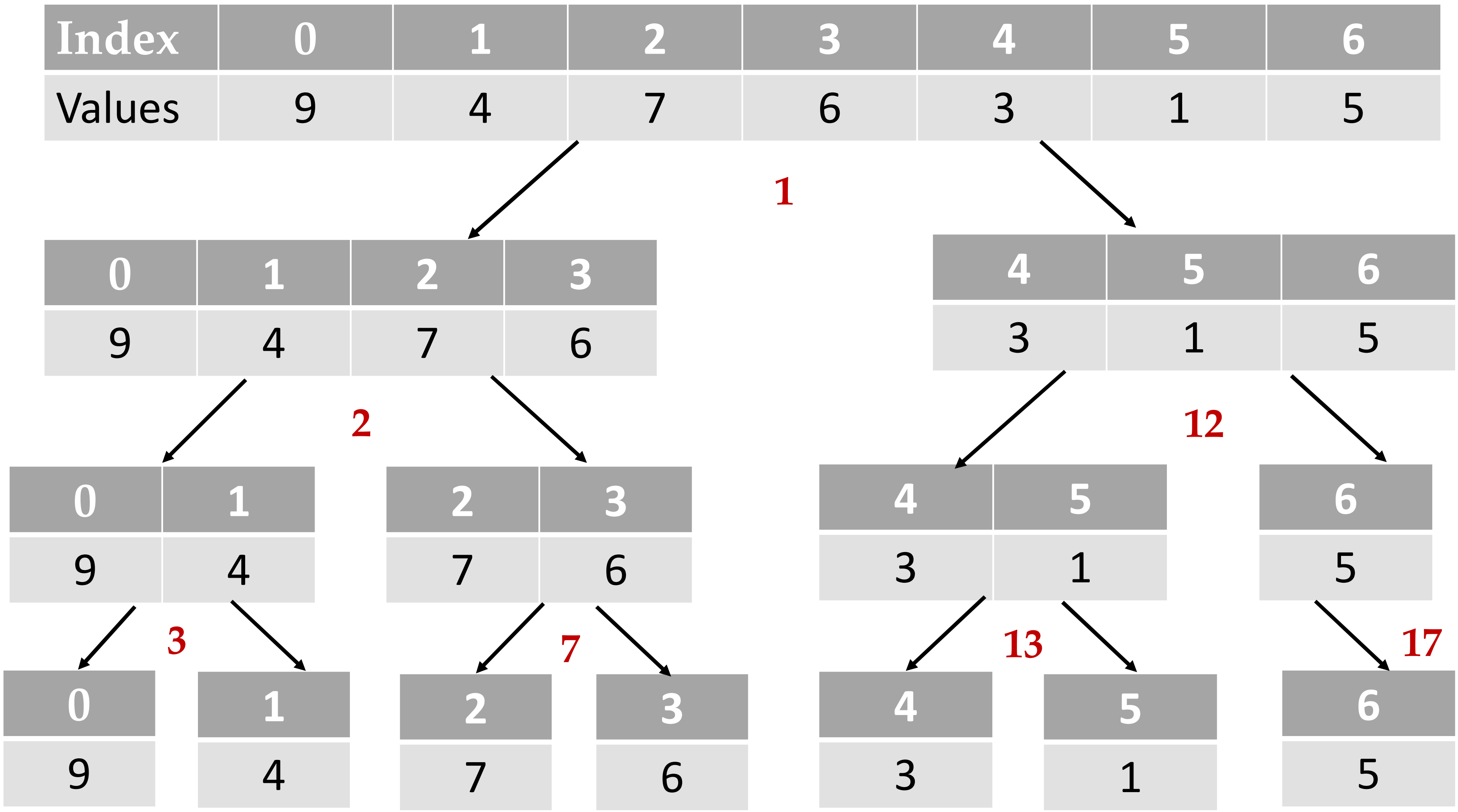
Sorting

Merge Sort

- Merge sort is based divide and conquer strategy.
- Can be implemented as external sorting, when the dataset is so big that it is impossible to load the whole dataset into memory, here sorting is done in chunks.
- Merging is the process of combining two sorted files to make one bigger sorted file.
- Selection is the process of dividing a file into two parts: k smallest elements and n – k largest elements.
- Selection and merging are opposite operations
  - selection splits a list into two lists
  - merging joins two files to make one file

Algorithm – Merge Sort

1. **Divide:** Divide the list or array recursively into two halves until it can no more be divided.
2. **Conquer:** Each subarray is sorted individually using the merge sort algorithm.
3. **Merge:** The sorted subarrays are merged back together in sorted order. The process continues until all elements from both subarrays have been merged.



Index	0	1	2	3	4	5	6
Values	9	4	7	6	3	1	5

0	1	2	3
9	4	7	6

4	5	6
3	1	5

0	1
9	4

2	3
7	6

4	5
3	1

6
5

0
9

1
4

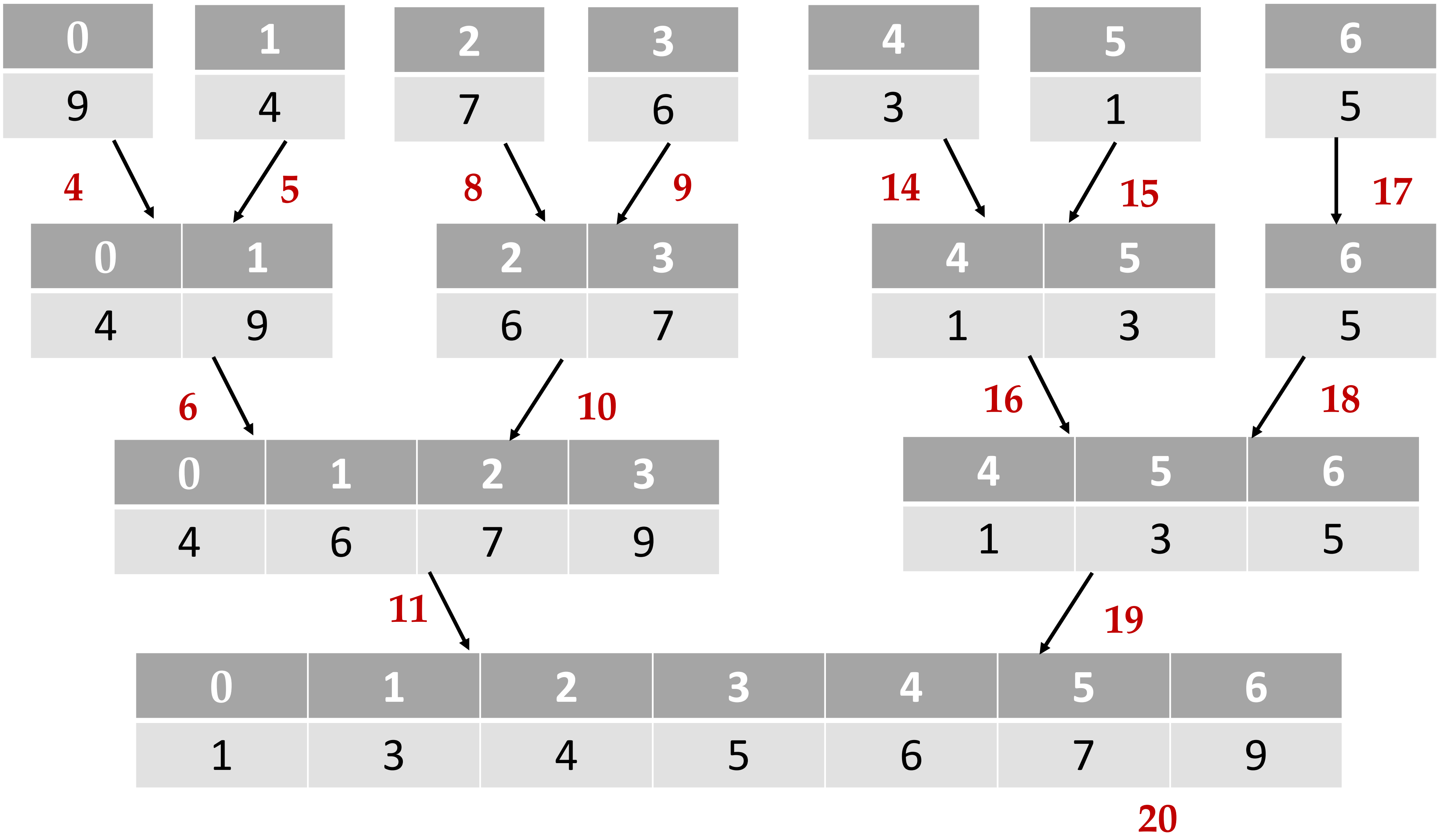
2
7

3
6

4
3

5
1

6
5



0
9

1
4

2
7

3
6

4
3

5
1

6
5

0	1
4	9

2	3
6	7

4	5
1	3

6
5

0	1	2	3
4	6	7	9

4	5	6
1	3	5

0	1	2	3	4	5	6
1	3	4	5	6	7	9

Performance – Merge Sort

- Worst case complexity : O(nlogn)
- Best case complexity : O(nlogn)
- Average case complexity : O(nlogn)
- Worst case space complexity: O(n) auxiliary

Quick Sort

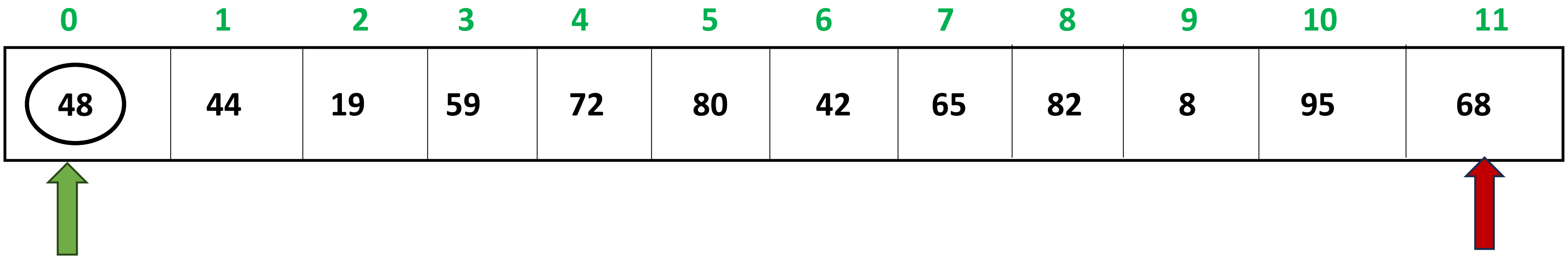
- It is an example of a divide-and-conquer algorithmic technique.
- It is also called partition exchange sort.
- It uses recursive calls for sorting the elements, and it is one of the famous algorithms among comparison-based sorting algorithms.
- Divide: The array A[low ...high] is partitioned into two non-empty sub arrays A[low ...q] and A[q + 1... high], such that each element of A[low ... high] is less than or equal to each element of A[q + 1... high]. The index q is computed as part of this partitioning procedure.
- Conquer: The two sub arrays A[low ...q] and A[q + 1 ...high] are sorted by recursive calls to Quick sort.

Algorithm – Quick Sort

The recursive algorithm consists of four steps:

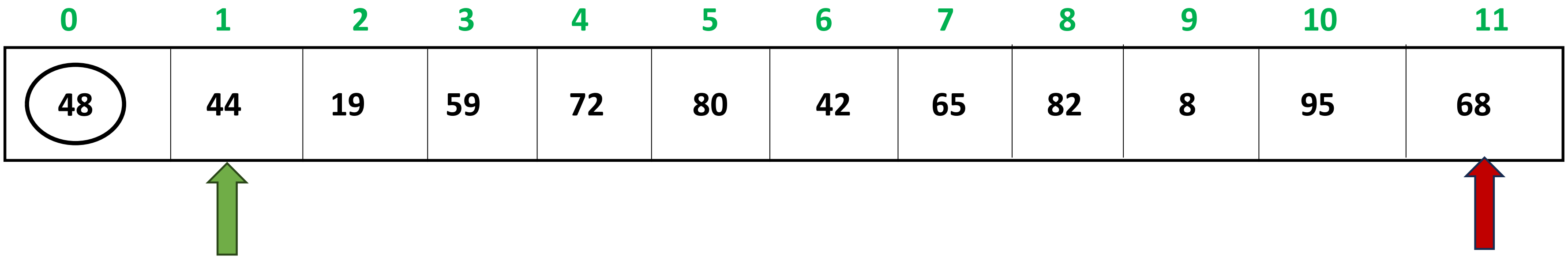
- If there are one or no elements in the array to be sorted, return.
- Pick an element in the array to serve as the “pivot” point. (Usually the left-most element in the array is used.)
- Split the array into two parts – one with elements larger than the pivot and the other with elements smaller than the pivot.
- Recursively repeat the algorithm for both halves of the original array.

Tip: In this algorithm, we place the pivot element at right place.



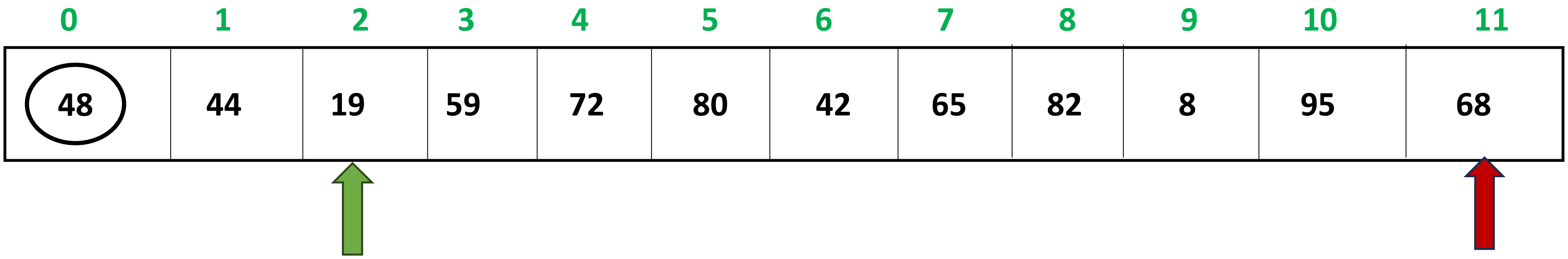
0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	59	72	80	42	65	82	8	95	68

```
public static int partition(int arr[], int low, int high)
{                                                     0        11
   int left, right, pivot, t;

   pivot=arr[low];       // pivot = 48
   left=low;             // left = 0
   right =high;          // right = 11

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
         left++;

     while(arr[right] > pivot)
         right--;
```



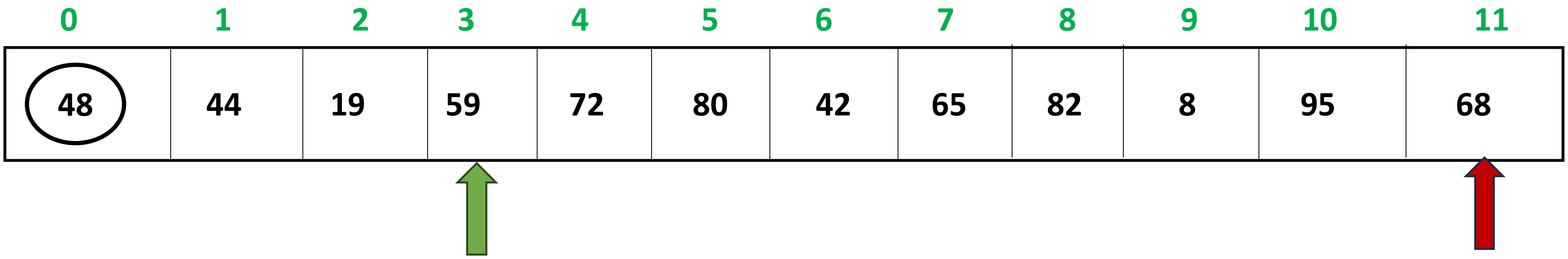
0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	59	72	80	42	65	82	8	95	68

```
public static int partition(int arr[], int low, int high)
{                                               0       11
   int left, right, pivot, t;

   pivot=arr[low];       // pivot = 48
   left=low;             // left = 0
   right =high;          // right = 11

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
         left++;

     while(arr[right] > pivot)
         right--;
```



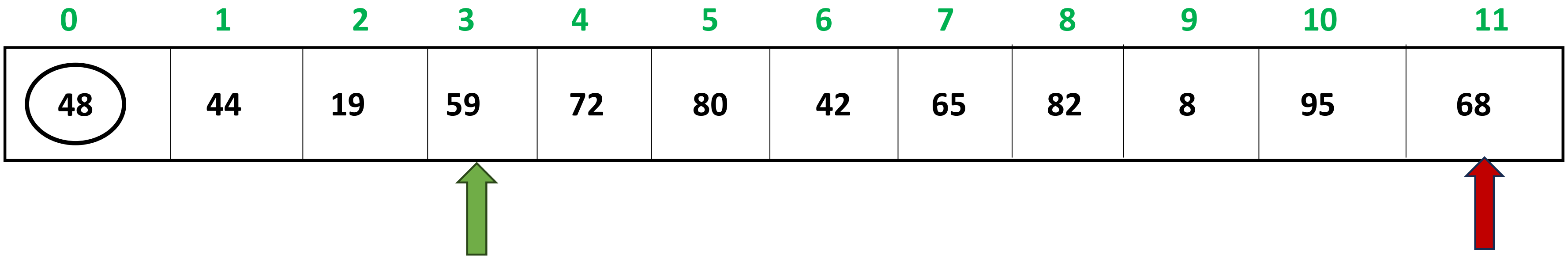
0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	59	72	80	42	65	82	8	95	68

```
public static int partition(int arr[], int low, int high)
{                                                    0        11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 48
  left=low;             // left = 0
  right =high;          // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```



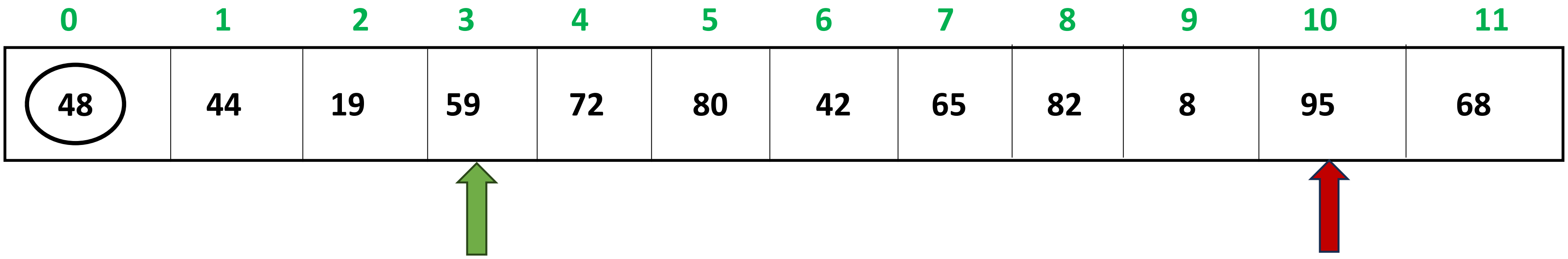
0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	59	72	80	42	65	82	8	95	68

```
public static int partition(int arr[], int low, int high)
{                                                     0        11
   int left, right, pivot, t;

   pivot=arr[low];       // pivot = 48
   left=low;             // left = 0
   right =high;          // right = 11

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
         left++;

     while(arr[right] > pivot)
         right--;
```



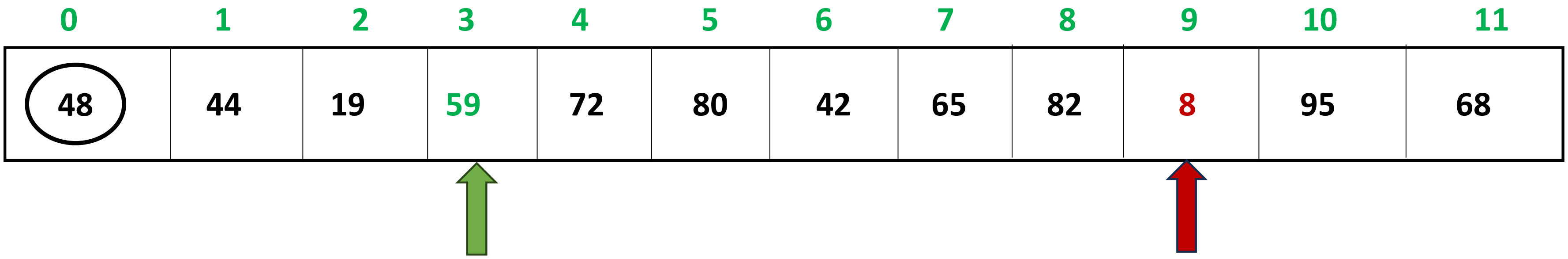
0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	59	72	80	42	65	82	8	95	68

```
public static int partition(int arr[], int low, int high)
{                                                  0        11
  int left, right, pivot, t;

  pivot=arr[low];        // pivot = 48
  left=low;              // left = 0
  right =high;           // right = 11

  while(left <= right)
  {
    while((arr[left]<=pivot) && (left<high))
       left++;

    while(arr[right] > pivot)
       right--;
```



0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	59	72	80	42	65	82	8	95	68

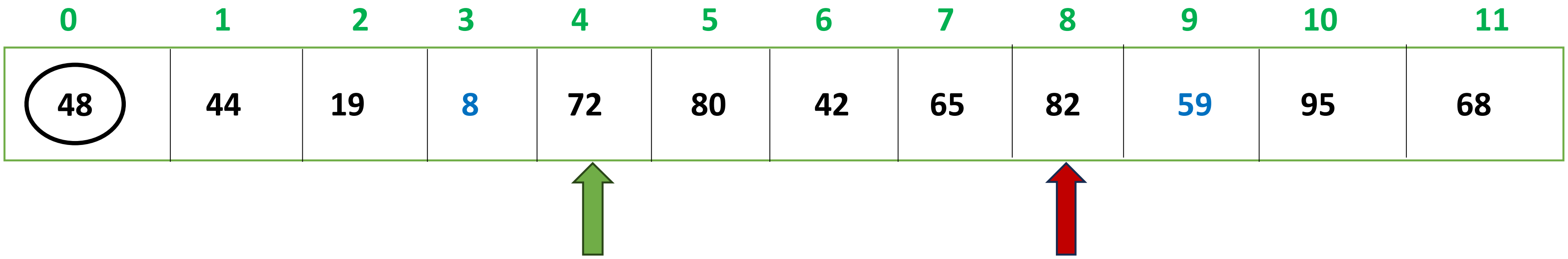
```
public static int partition(int arr[], int low, int high)
{                                                 0       11
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 48
  left=low;            // left = 0
  right =high;         // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```



0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	59	72	80	42	65	82	8	95	68

```
public static int partition(int arr[], int low, int high)
{                                                0        11
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 48
  left=low;            // left = 0
  right =high;         // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

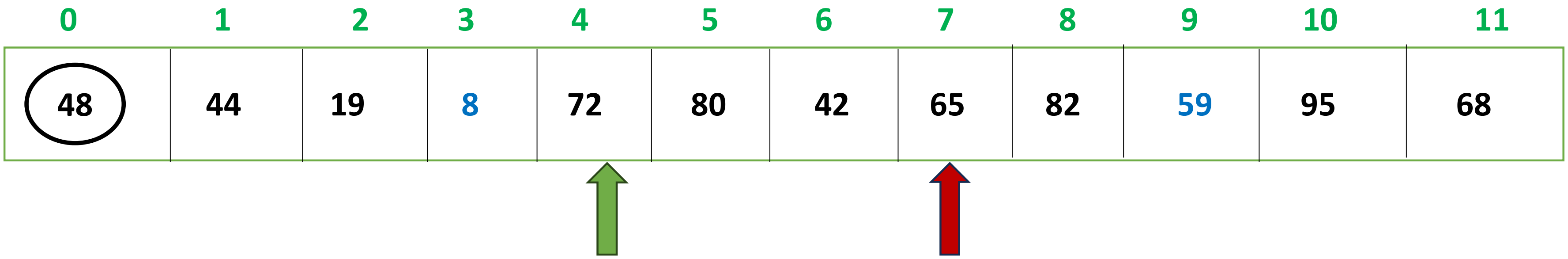
```
if(left < right)
   {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
   }
   else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;       // right =
4
}// end of function
```



0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	8	72	80	42	65	82	59	95	68

```
public static int partition(int arr[], int low, int high)
{                                              0      11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 48
  left=low;             // left = 0
  right =high;          // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

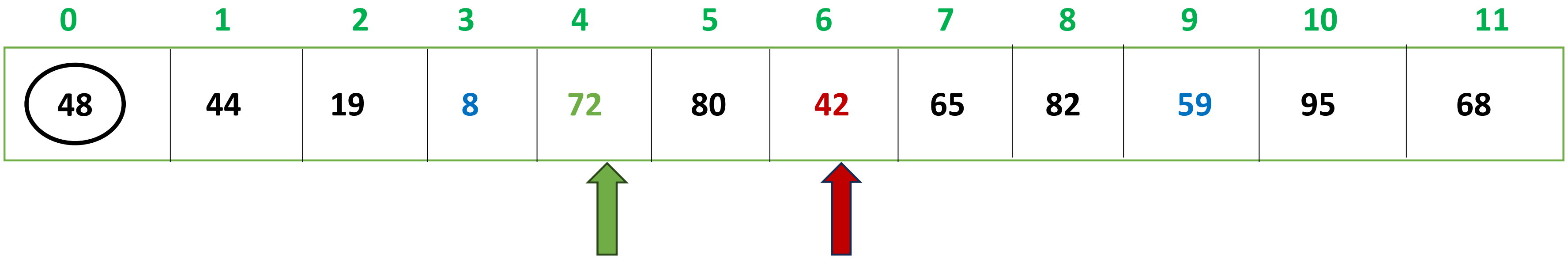
```
if(left < right)
   {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
   }
   else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right =
4
}// end of function
```



0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	8	72	80	42	65	82	59	95	68

```
public static int partition(int arr[], int low, int high)
{                                                   0       11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 48
  left=low;             // left = 0
  right =high;          // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

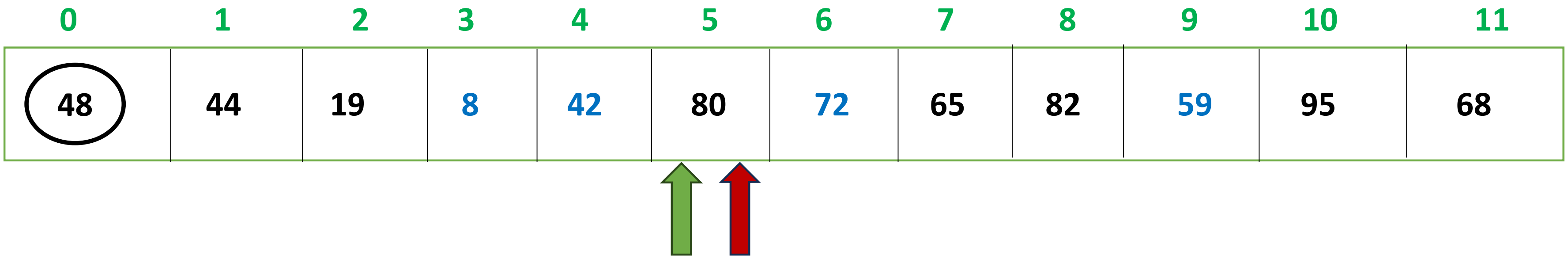
```
if(left < right)
   {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
   }
   else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right =
4
}// end of function
```



0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	8	72	80	42	65	82	59	95	68

```
public static int partition(int arr[], int low, int high)
{                                                   0        11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 48
  left=low;             // left = 0
  right =high;          // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

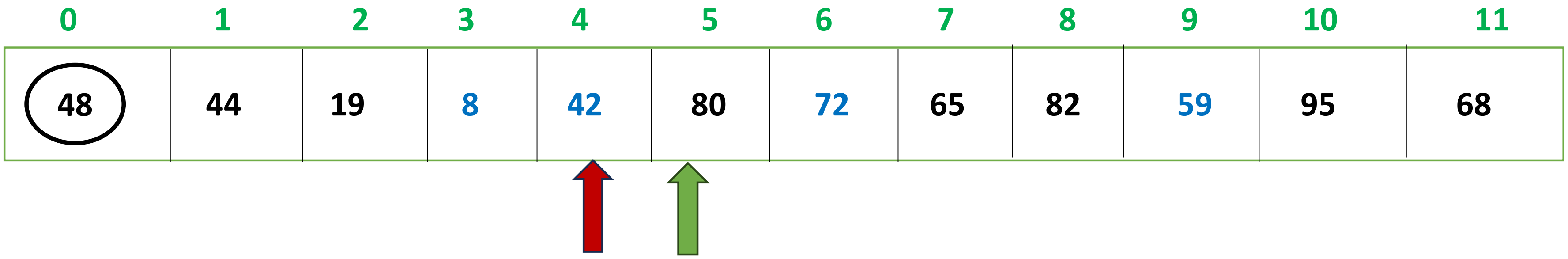
```
if(left < right)
   {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
   }
   else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right =
4
}// end of function
```



0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	8	42	80	72	65	82	59	95	68

```
public static int partition(int arr[], int low, int high)
{                                                0       11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 48
  left=low;             // left = 0
  right =high;          // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

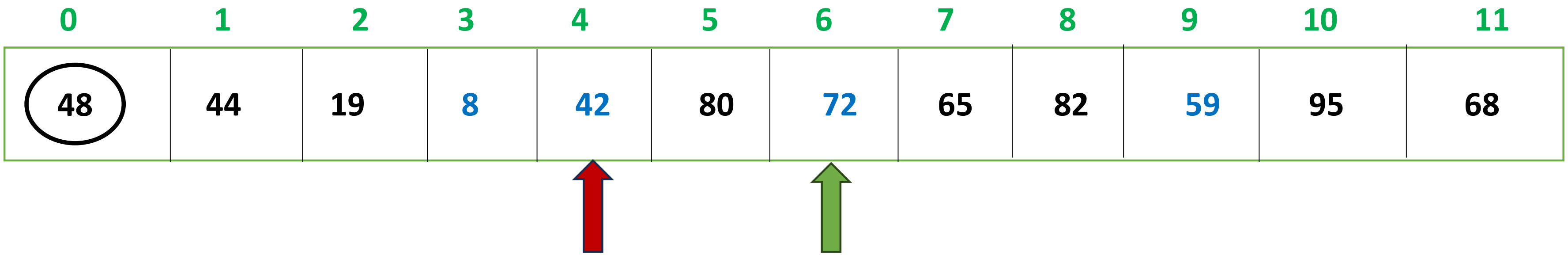
```
if(left < right)
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;     // right =
4
}// end of function
```



0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	8	42	80	72	65	82	59	95	68

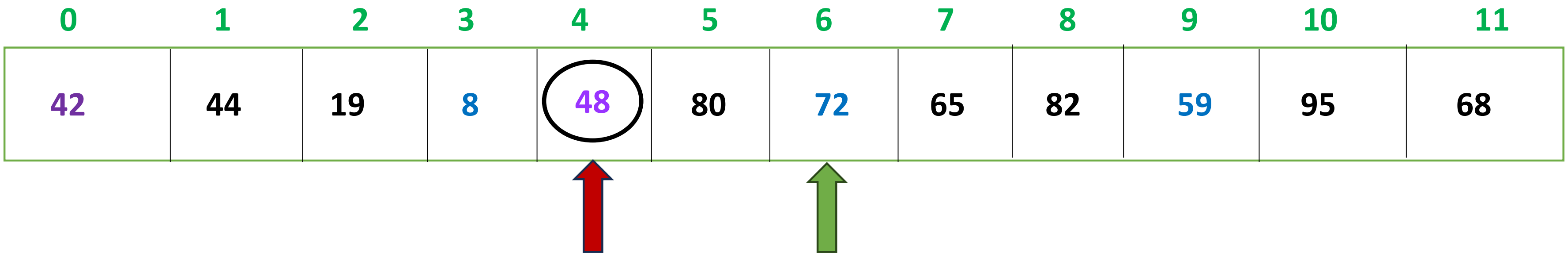
```
public static int partition(int arr[], int low, int high)
{                                              0      11
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 48
  left=low;            // left = 0
  right =high;         // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
if(left < right)
   {
    t=arr[left];
    arr[left] = arr[right] ;
    arr[right] = t;
    left++; right--;
   }
   else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;     // right =
4
}// end of function
```



0	1	2	3	4	5	6	7	8	9	10	11
48	44	19	8	42	80	72	65	82	59	95	68

```
public static int partition(int arr[], int low, int high)
{                                                  0        11
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 48
  left=low;            // left = 0
  right =high;         // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;

   if(left < right)
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;    // right = 4
}// end of function
```



0	1	2	3	4	5	6	7	8	9	10	11
42	44	19	8	48	80	72	65	82	59	95	68

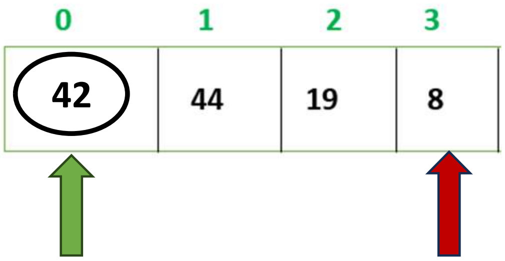
```
public static int partition(int arr[], int low, int high)
{                                                    0        11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 48
  left=low;             // left = 0
  right =high;          // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;

if(left < right)
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right =
4
}// end of function
```

0	1	2	3	4	5	6	7	8	9	10	11
42	44	19	8	48	80	72	65	82	59	95	68

```
public static int partition(int arr[], int low, int high)
{                                              0       11
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 48
  left=low;            // left = 0
  right =high;         // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

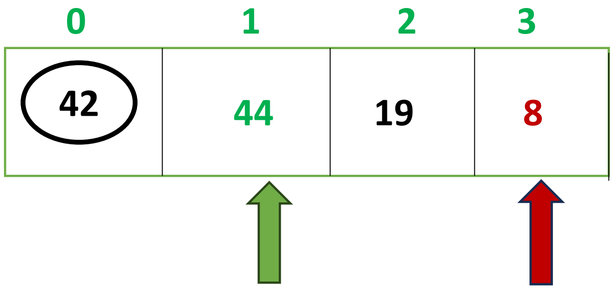
```
     if(left < right)
       {
        t=arr[left];
        arr[left] = arr[right] ;
        arr[right] = t;
        left++; right--;
       }
       else left++;
    }  // end of while loop

    arr[low] = arr[right];
    arr[right] =pivot;

    return right;   // right = 4
}// end of function
```

```
public static void QuickSort(int arr[],
            int low,int high)
{                   //0       11
  int pivloc;

  if(low>=high)  return;

  pivloc = partition(arr,low,high); //4

  QuickSort(arr,low,pivloc-1);
               //    0       3
  QuickSort(arr,pivloc+1, high);
               //     5         11
}
```



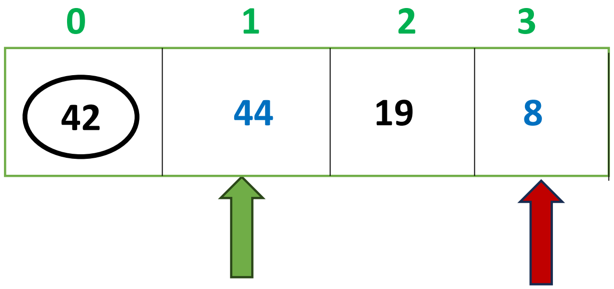
0	1	2	3
42	44	19	8

```
public static int partition(int arr[], int low, int high)
{                                              0       3
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 42
  left=low;            // left = 0
  right =high;         // right = 3

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```



0	1	2	3
42	44	19	8

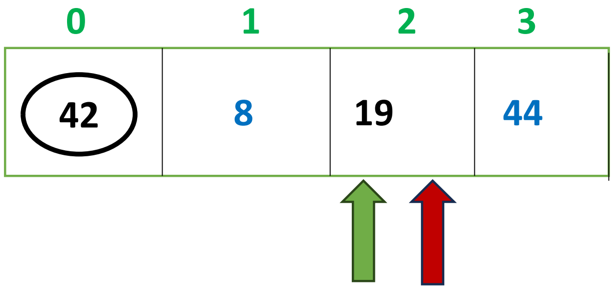
```
public static int partition(int arr[], int low, int high)
{                                                    0         3
   int left, right, pivot, t;

   pivot=arr[low];        // pivot = 42
   left=low;             // left = 0
   right =high;          // right = 3

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
        left++;

     while(arr[right] > pivot)
        right--;
```



0	1	2	3
42	44	19	8

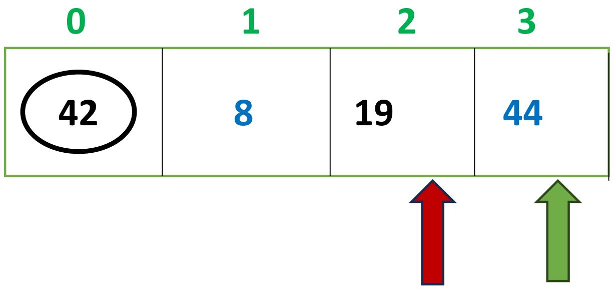
```
public static int partition(int arr[], int low, int high)
{                                                           0        3
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 42
  left=low;             // left = 0
  right =high;          // right = 3

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
if(left < right)
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right = 3
}// end of function
```

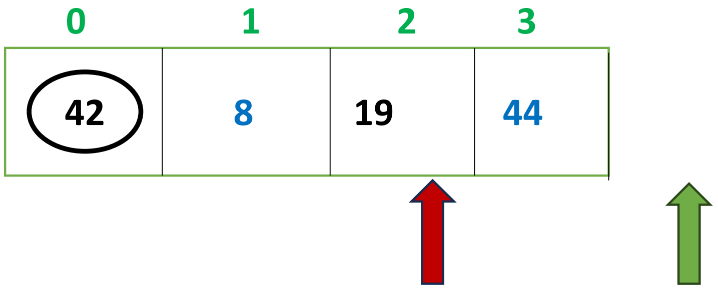


0	1	2	3
42	8	19	44

```
public static int partition(int arr[], int low, int high)
{                                                        0       3
   int left, right, pivot, t;

   pivot=arr[low];       // pivot = 42
   left=low;             // left = 0
   right =high;          // right = 3

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
        left++;

     while(arr[right] > pivot)
        right--;
```



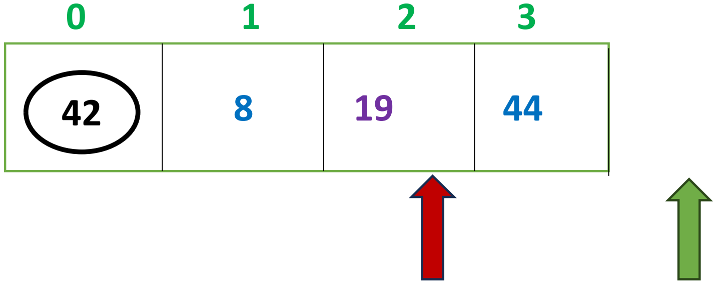
```
public static int partition(int arr[], int low, int high)
{                                                   0        3
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 42
  left=low;             // left = 0
  right =high;          // right = 3

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```



0	1	2	3
42	8	19	44

```
public static int partition(int arr[], int low, int high)
{                                                0        3
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 42
  left=low;            // left = 0
  right =high;         // right = 3

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

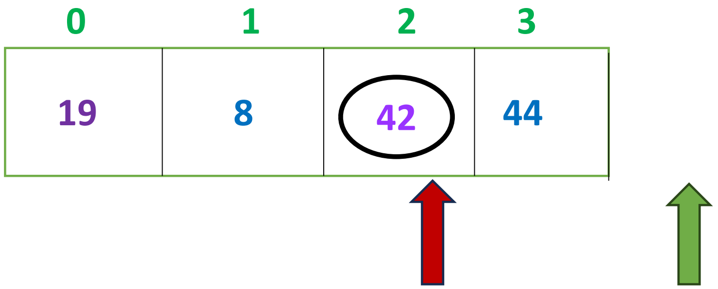
```
     if(left < right)    //false
       {
         t=arr[left];
         arr[left] = arr[right] ;
         arr[right] = t;
         left++; right--;
       }
       else left++;
   }  // end of while loop
```



0	1	2	3
42	8	19	44

```
public static int partition(int arr[], int low, int high)
{                                                 0      3
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 42
  left=low;             // left = 0
  right =high;          // right = 3

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
if(left < right)    //false
   {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
   }
   else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right = 2
}// end of function
```

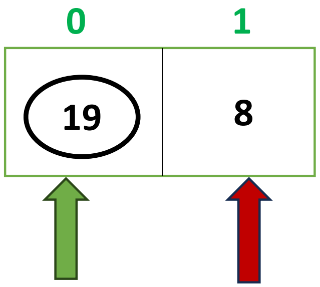


0	1	2	3
19	8	42	44

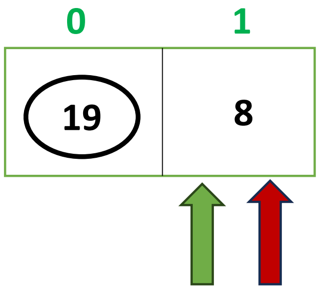
```
public static int partition(int arr[], int low, int high)
{                                                          0        3
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 42
  left=low;            // left = 0
  right =high;         // right = 3

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
if(left < right)    //false
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;       // right =
2
}// end of function
```

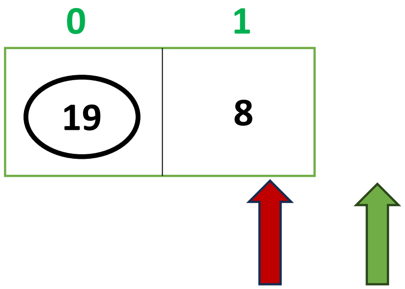
```
public static void QuickSort(int arr[],
              int low,int high)
{             //   0          3
  int pivloc;

  if(low>=high)  return;

  pivloc = partition(arr,low,high);  //2

  QuickSort(arr,low,pivloc-1);
                //    0          1
  QuickSort(arr,pivloc+1, high);
                //      3          3
}
```



```
public static int partition(int arr[], int low, int high)
{                                                    0        1
   int left, right, pivot, t;

   pivot=arr[low];       // pivot = 19
   left=low;             // left = 0
   right =high;          // right = 1

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
        left++;

     while(arr[right] > pivot)
        right--;
```

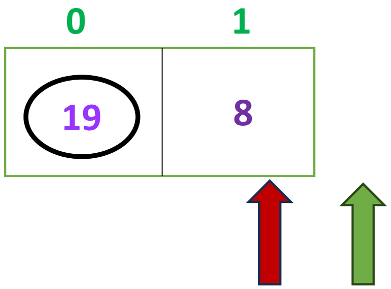


```
public static int partition(int arr[], int low, int high)
{                                                    0        1
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 19
  left=low;            // left = 0
  right =high;         // right = 1

  while(left <= right)
  {
    while((arr[left]<=pivot) && (left<high))
       left++;

    while(arr[right] > pivot)
       right--;
```

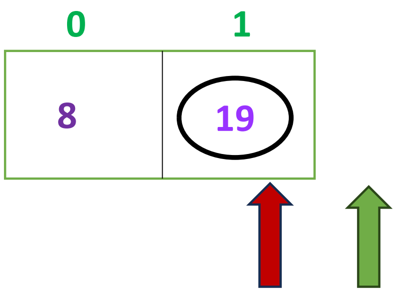


```
public static int partition(int arr[], int low, int high)
{                                                    0       1
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 19
  left=low;            // left = 0
  right =high;         // right = 1

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
if(left < right)    //false
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;       // right =
1
}// end of function
```



```
public static int partition(int arr[], int low, int high)
{                                                 0         1
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 19
  left=low;             // left = 0
  right =high;          // right = 1

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;

     if(left < right)    //false
      {
        t=arr[left];
        arr[left] = arr[right] ;
        arr[right] = t;
        left++; right--;
      }
      else left++;
   }  // end of while loop

   arr[low] = arr[right];
   arr[right] =pivot;

   return right;      // right = 1
}// end of function
```



```
public static int partition(int arr[], int low, int high)
{                                                   0      1
  int left, right, pivot, t;

  pivot=arr[low];        // pivot = 19
  left=low;              // left = 0
  right =high;           // right = 1

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
if(left < right)    //false
   {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
   }
   else left++;
 }  // end of while loop

 arr[low] = arr[right];
 arr[right] =pivot;

 return right;  // right = 1
}// end of function
```

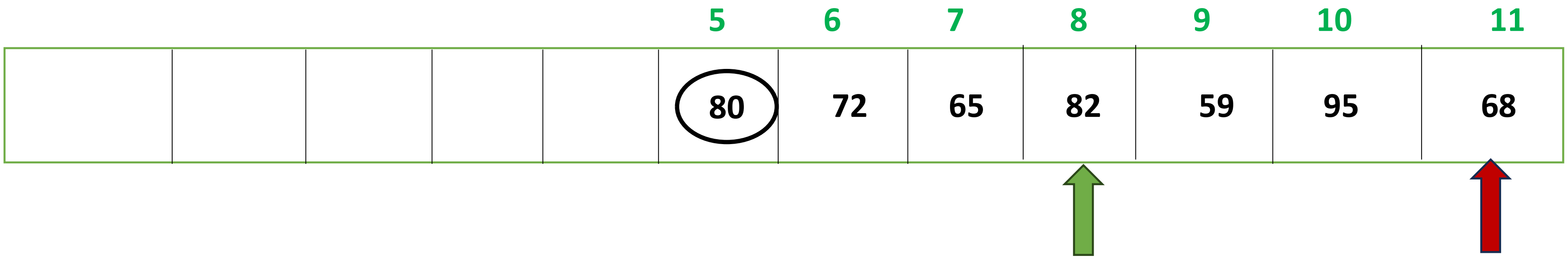
```
public static void QuickSort(int arr[],
               int low,int high)
{
  int pivloc;

  if(low>=high)  return;

  pivloc = partition(arr,low,high);  //1

  QuickSort(arr,low,pivloc-1);
               //    0         0         //*****
  QuickSort(arr,pivloc+1, high);
               //      2          1     //*****
}
```

0	1	2	3	4	5	6	7	8	9	10	11
8	19	42	44	48	80	72	65	82	59	95	68

```
public static int partition(int arr[], int low, int high)
{                                                         5        11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 80
  left=low;             // left = 5
  right =high;          // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

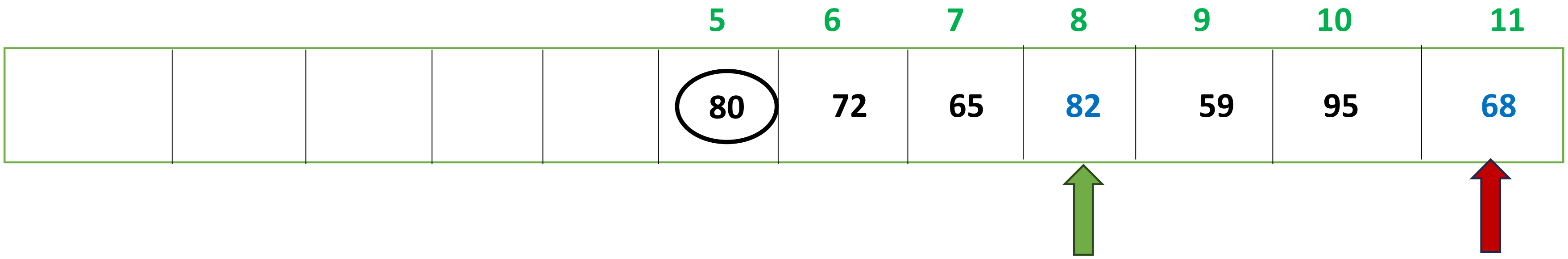
```
if(left < right)
   {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
   }
   else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;   // right = 4
}// end of function
```

```
public static void QuickSort(int arr[],
               int low,int high)
{
  int pivloc;

  if(low>=high)  return;

  pivloc = partition(arr,low,high);

  QuickSort(arr,low,pivloc-1);
               //    0          3
  QuickSort(arr,pivloc+1, high);
               //      5          11
}
```



					5	6	7	8	9	10	11
					80	72	65	82	59	95	68

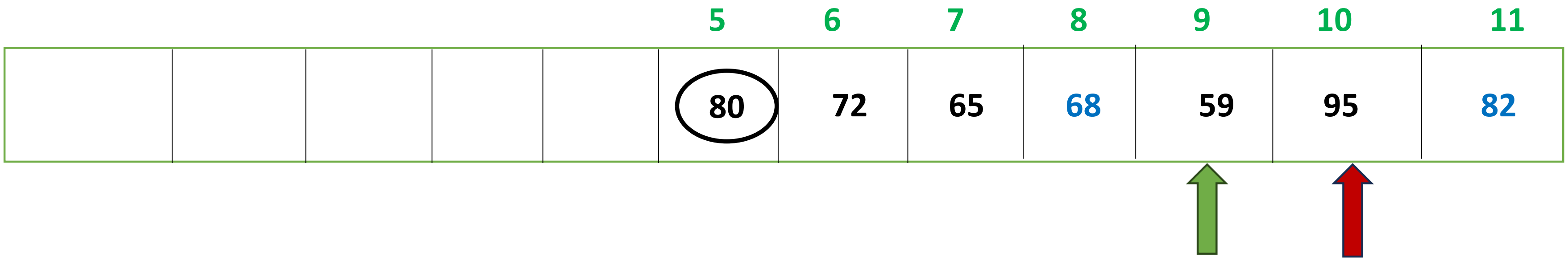
```
public static int partition(int arr[], int low, int high)
{                                                  5        11
   int left, right, pivot, t;

   pivot=arr[low];       // pivot = 80
   left=low;             // left = 5
   right =high;          // right = 11

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
        left++;

     while(arr[right] > pivot)
        right--;
```



					5	6	7	8	9	10	11
					80	72	65	82	59	95	68

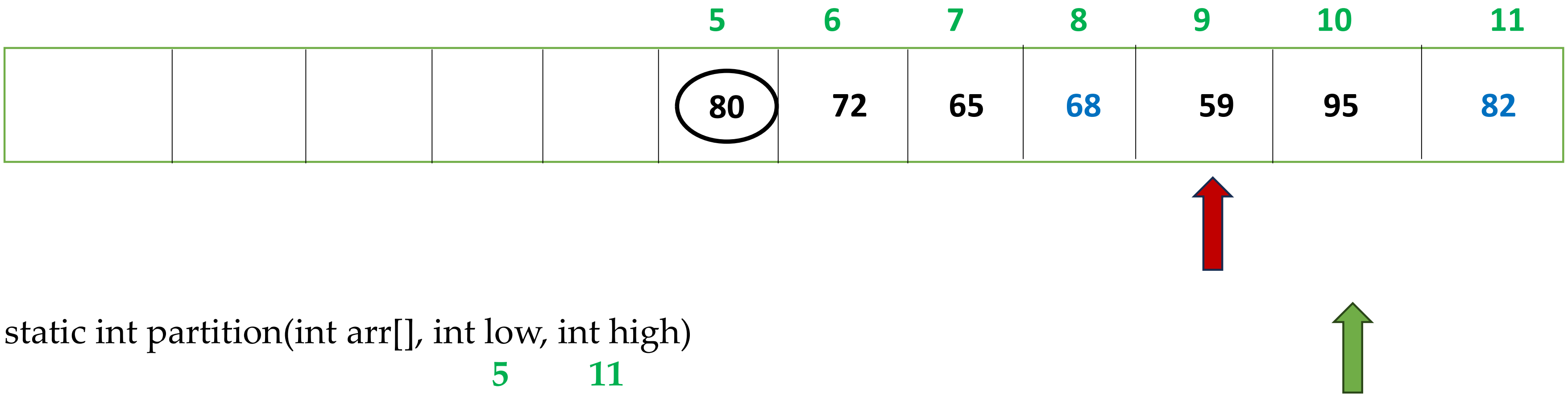
```
public static int partition(int arr[], int low, int high)
{                                              5       11
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 80
  left=low;           // left = 5
  right =high;        // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
if(left < right)    //true
   {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
   }
   else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right =
2
}// end of function
```



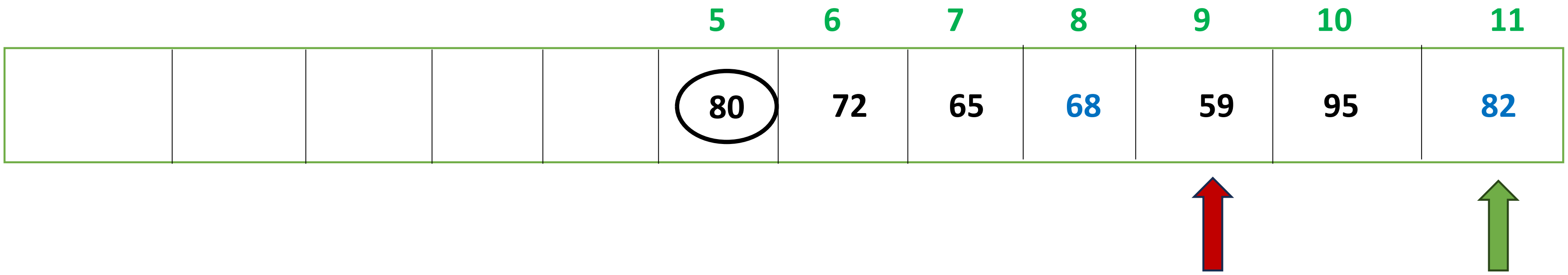
					5	6	7	8	9	10	11
					80	72	65	68	59	95	82

```
public static int partition(int arr[], int low, int high)
{                                                 5        11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 80
  left=low;             // left = 5
  right =high;          // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;

    if(left < right)    //false
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop
```



					5	6	7	8	9	10	11
					80	72	65	68	59	95	82

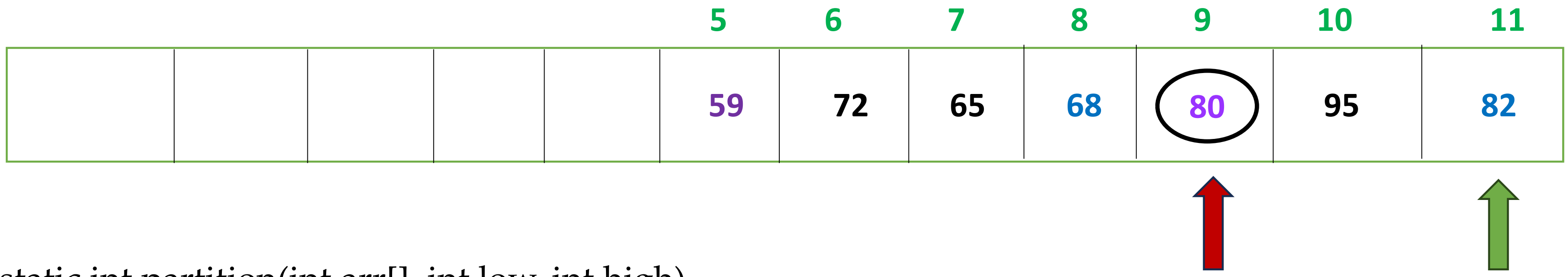
```
public static int partition(int arr[], int low, int high)
{                                                  5        11
   int left, right, pivot, t;

   pivot=arr[low];      // pivot = 80
   left=low;            // left = 5
   right =high;         // right = 11

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
        left++;

     while(arr[right] > pivot)
        right--;
```



					5	6	7	8	9	10	11
					80	72	65	68	59	95	82

```
public static int partition(int arr[], int low, int high)
{                                               5        11
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 80
  left=low;            // left = 5
  right =high;         // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

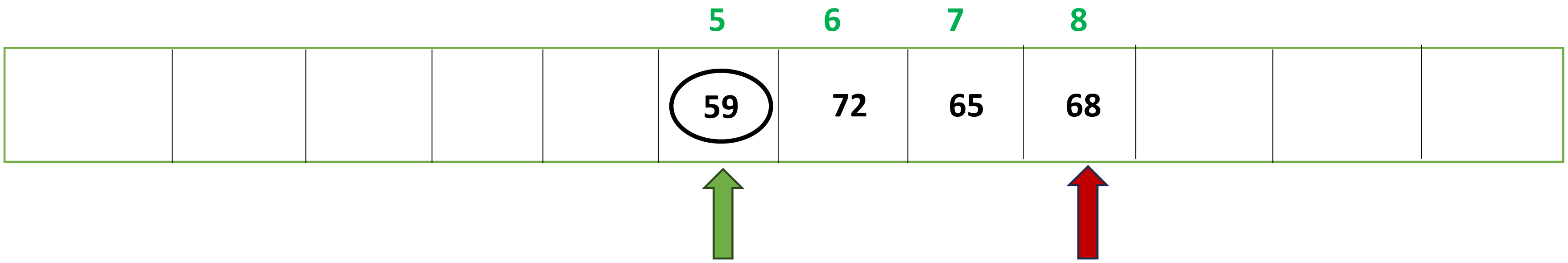
```
if(left < right)    //false
   {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
   }
   else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;     // right =
9
}// end of function
```



					5	6	7	8	9	10	11
					59	72	65	68	80	95	82

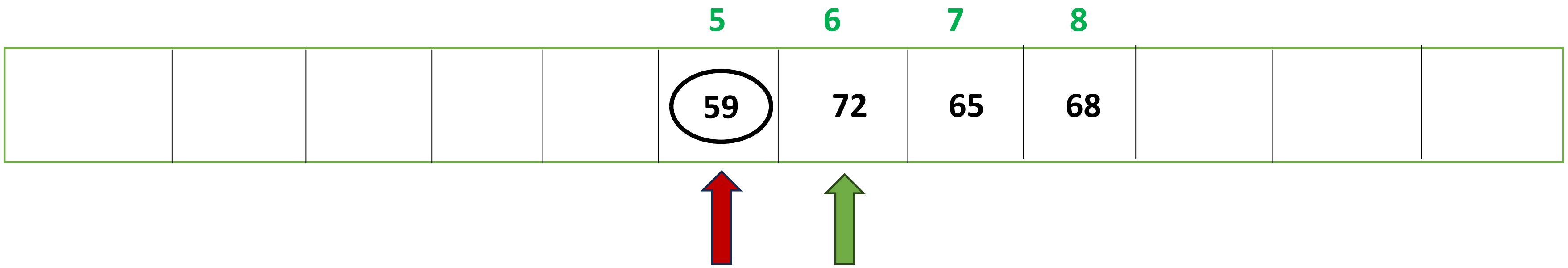
```
public static int partition(int arr[], int low, int high)
{                                                          5        11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 80
  left=low;             // left = 5
  right =high;          // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;

   if(left < right)    //false
     {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
     }
     else left++;
   }  // end of while loop

   arr[low] = arr[right];
   arr[right] =pivot;

   return right;  // right = 9
}// end of function
```

```
public static void QuickSort(int arr[],
              int low, int high)
{             // 5          11
  int pivloc;

  if(low>=high)  return;

  pivloc = partition(arr,low,high); //9

  QuickSort(arr,low,pivloc-1);
             //    5         8
  QuickSort(arr,pivloc+1, high);
             //     10        11
}
```



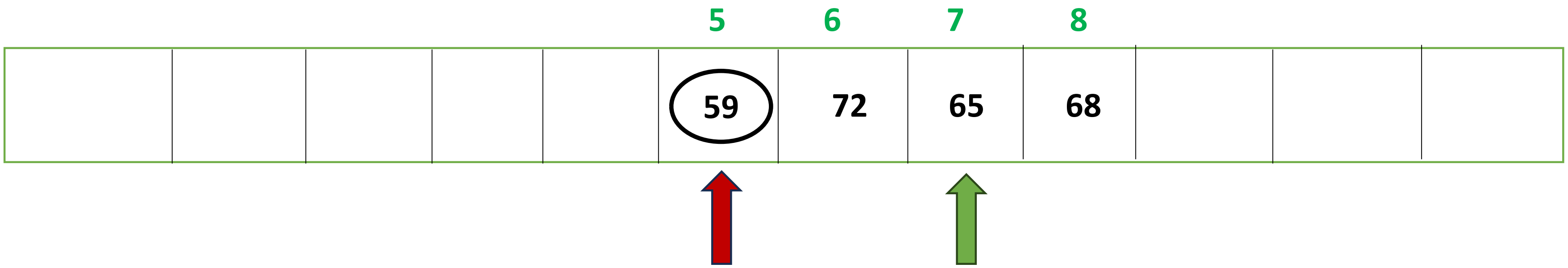
					5	6	7	8			
					59	72	65	68			

```
public static int partition(int arr[], int low, int high)
{                                                    5       8
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 59
  left=low;             // left = 5
  right =high;          // right = 8

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```



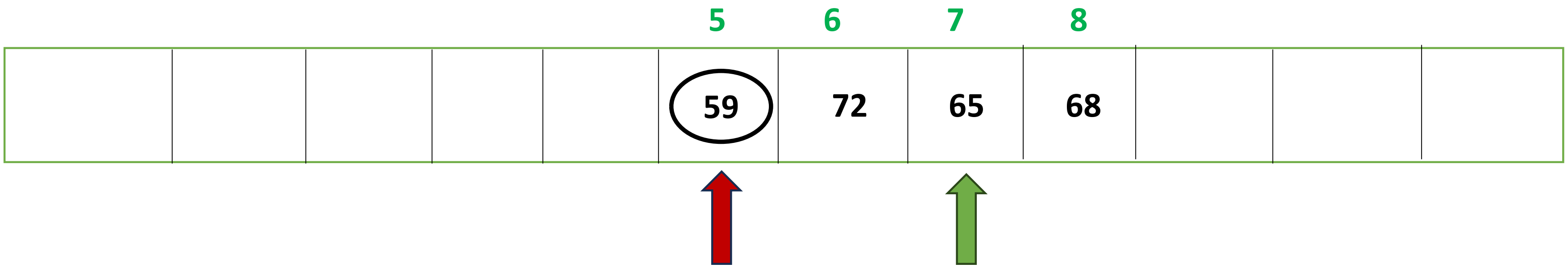
					5	6	7	8			
					59	72	65	68			

```
public static int partition(int arr[], int low, int high)
{                                                        5         8
   int left, right, pivot, t;

   pivot=arr[low];       // pivot = 59
   left=low;             // left = 5
   right =high;          // right = 8

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
         left++;

     while(arr[right] > pivot)
         right--;
```



					5	6	7	8			
					59	72	65	68			

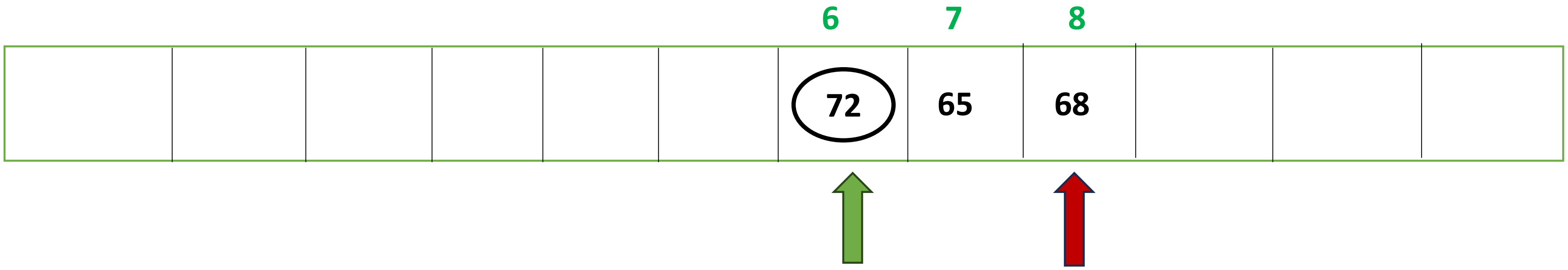
```
public static int partition(int arr[], int low, int high)
{                                                   5        8
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 59
  left=low;            // left = 5
  right =high;         // right = 8

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;

   if(left < right)    //false
     {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
     }
     else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right = 5
}// end of function
```



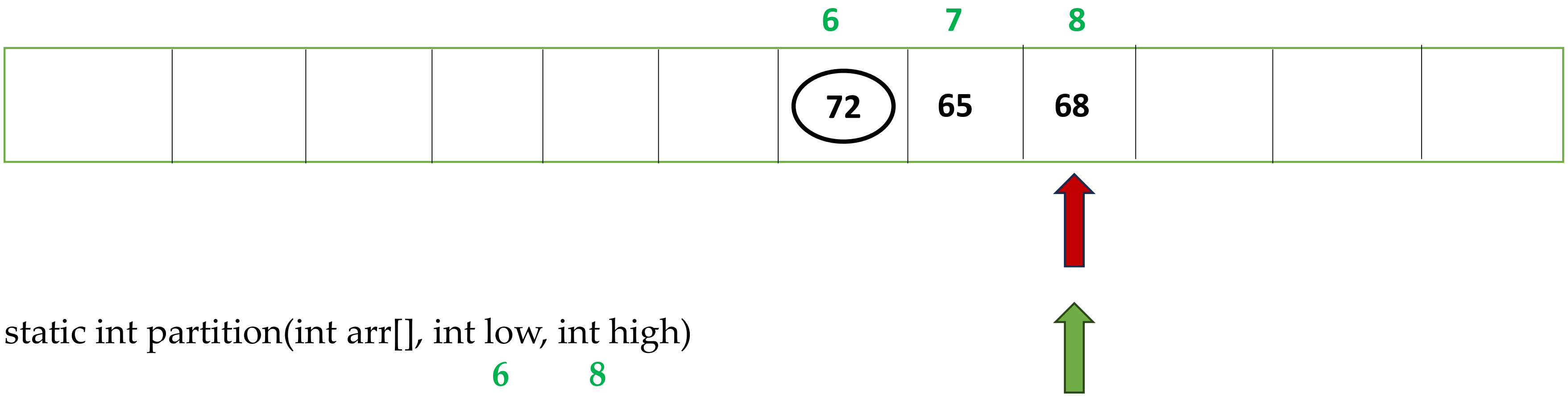
```
public static int partition(int arr[], int low, int high)
{                                              5       8
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 59
  left=low;            // left = 5
  right =high;         // right = 8

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
  if(left < right)    //false
    {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;  // right = 5
}// end of function
```

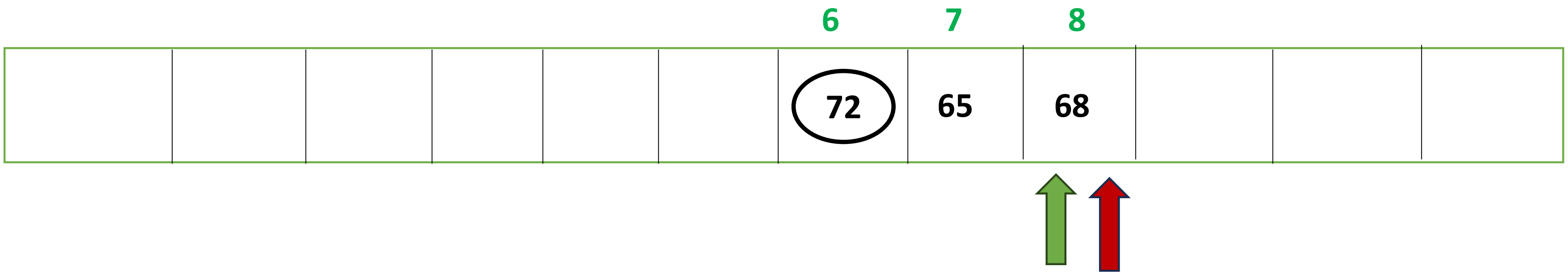
```
public static void QuickSort(int arr[],
              int low,int high)
{                 // 5        8
  int pivloc;

  if(low>=high)  return;

  pivloc = partition(arr,low,high); //5

  QuickSort(arr,low,pivloc-1);
                //    5        4
  QuickSort(arr,pivloc+1, high);
                //      6        8
}
```



```
public static int partition(int arr[], int low, int high)
{                                                   6        8
   int left, right, pivot, t;

   pivot=arr[low];        // pivot = 72
   left=low;              // left = 6
   right =high;           // right = 8

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
        left++;

     while(arr[right] > pivot)
        right--;
```



```
public static int partition(int arr[], int low, int high)
{                                                   6       8
   int left, right, pivot, t;

   pivot=arr[low];       // pivot = 72
   left=low;             // left = 6
   right =high;          // right = 8

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
        left++;

     while(arr[right] > pivot)
        right--;
```

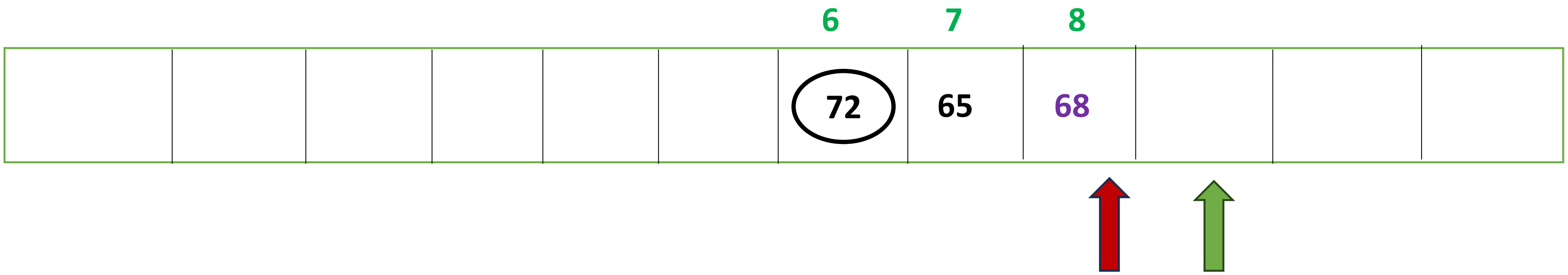


6 7 8

72 65 68

```
public static int partition(int arr[], int low, int high)
{                                              6        8
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 72
  left=low;            // left = 6
  right =high;         // right = 8

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

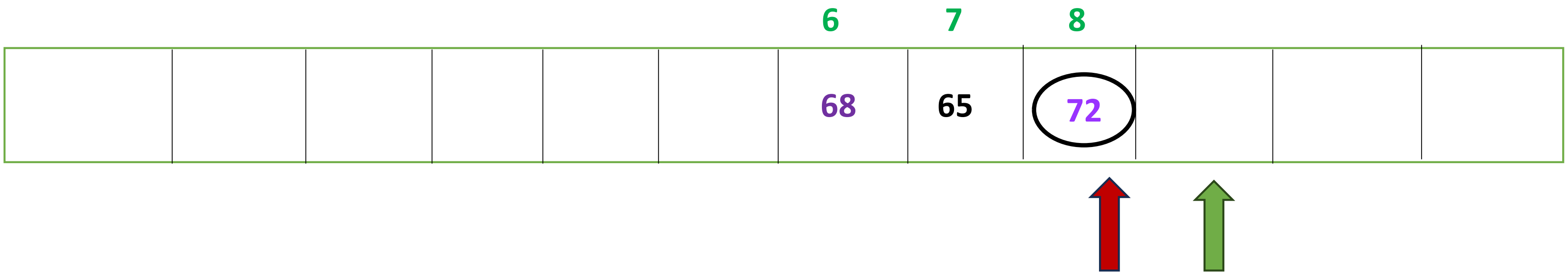
```
if(left < right)    //false
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right =
}// end of function
```



						6	7	8			
						72	65	68			

```
public static int partition(int arr[], int low, int high)
{                                                     6        8
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 72
  left=low;             // left = 6
  right =high;          // right = 8

  while(left <= right)
  {
    while((arr[left]<=pivot) && (left<high))
       left++;

    while(arr[right] > pivot)
       right--;
```

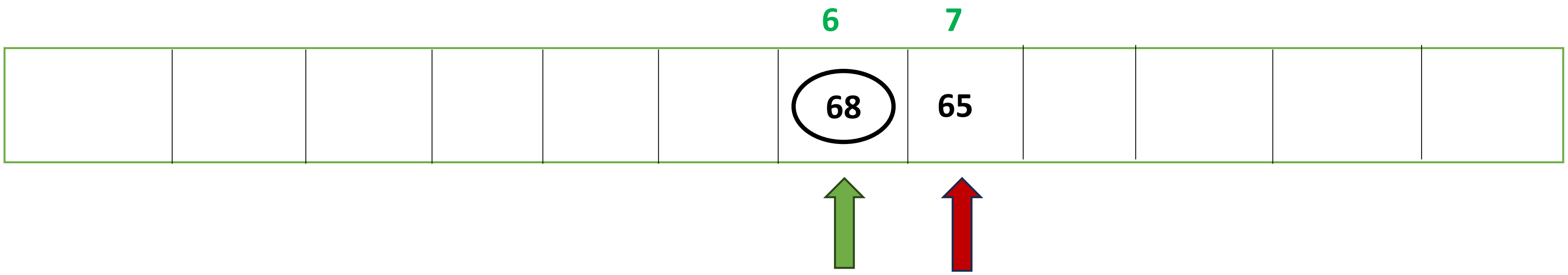
```
  if(left < right)    //false
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right =
}// end of function
```



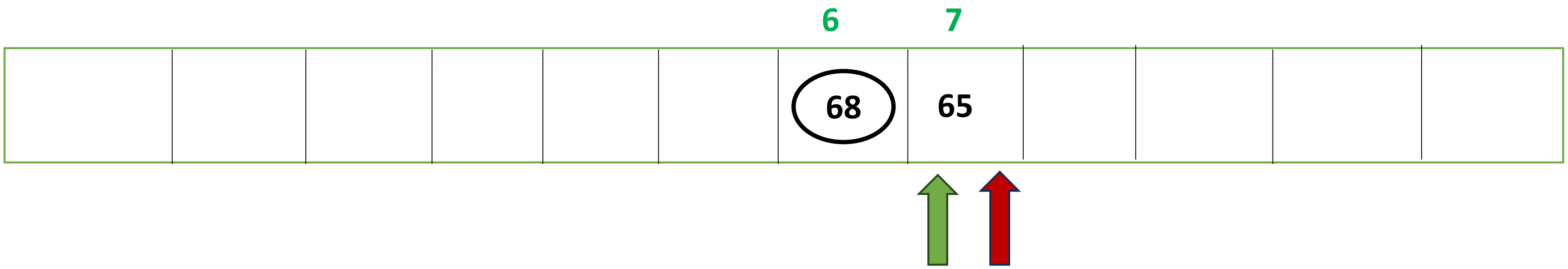
```
public static int partition(int arr[], int low, int high)
{                                                    6        8
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 72
  left=low;             // left = 6
  right =high;          // right = 8

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
if(left < right)    //false
   {
    t=arr[left];
    arr[left] = arr[right] ;
    arr[right] = t;
    left++; right--;
   }
   else left++;
 }  // end of while loop

 arr[low] = arr[right];
 arr[right] =pivot;

 return right;  // right = 8
}// end of function
```

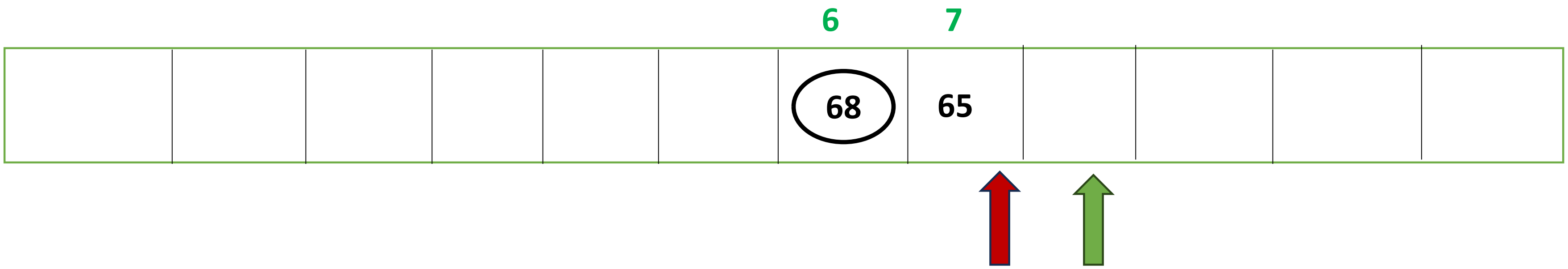
```
public static void QuickSort(int arr[],
              int low,int high)
{                // 6       8
  int pivloc;

  if(low>=high)  return;

  pivloc = partition(arr,low,high); //8

  QuickSort(arr,low,pivloc-1);
                 //   6       7
  QuickSort(arr,pivloc+1, high);
                 //     9      8
}
```



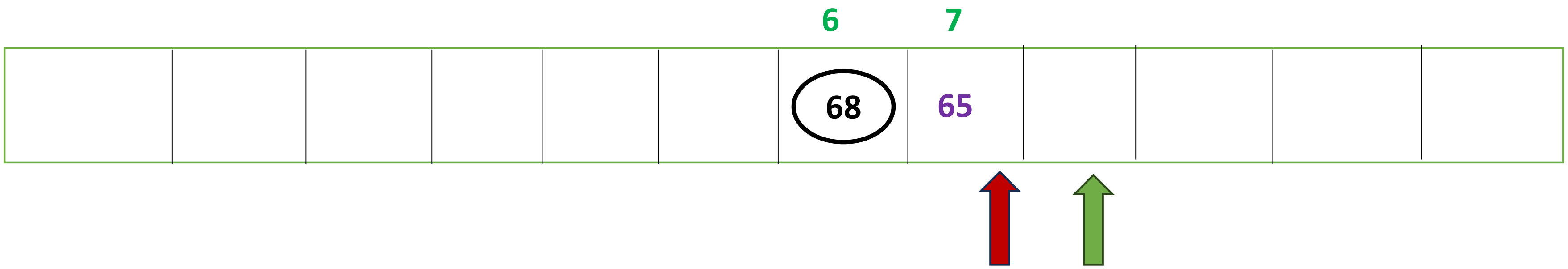
```
public static int partition(int arr[], int low, int high)
{                                                6       7
   int left, right, pivot, t;

   pivot=arr[low];        // pivot = 68
   left=low;              // left = 6
   right =high;           // right = 7

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
        left++;

     while(arr[right] > pivot)
        right--;
```



```
public static int partition(int arr[], int low, int high)
{                                            6       7
   int left, right, pivot, t;

   pivot=arr[low];      // pivot = 68
   left=low;            // left = 6
   right =high;         // right = 7

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
        left++;

     while(arr[right] > pivot)
        right--;
```



```
public static int partition(int arr[], int low, int high)
{                                                    6         7
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 68
  left=low;             // left = 6
  right =high;          // right = 7

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;

   if(left < right)    //false
    {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right = 8
}// end of function
```

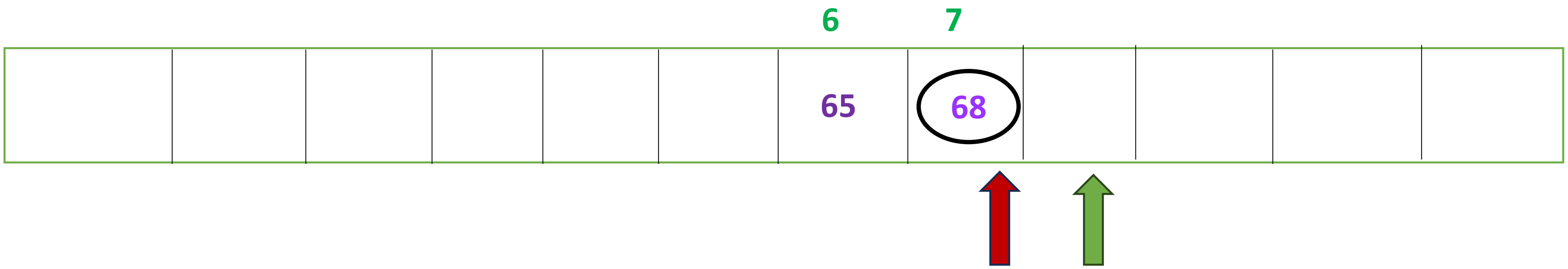


6 7

68 65

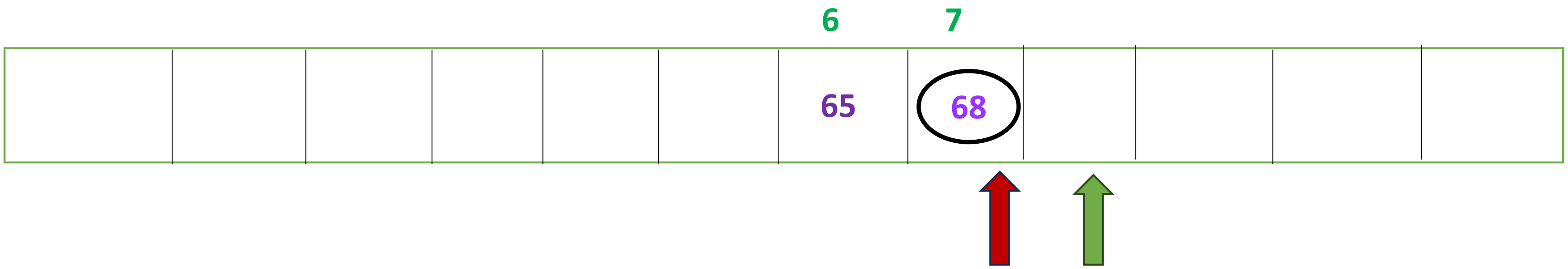
```
public static int partition(int arr[], int low, int high)
{                                                   6       7
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 68
  left=low;            // left = 6
  right =high;         // right = 7

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
if(left < right)    //false
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right =
8
}// end of function
```



```
public static int partition(int arr[], int low, int high)
{                                                        6        7
   int left, right, pivot, t;

   pivot=arr[low];       // pivot = 68
   left=low;             // left = 6
   right =high;          // right = 7

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
        left++;

     while(arr[right] > pivot)
        right--;
```

```
if(left < right)    //false
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right =
7
}// end of function
```



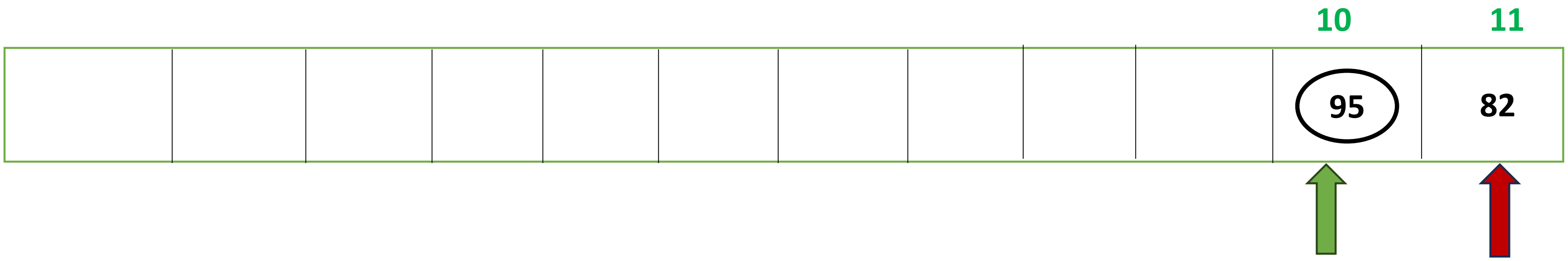
```
public static int partition(int arr[], int low, int high)
{                                                  6       7
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 68
  left=low;             // left = 6
  right =high;          // right = 7

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;

   if(left < right)    //false
    {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;  // right = 7
}// end of function
```

```
public static void QuickSort(int arr[],
               int low,int high)
{                // 6        7
 int pivloc;

 if(low>=high)  return;

 pivloc = partition(arr,low,high); //7

 QuickSort(arr,low,pivloc-1);
                //    6        6
 QuickSort(arr,pivloc+1, high);
                //     8        7
}
```

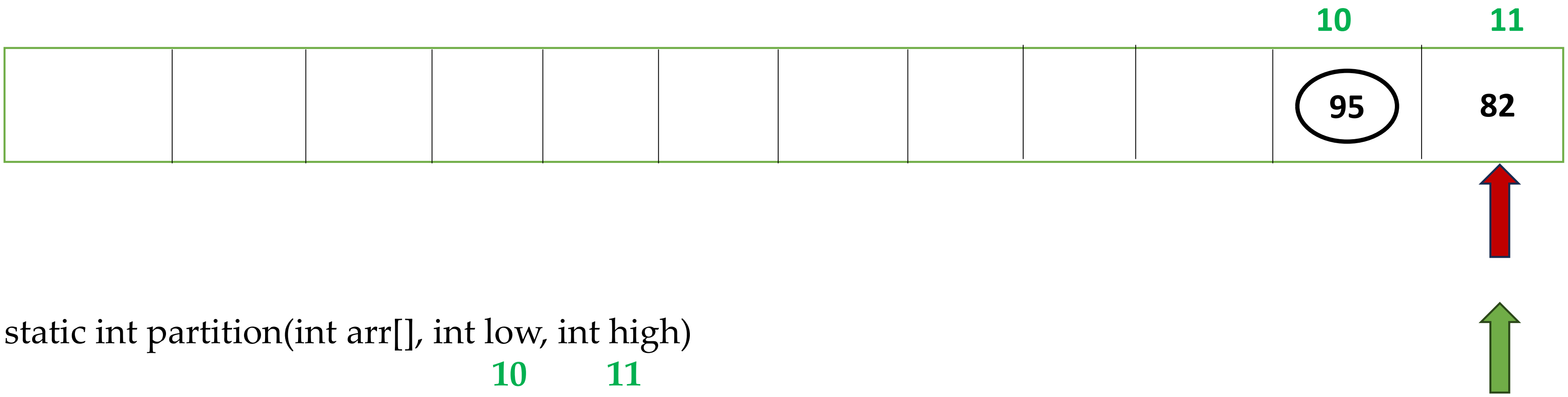
										10	11
										95	82

```
public static int partition(int arr[], int low, int high)
{                                                   5       11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 95
  left=low;             // left = 5
  right =high;          // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;

if(left < right)    //false
    {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right; // right = 9
}// end of function
```

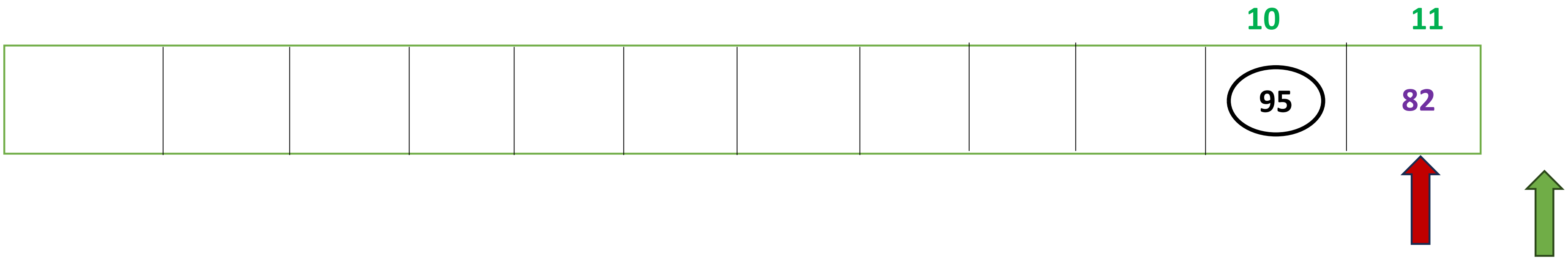
```
public static void QuickSort(int arr[],
                int low, int high)
{               // 5           11
  int pivloc;

  if(low>=high)  return;

  pivloc = partition(arr,low,high); //9

  QuickSort(arr,low,pivloc-1);
                //    5         8
  QuickSort(arr,pivloc+1, high);
                //      10         11
}
```



```
public static int partition(int arr[], int low, int high)
{                                                    10        11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 95
  left=low;             // left = 10
  right =high;          // right = 11

  while(left <= right)
  {
    while((arr[left]<=pivot) && (left<high))
       left++;

    while(arr[right] > pivot)
       right--;
```

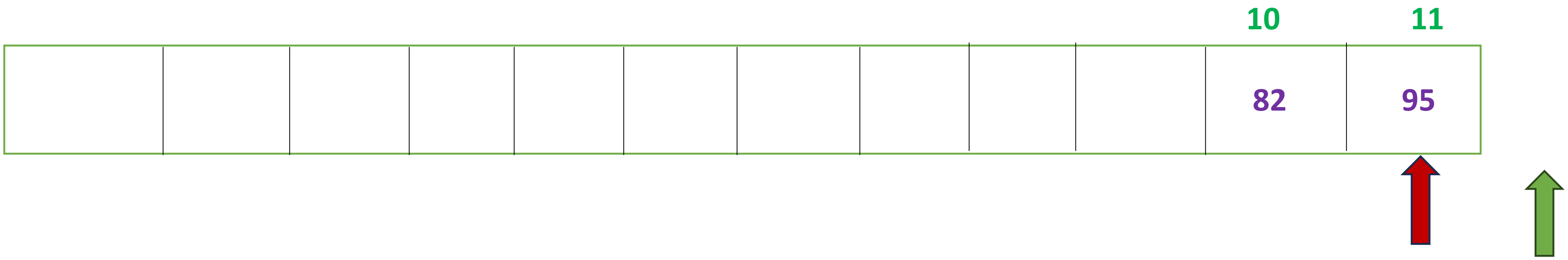


```
public static int partition(int arr[], int low, int high)
{                                                 10        11
   int left, right, pivot, t;

   pivot=arr[low];       // pivot = 95
   left=low;             // left = 10
   right =high;          // right = 11

   while(left <= right)
   {
     while((arr[left]<=pivot) && (left<high))
        left++;

     while(arr[right] > pivot)
        right--;
```



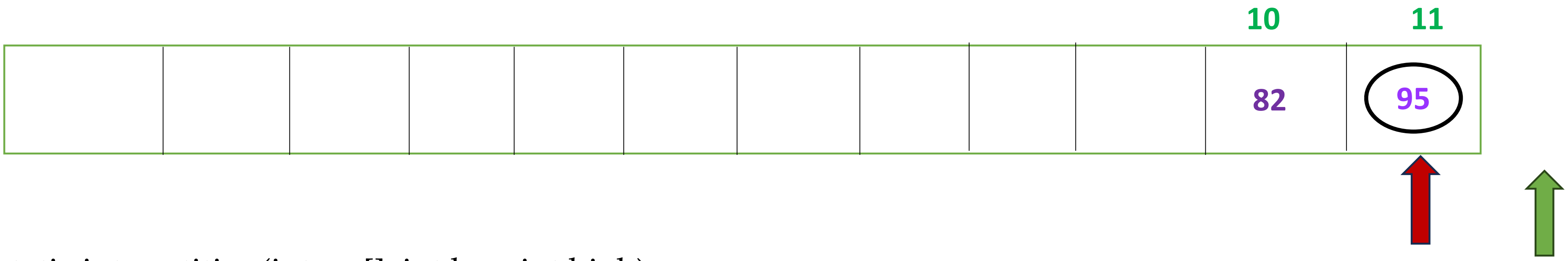
```
public static int partition(int arr[], int low, int high)
{                                              10      11
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 95
  left=low;            // left = 10
  right =high;         // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
if(left < right)    //false
   {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
   }
   else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right =
9
}// end of function
```



```
public static int partition(int arr[], int low, int high)
{                                                   10      11
  int left, right, pivot, t;

  pivot=arr[low];       // pivot = 95
  left=low;             // left = 10
  right =high;          // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
if(left < right)    //false
    {
      t=arr[left];
      arr[left] = arr[right] ;
      arr[right] = t;
      left++; right--;
    }
    else left++;
  }  // end of while loop

  arr[low] = arr[right];
  arr[right] =pivot;

  return right;      // right =
11
}// end of function
```



```
public static int partition(int arr[], int low, int high)
{                                                10        11
  int left, right, pivot, t;

  pivot=arr[low];      // pivot = 95
  left=low;            // left = 10
  right =high;         // right = 11

  while(left <= right)
  {
   while((arr[left]<=pivot) && (left<high))
      left++;

   while(arr[right] > pivot)
      right--;
```

```
  if(left < right)    //false
    {
     t=arr[left];
     arr[left] = arr[right] ;
     arr[right] = t;
     left++; right--;
    }
   else left++;
  }  // end of while loop

   arr[low] = arr[right];
   arr[right] =pivot;

   return right;      // right = 11
}// end of function
```

```
public static void QuickSort(int arr[],
             int low, int high)
{               // 10          11
  int pivloc;

  if(low>=high)  return;

  pivloc = partition(arr,low,high); //11

  QuickSort(arr,low,pivloc-1);
              //    10        10
  QuickSort(arr,pivloc+1, high);
              //      12        11
}
```

Performance – Quick Sort

- Worst case Complexity: O(n^2)
- Best case Complexity: O(nlogn)
- Average case Complexity: O(nlogn)
- Worst case space Complexity: O(1)