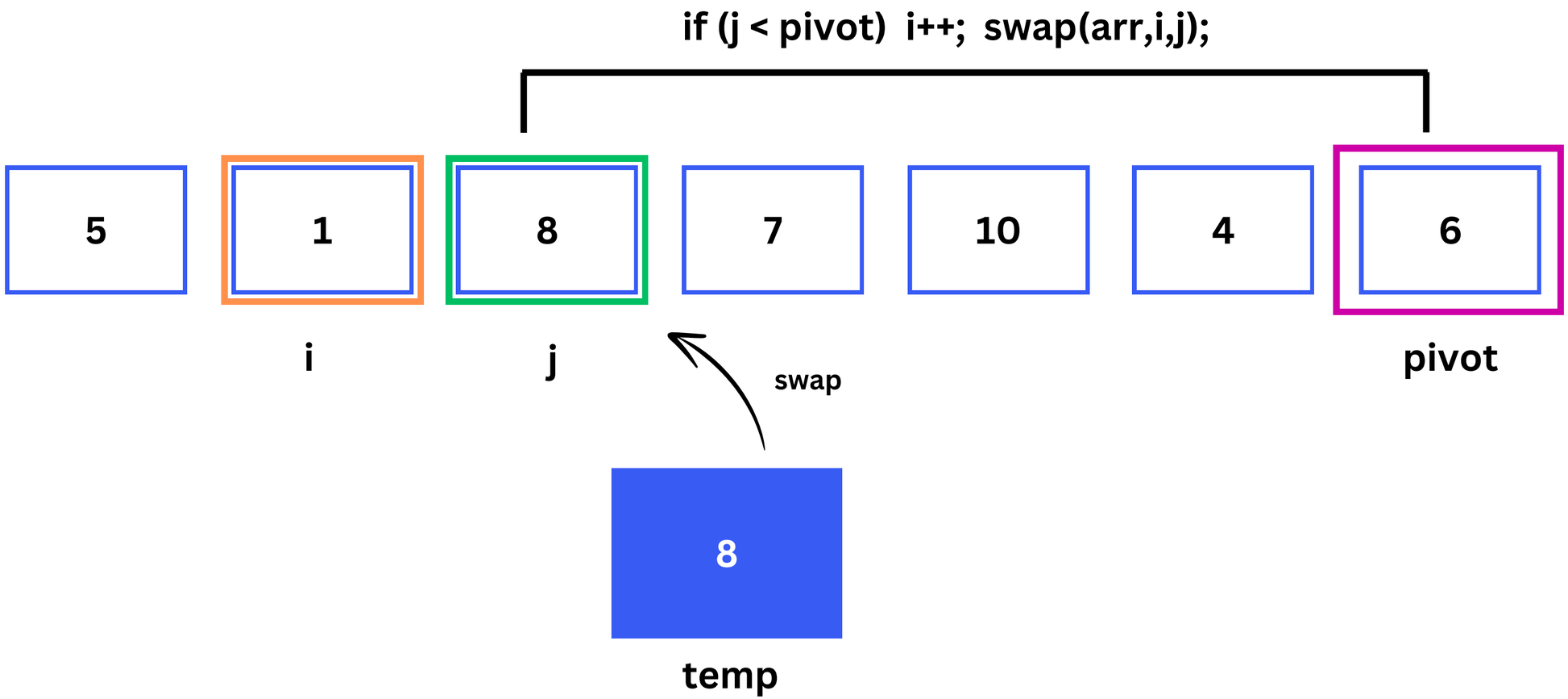
Quick Sort



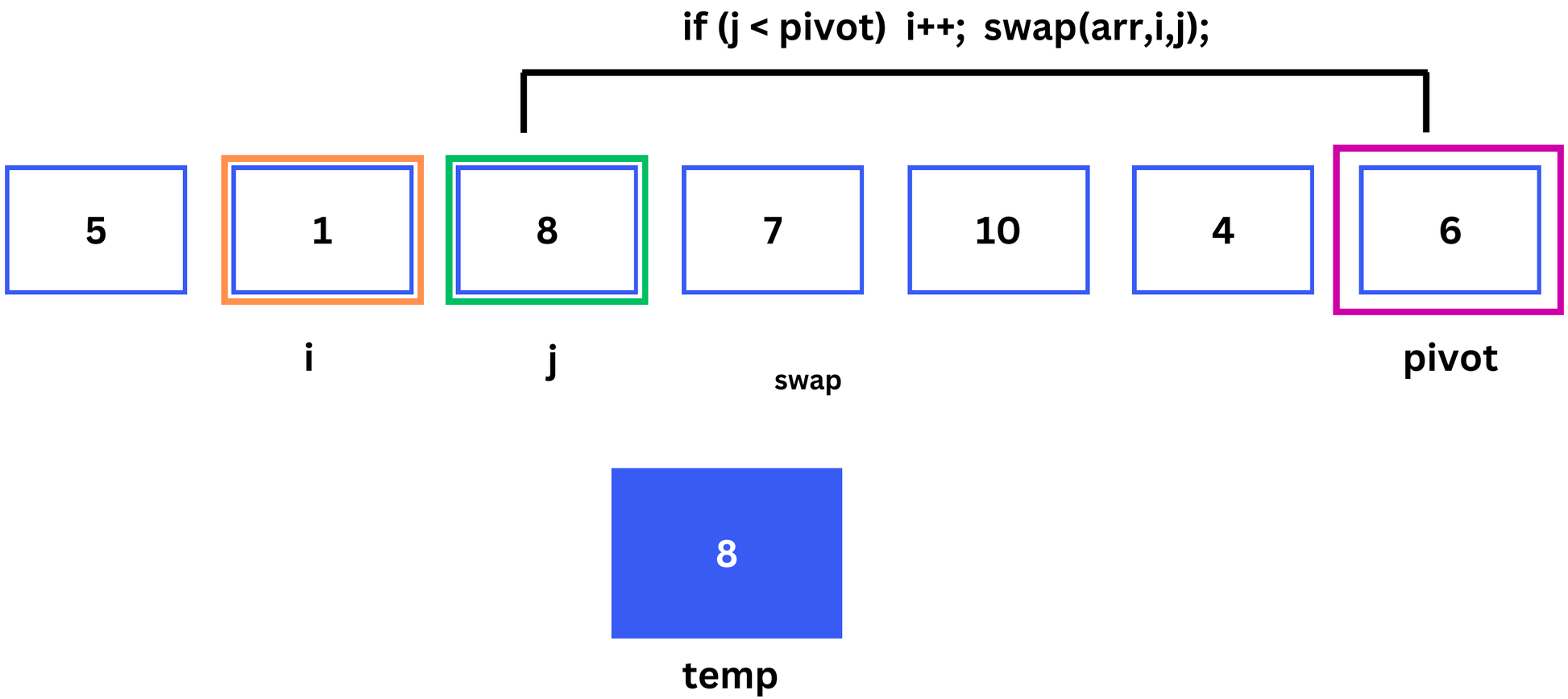
Quick Sort



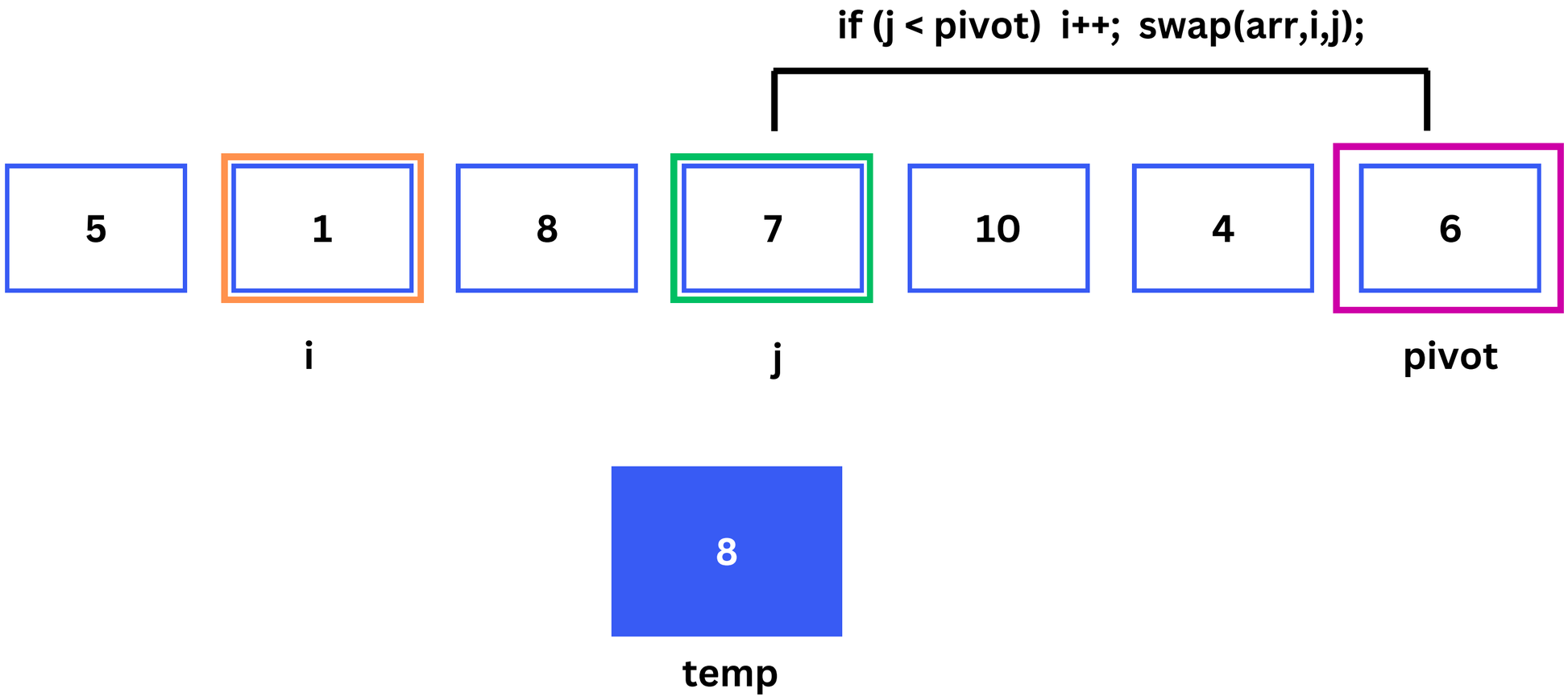
Quick Sort



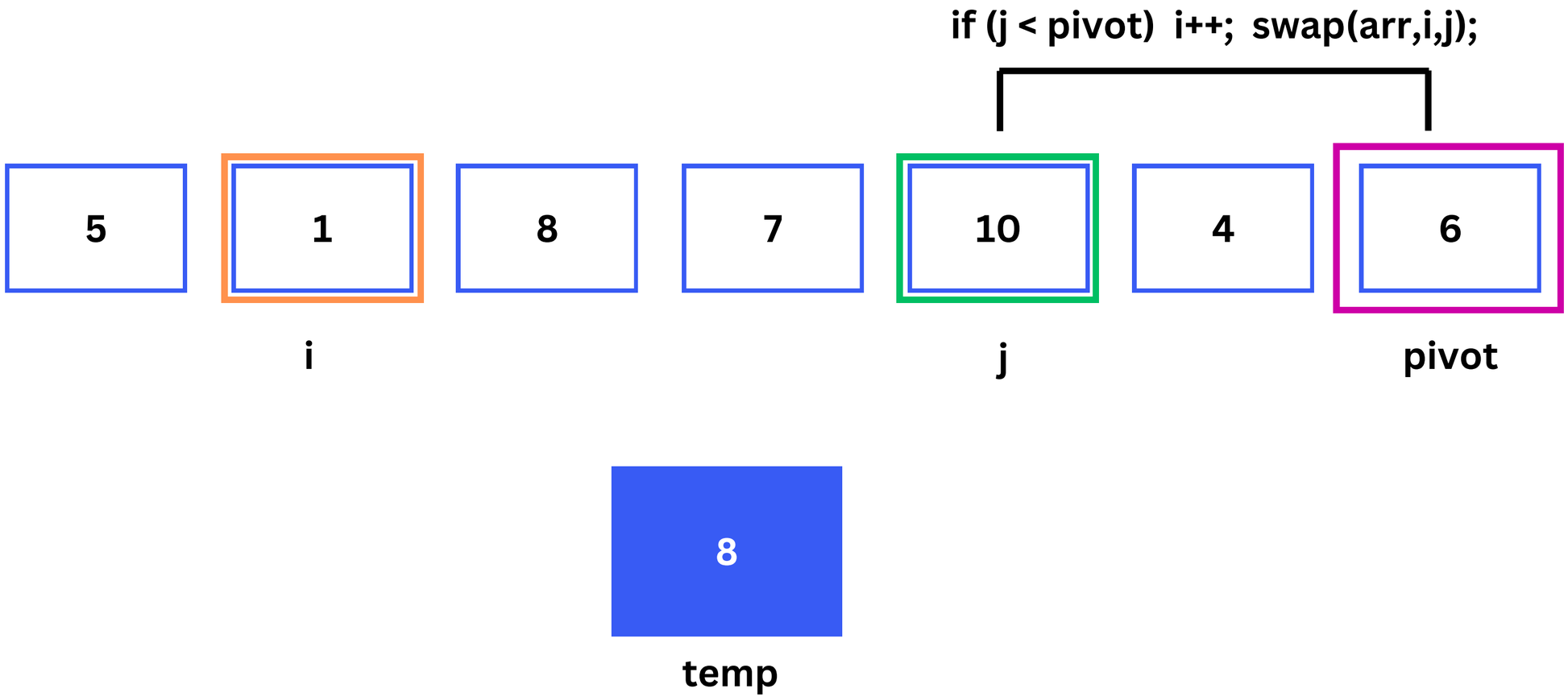
Quick Sort



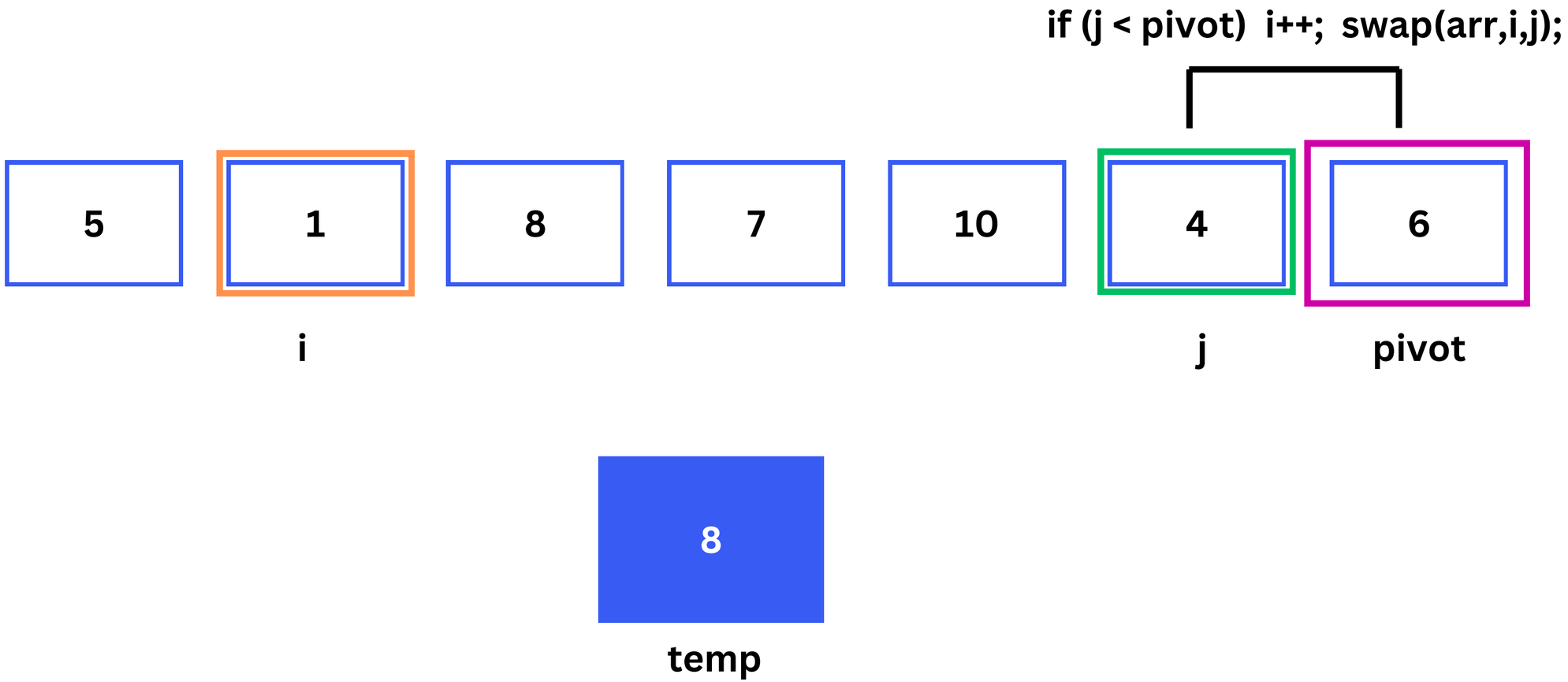
Quick Sort



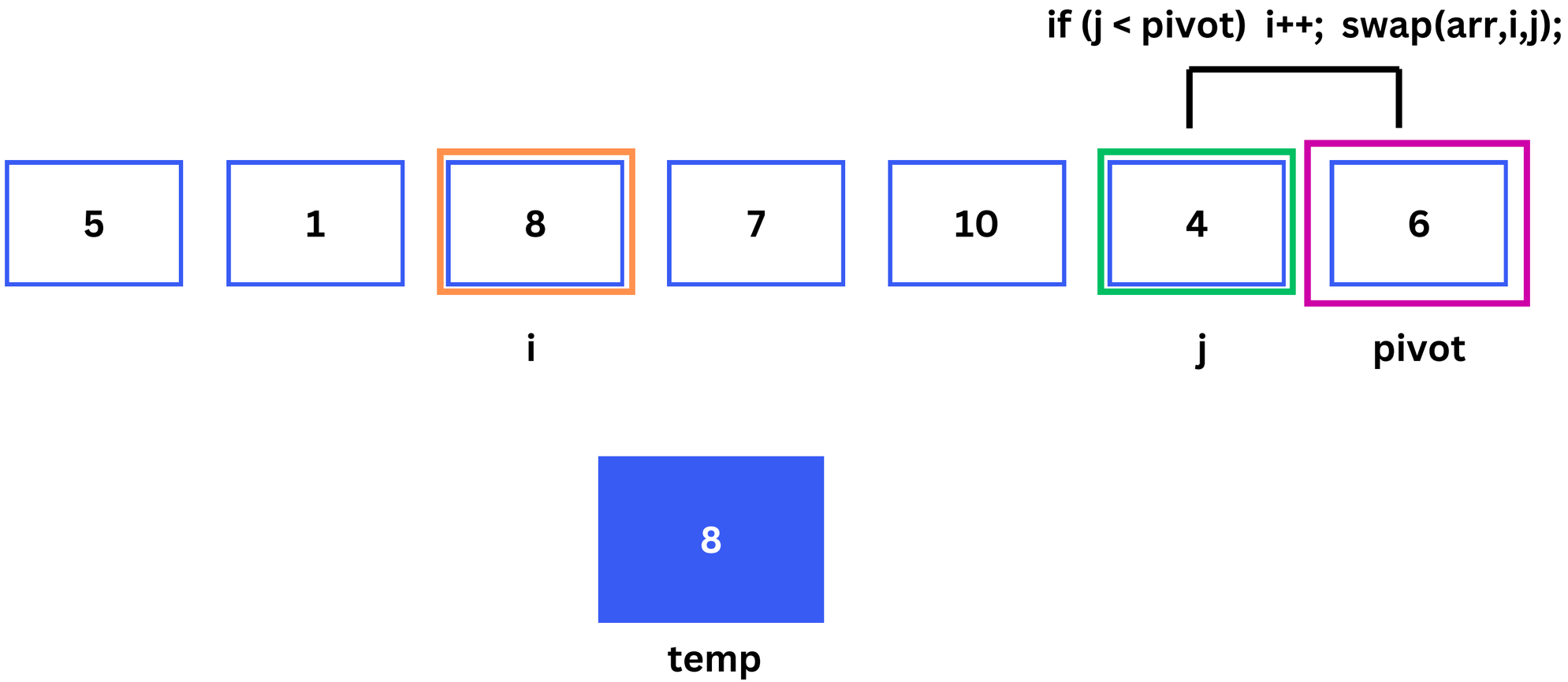
Quick Sort



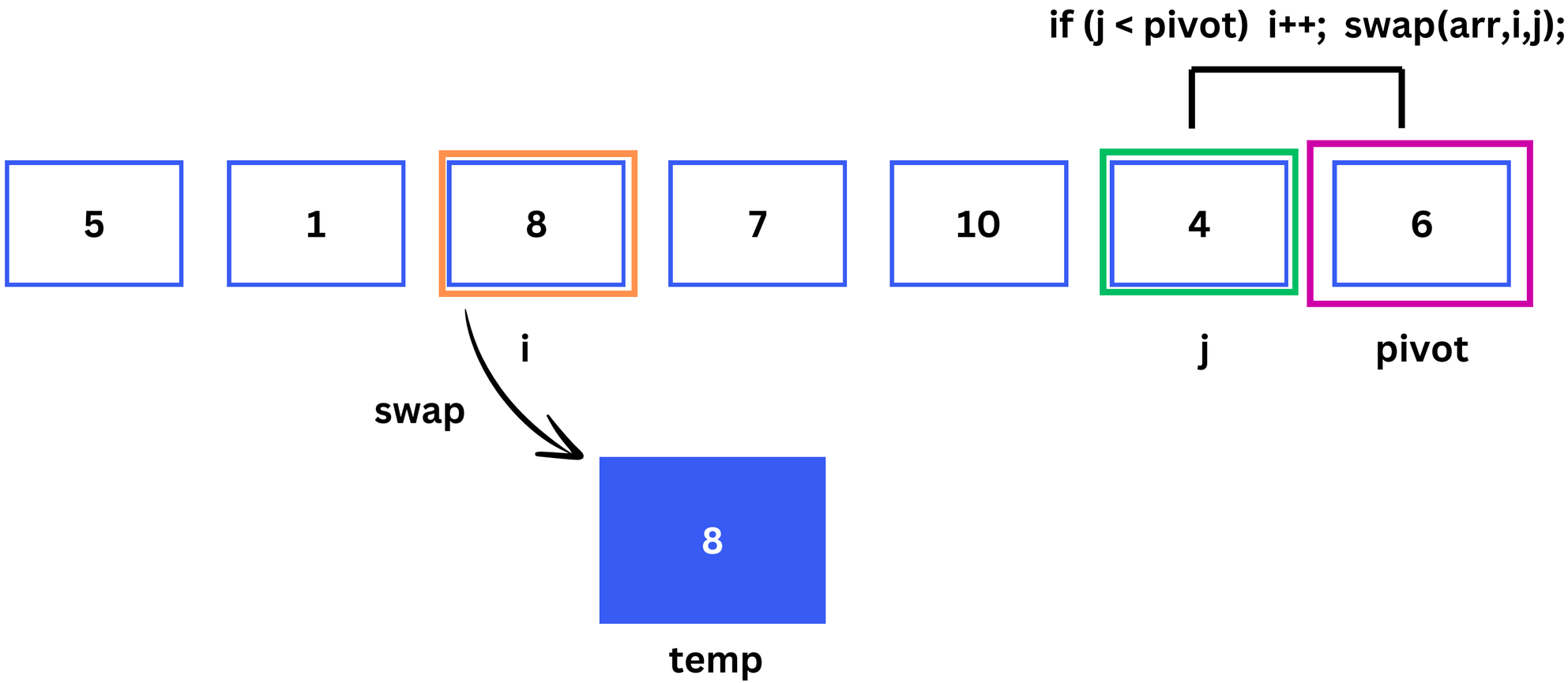
Quick Sort



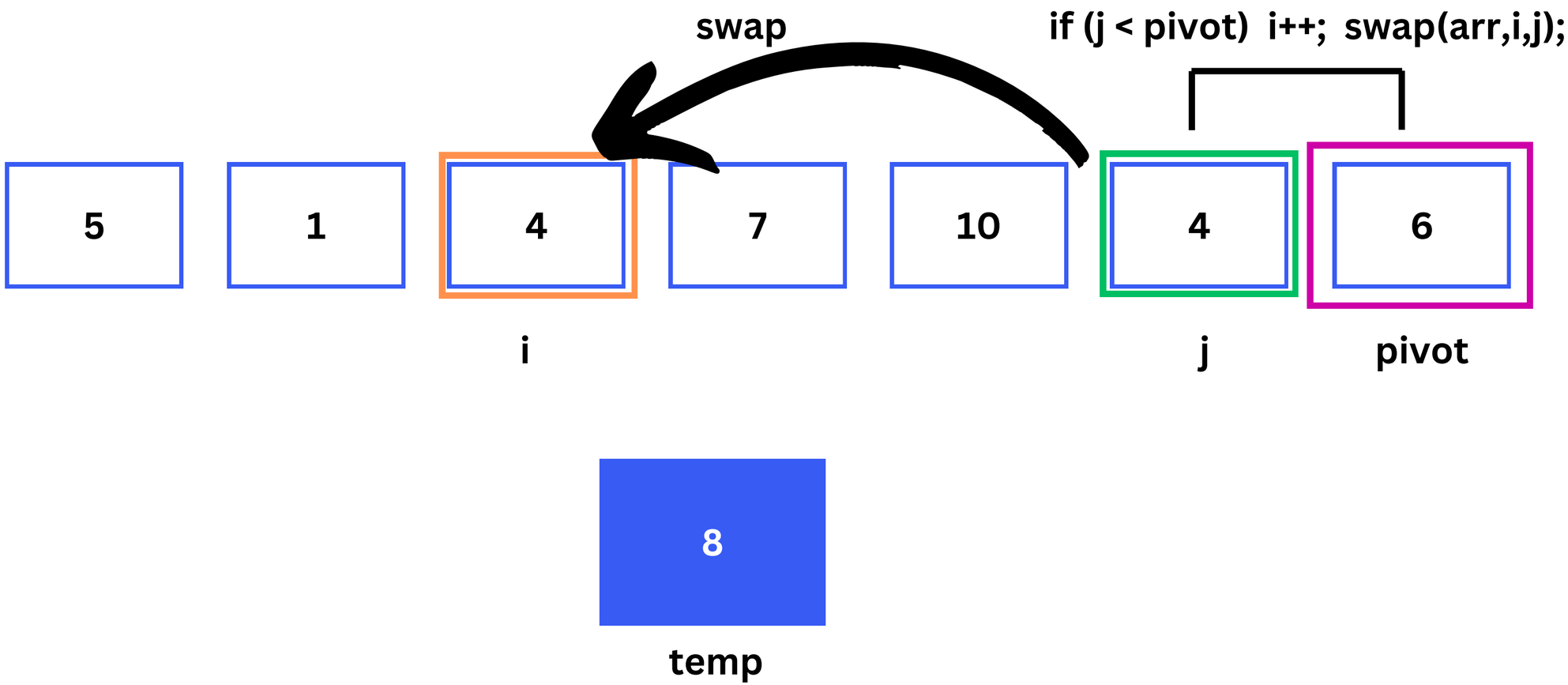
Quick Sort



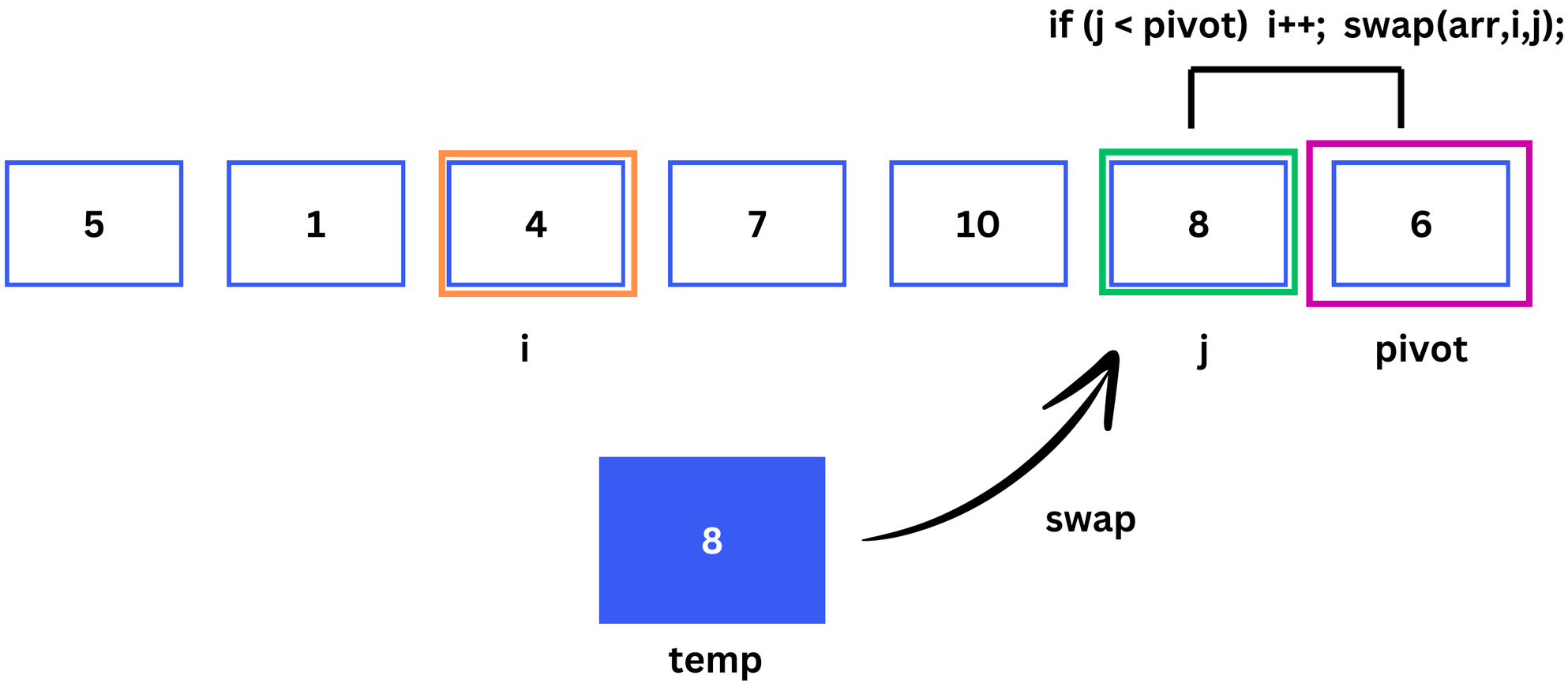
Quick Sort



Quick Sort

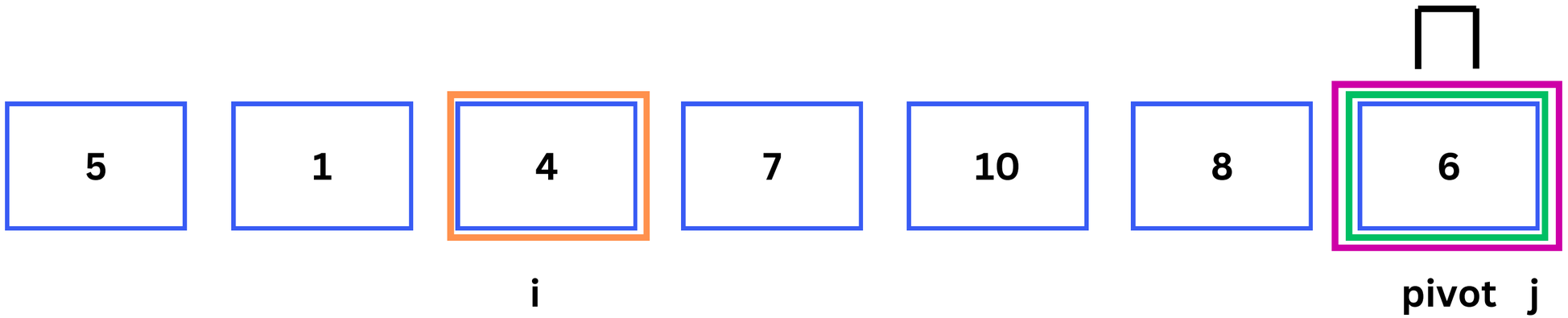


Quick Sort



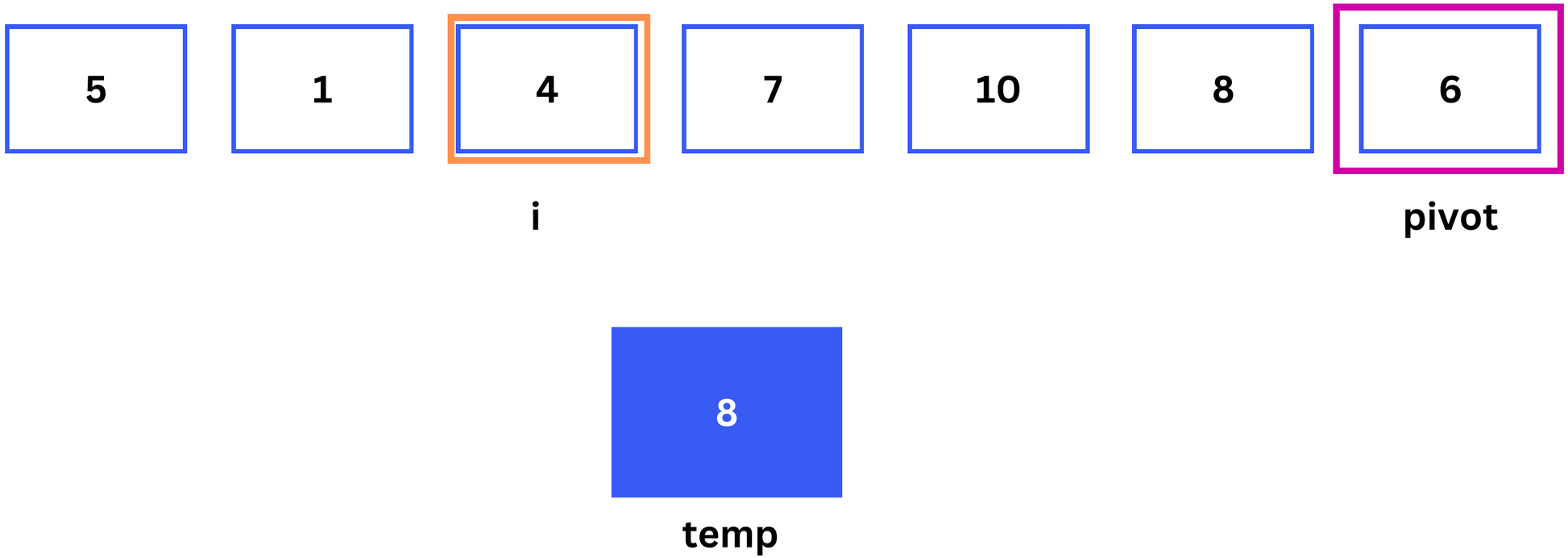
Quick Sort

if (j < pivot) i++; swap(arr,i,j);

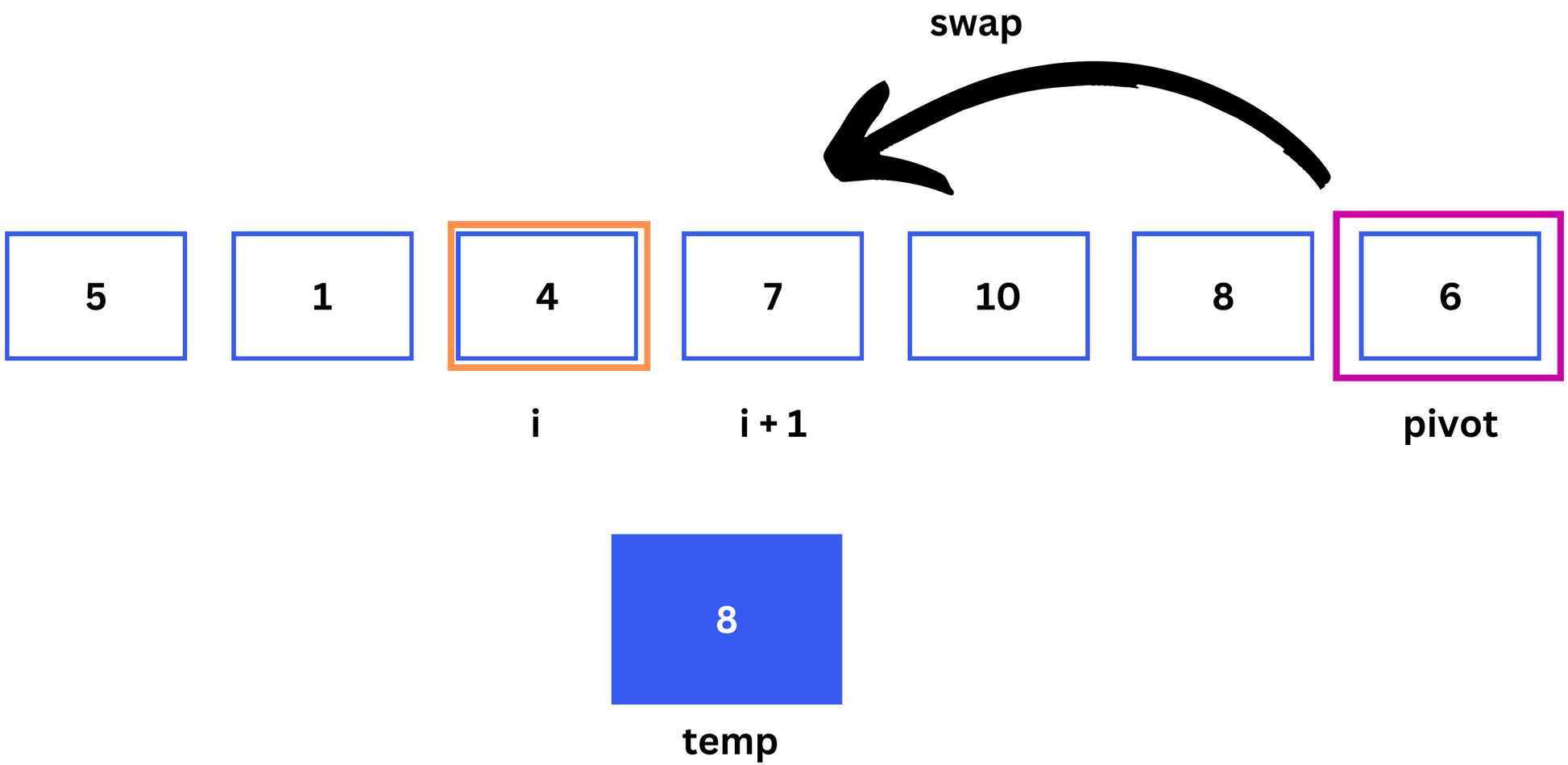


temp

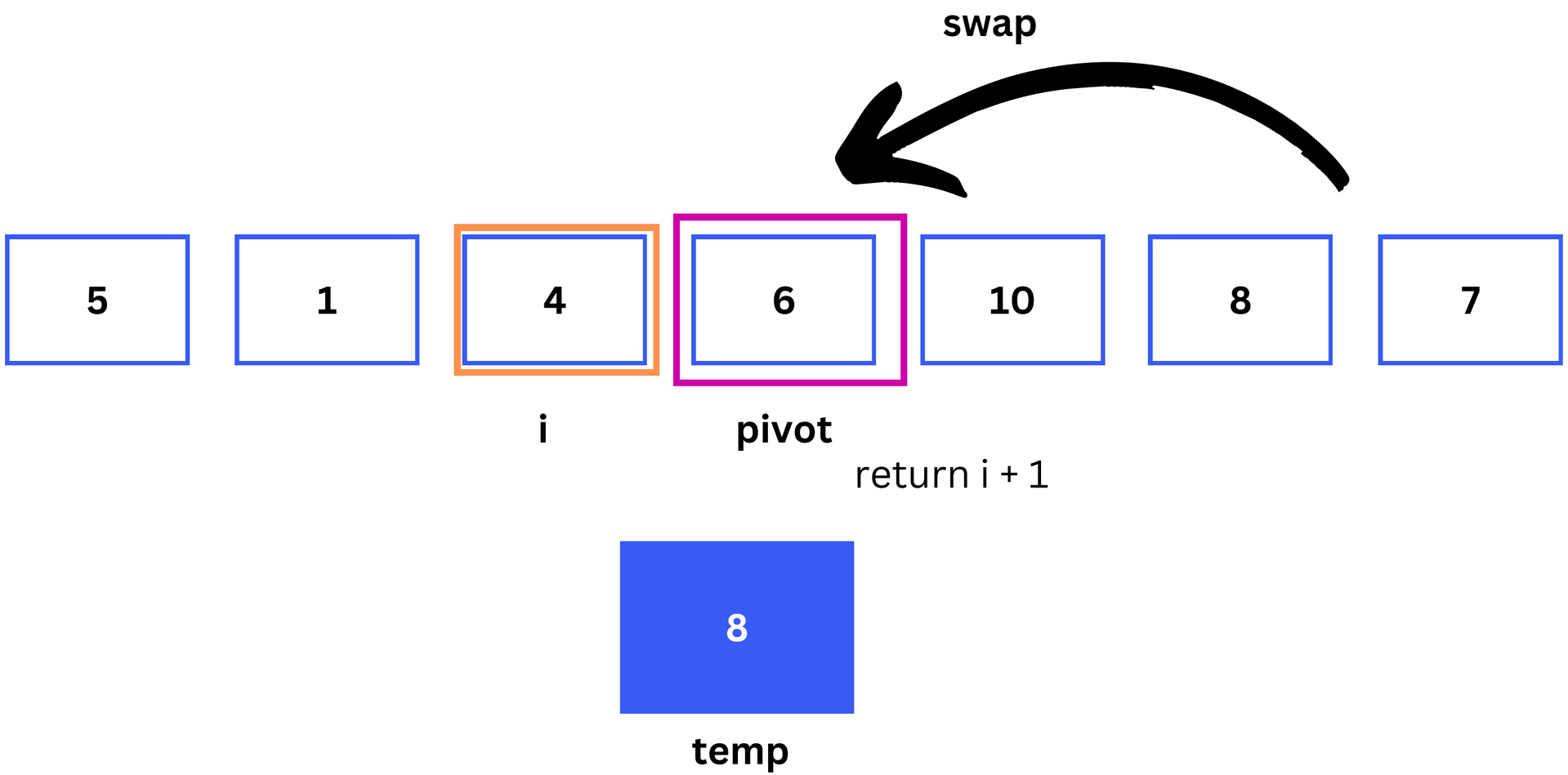
Quick Sort



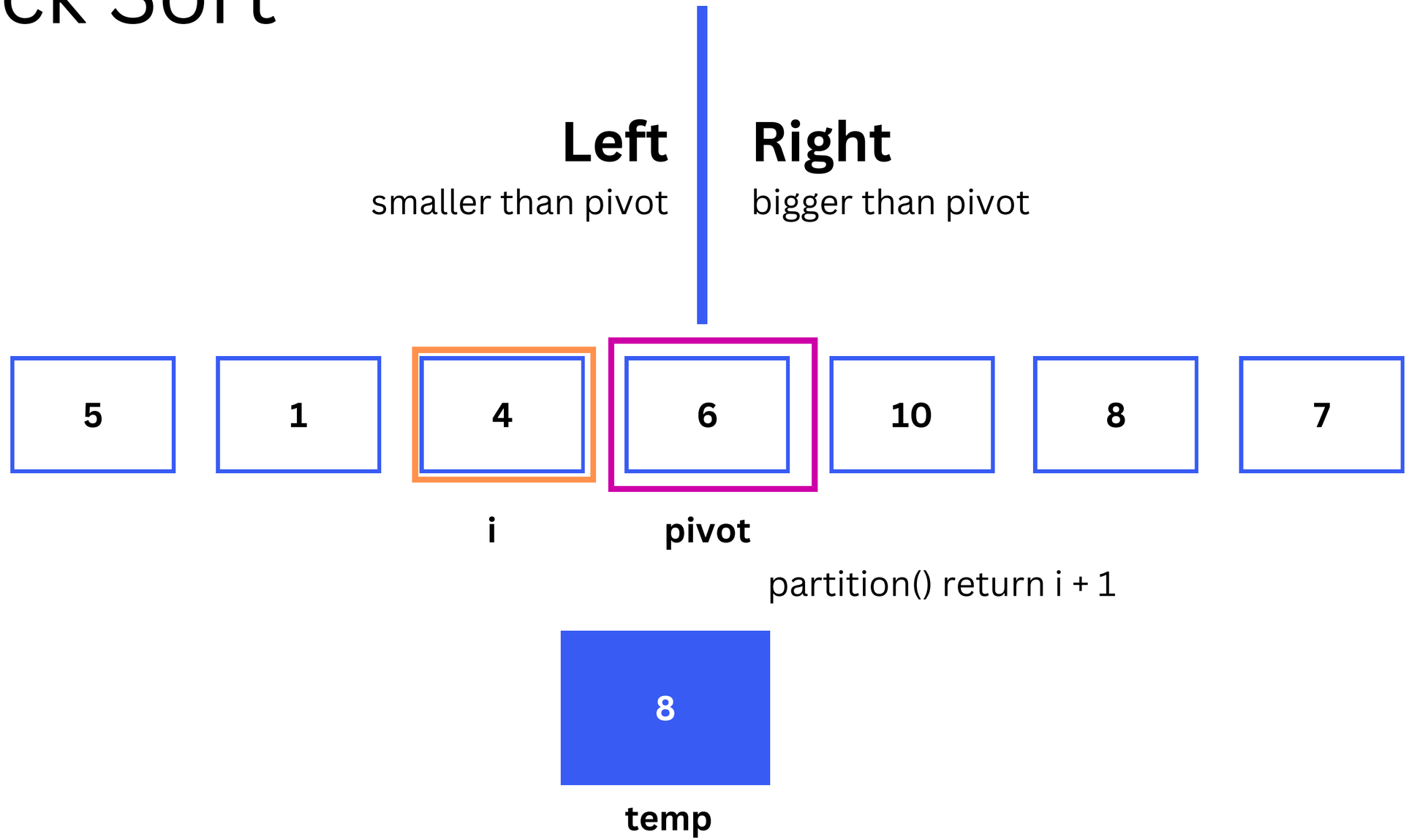
Quick Sort



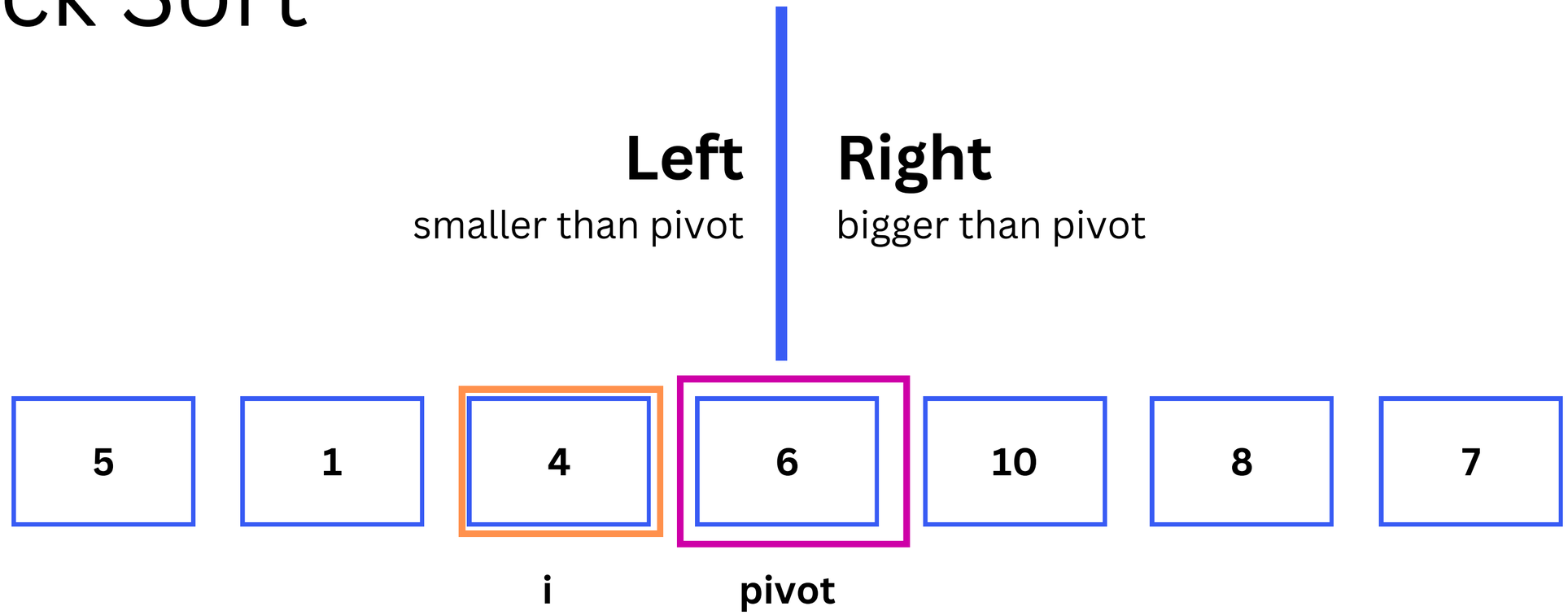
Quick Sort



Quick Sort


Left
smaller than pivot
Right
bigger than pivot
5
1
4
6
10
8
7
i
pivot
partition() return i + 1
8
temp

Quick Sort



partition() return i + 1

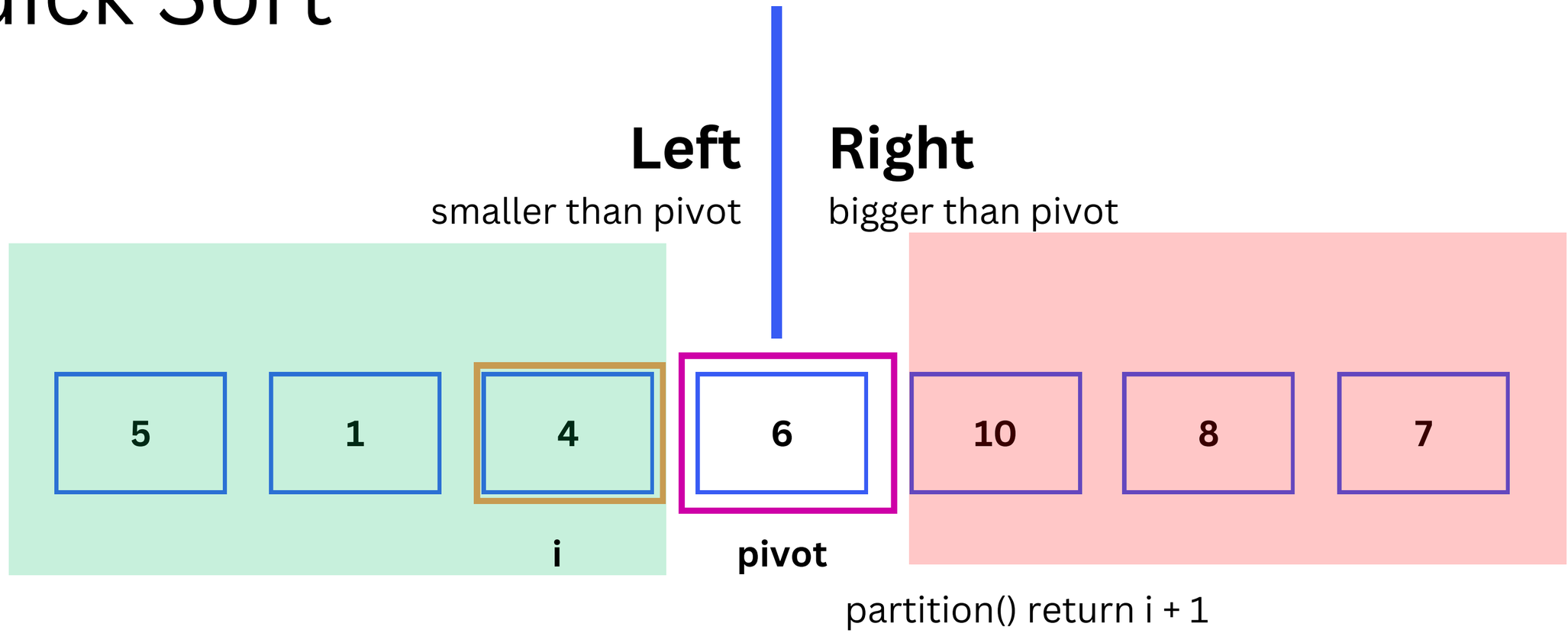
pi = 3 (ตำแหน่งที่ไม่ใช้)

quickSort(arr, low, pi - 1); **0 --> pi -1 = 2**

quickSort(arr, pi + 1, high); **pi + 1 --> arr.length -1 = 6**

เรียก recursive ทำฝั่งซ้าย และฝั่งขวา

Quick Sort



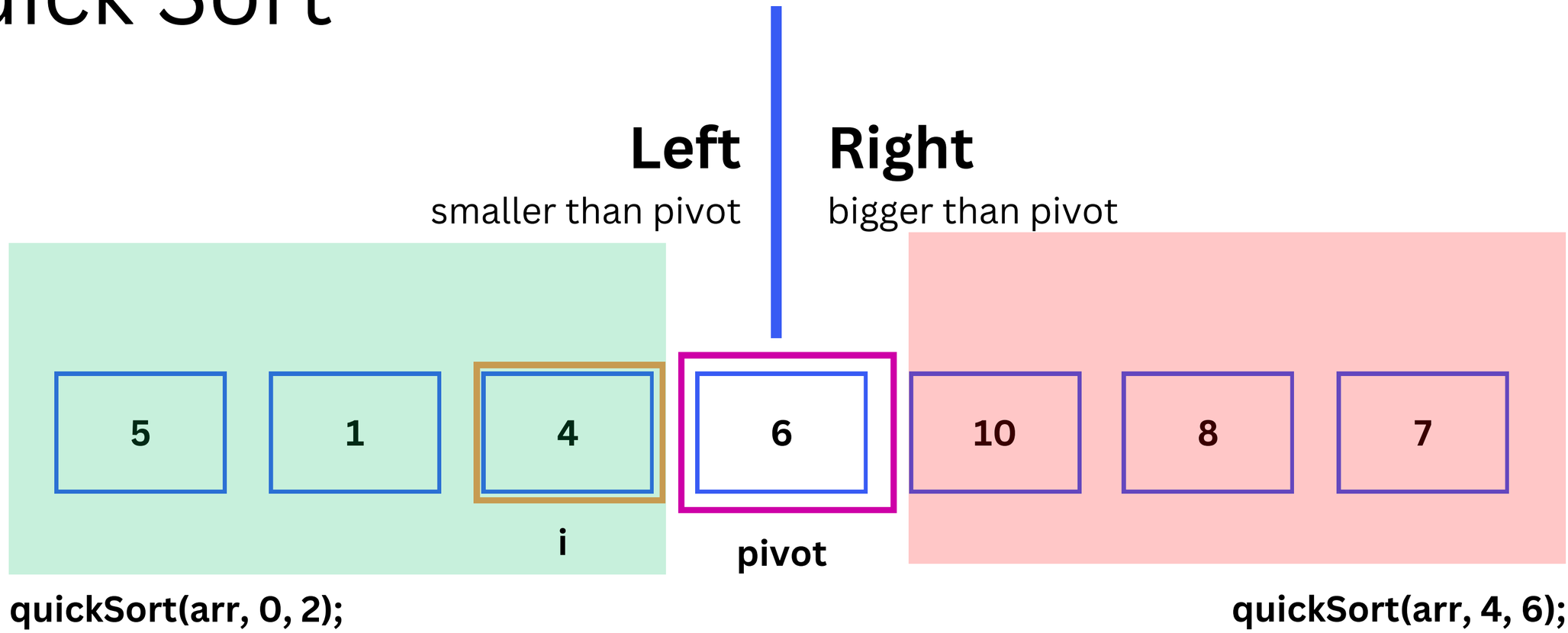
pi = 3 (ตำแหน่งที่ไม่ใช้)

quickSort(arr, low, pi - 1); **0 --> pi -1 = 2**

quickSort(arr, pi + 1, high); **pi + 1 --> arr.length -1 = 6**

เรียก recursive ทำผั่งซ้าย และฝั่งขวา

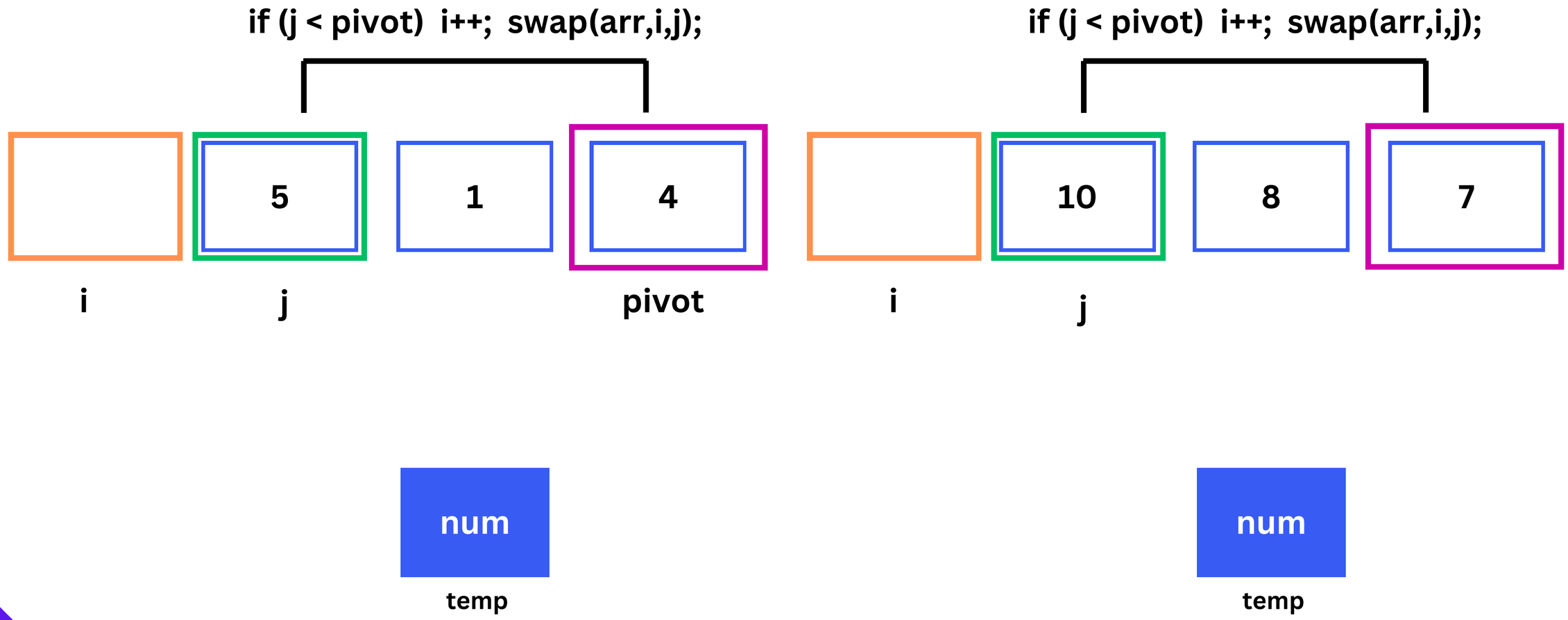
Quick Sort


Left
smaller than pivot
Right
bigger than pivot
5
1
4
6
10
8
7
i
pivot
quickSort(arr, 0, 2);
quickSort(arr, 4, 6);

Quick Sort

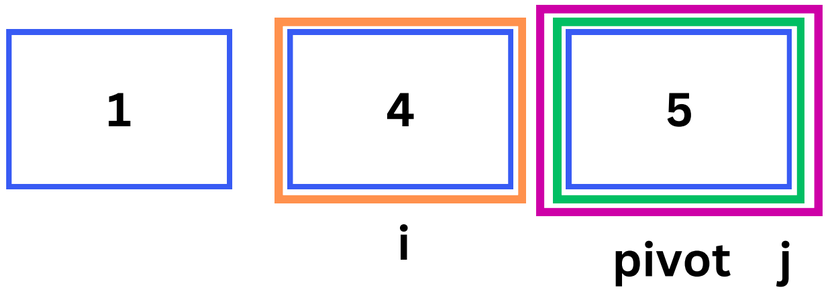
quickSort(arr, 0, 2);

quickSort(arr, 4, 6);



Quick Sort

quickSort(arr, 0, 2);

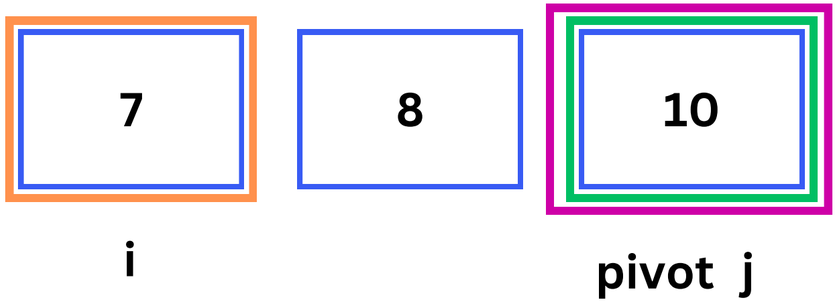


pi = 2 (ตำแหน่งที่ไม่ใช้)

quickSort(arr, low, pi - 1); **0 --> 1** ทำ

quickSort(arr, pi + 1, high); **3 --> 2** ไม่ทำ เพราะ 3 >2

quickSort(arr, 4, 6);



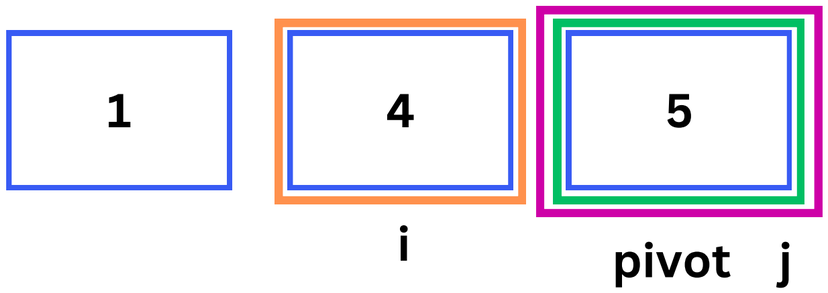
pi = 2 (ตำแหน่งที่ไม่ใช้)

quickSort(arr, low, pi - 1); **0 --> 1** ทำ

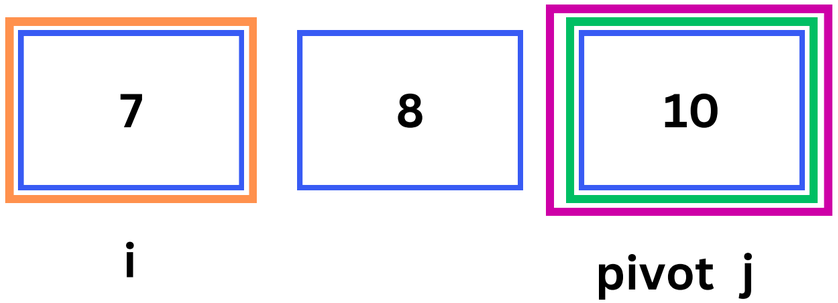
quickSort(arr, pi + 1, high); **3 --> 2** ไม่ทำ เพราะ 3 >2

Quick Sort

quickSort(arr, 0, 2);



quickSort(arr, 4, 6);



Quick Sort

quickSort(arr, 0, 2);

quickSort(arr, 4, 6);

Quick Sort

quickSort(arr, 0, 2);

quickSort(arr, 4, 6);