QUICK SORT

The name "Quick Sort" comes from the fact that, quick sort is capable of sorting a list of data elements significantly faster (twice or thrice faster) than any of the common sorting algorithms.

- Quick sort is a divide and conquer algorithm.
- Pick a element, called pivot, from the list.
- Partition the list so that the smaller elements are before the pivot, and the larger elements after the pivot.
- Recursively sort the sub-lists.

QUICK SORT

- Quick sort provides a fast and methodical approach to sort any lists of things.

- some of the applications where quick sort is used are,
  - ***Commercial computing***
  - ***Numerical computations***
  - ***Information search***

QUICK SORT

Quick sort follows the below steps:

Step 1 **−** Make any element as pivot.

Step 2 **−** Partition the array on the basis of pivot.

Step 3 **−** Apply quick sort on left partition recursively.

Step 4 **−** Apply quick sort on right partition recursively.

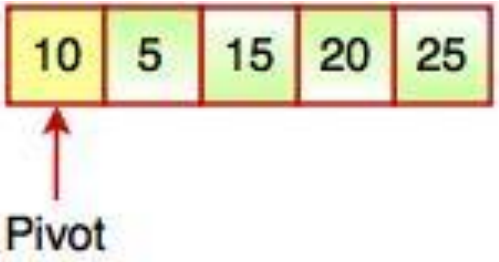
Quick Sort Algorithm

Step 1: Choose the highest index value as pivot.
Step 2: Take two variables to point left and right of the list excluding pivot.
Step 3: Left points to the low index.
Step 4: Right points to the high index.
Step 5: While value at left < (Less than) pivot move right.
Step 6: While value at right > (Greater than) pivot move left.
Step 7: If both Step 5 and Step 6 does not match, swap left and right.
Step 8: If left = (Less than or Equal to) right, the point where they met is new pivot.

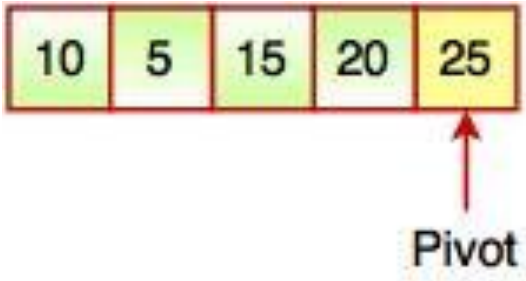
Quick Sort Algorithm

There are different versions of quick sort which choose the pivot in different ways:

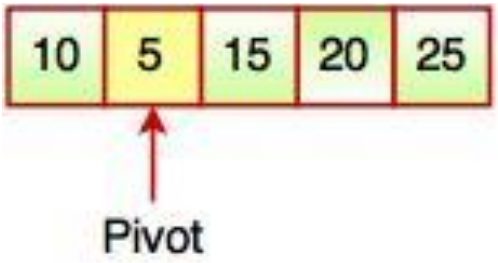
1. First element as pivot



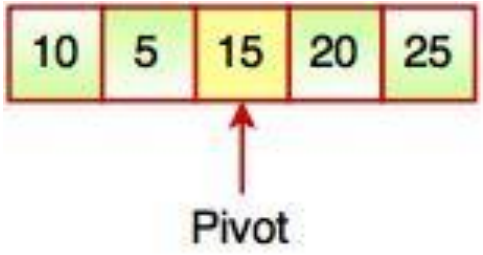
2. Last element as pivot



3. Random element as pivot



4. Median as pivot



Quick Sort Algorithm

```
quickSort(arr[], low, high)
{
    if (low < high)
    {
        pivot_index = partition(arr, low, high);

        quickSort(arr, low, pivot_index - 1);

            quickSort(arr, pivot_index + 1, high);

    }
}
```

- **Time complexity of Quick Sort Algorithm is O(nlogn)**

Partition Method

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

Quick Sort Implementation

```
# include <stdio.h>
// To swap two numbers
void swap(int* a, int* b)
{
        int t = *a;
        *a = *b;
        *b = t;
}
// Partition Method
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], &arr[j]);
     }
   }
   swap(&arr[i + 1], &arr[high]);
   return (i + 1);
}
```

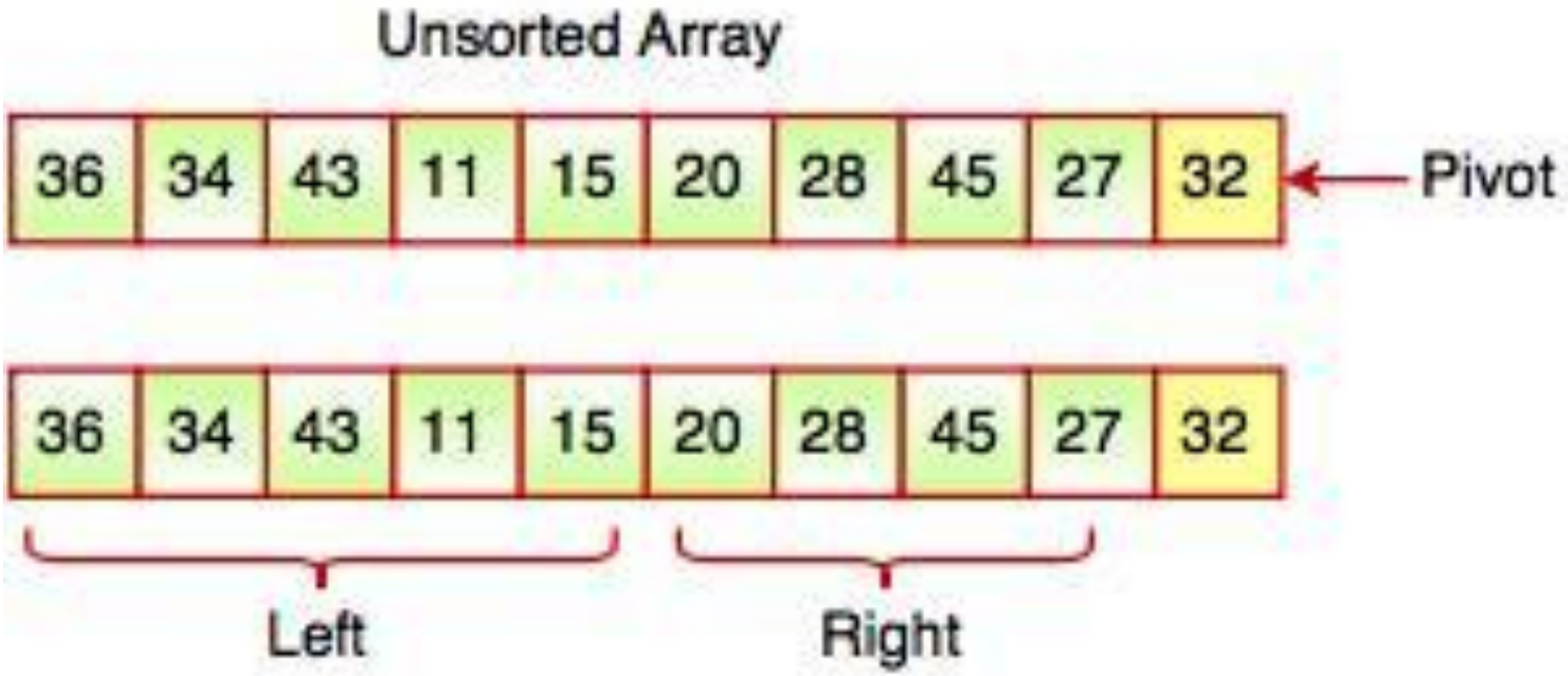
```
void quicksort(int a[], int p, int r)
{
   if(p < r)
   {
     int q; q = partition(a, p, r);
     quicksort(a, p, q-1);
     quicksort(a, q+1, r);
   }
}

void printArray(int a[], int size)
{
   int i;
   for (i=0; i < size; i++)
   {
     printf("%d ", a[i]);
   }
   printf("\n");
}
```

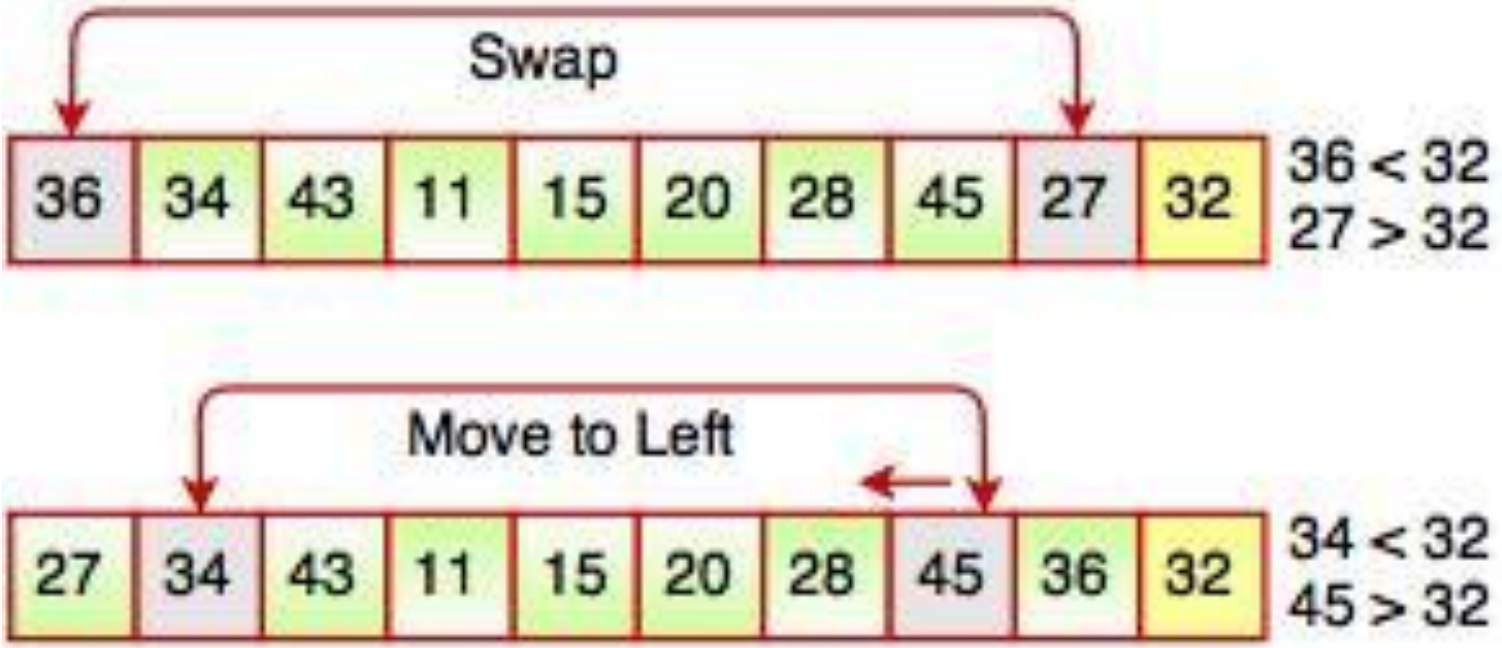
Quick Sort Implementation

```
int main()
{
   int arr[] = {9, 7, 5, 11, 12, 2, 14, 3, 10, 6};
   int n = sizeof(arr)/sizeof(arr[0]);
   // call quickSort function
   quicksort(arr, 0, n-1);
   printf("Sorted array: \n");
   printArray(arr, n);
   return 0;
}
```

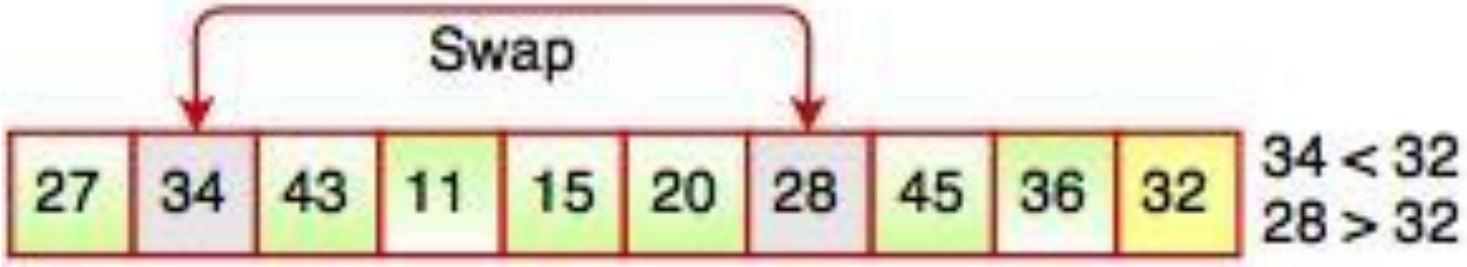
Quick Sort Example

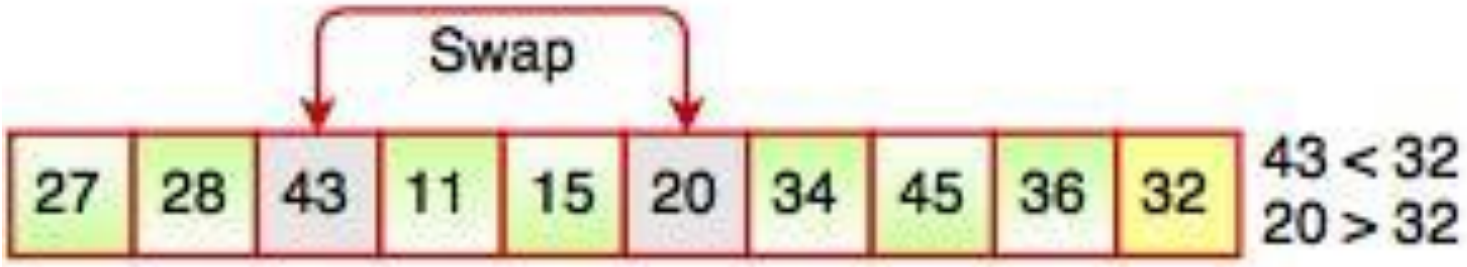


Quick Sort Example



Quick Sort Example





Quick Sort Example

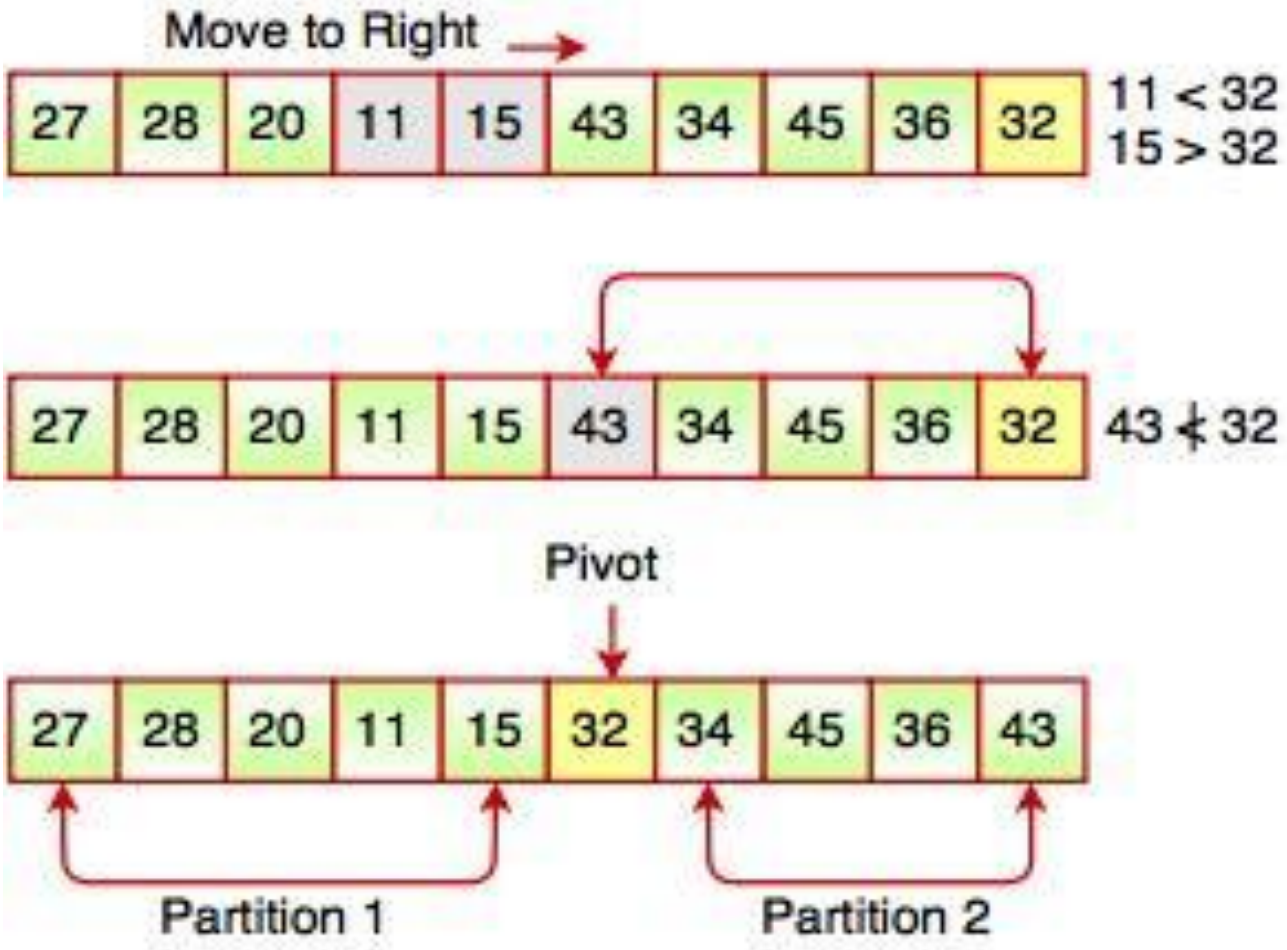
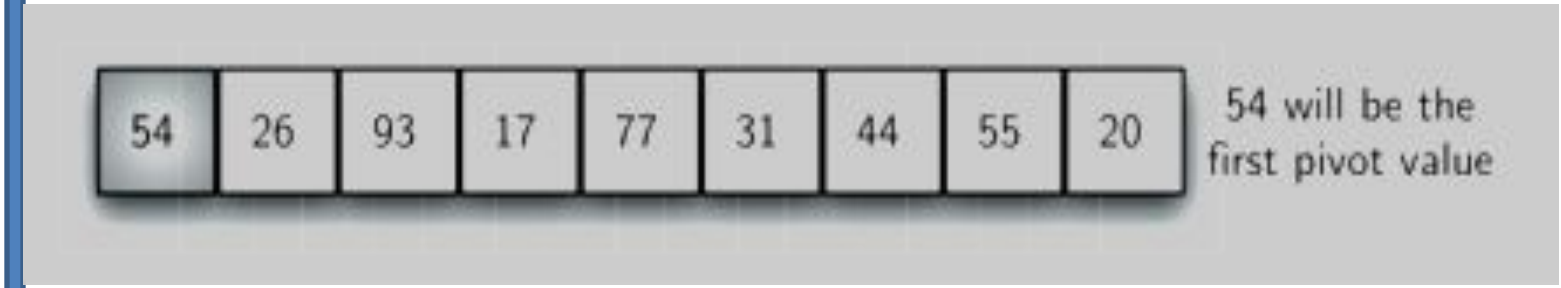
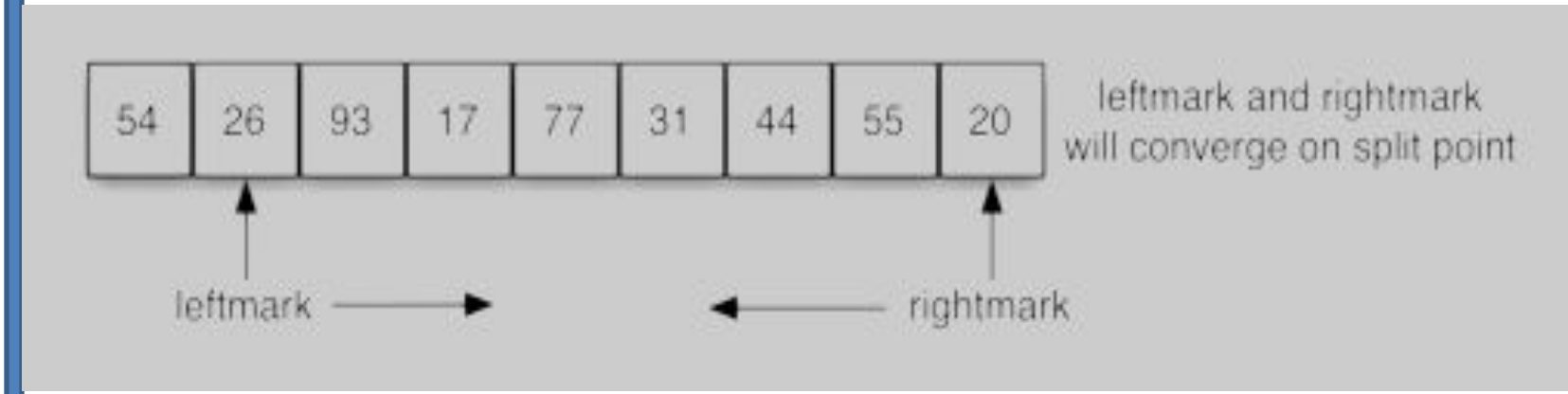


Fig. Finding Pivot Value in an Array

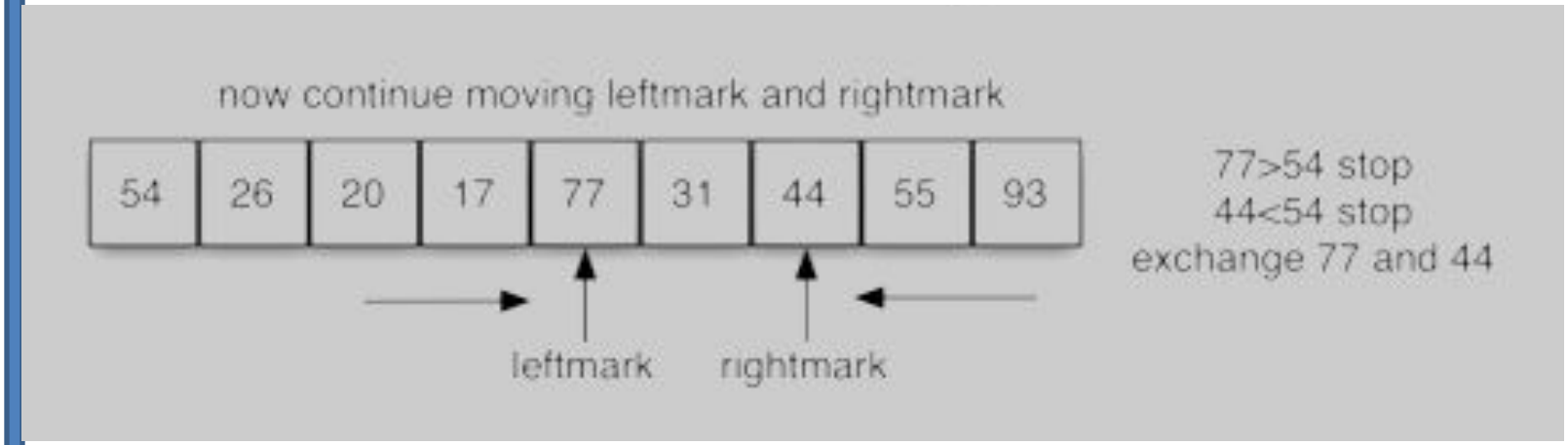
Quick Sort Example



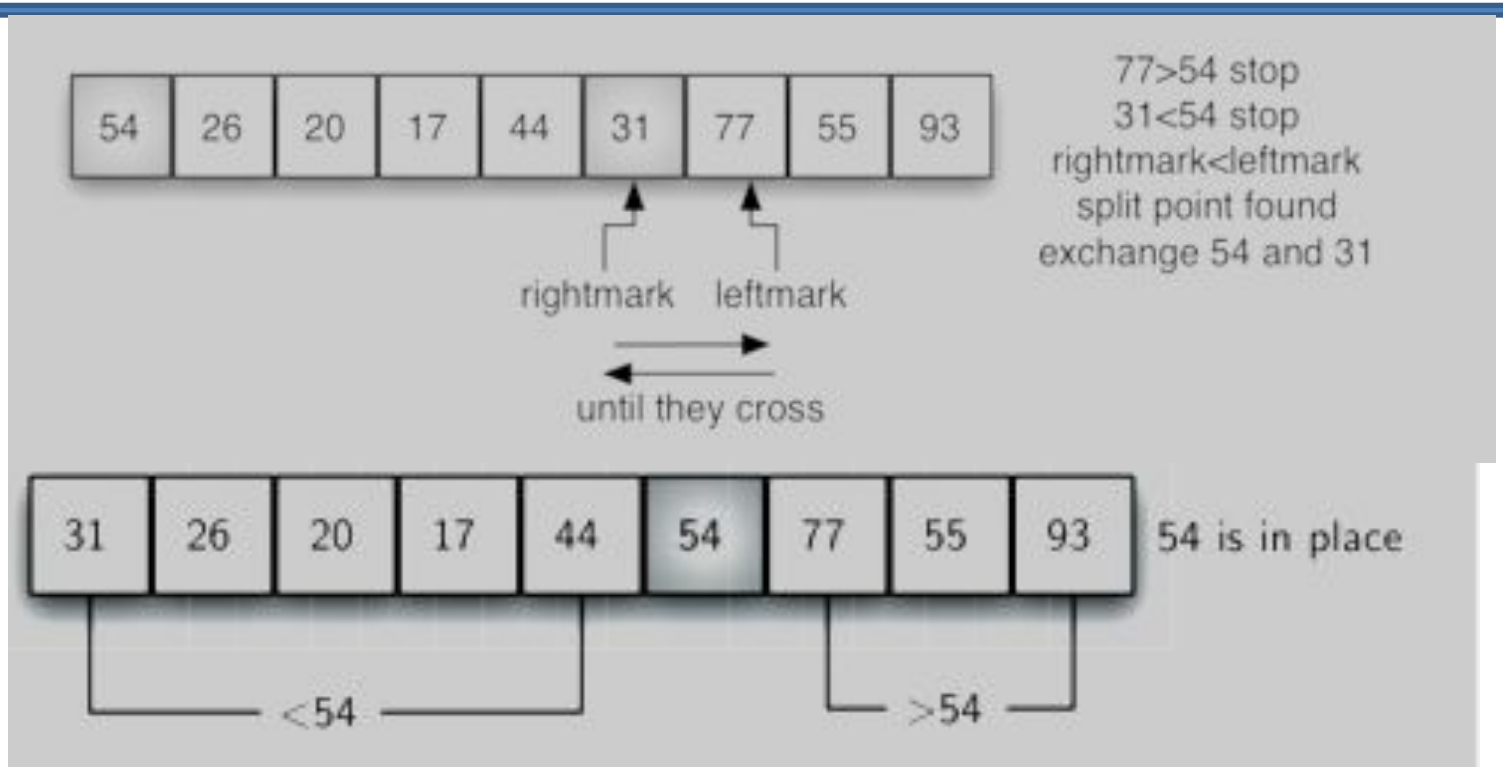


Quick Sort Example

Quick Sort Example



Quick Sort Example



Quick Sort

Thank You.

Qu
rie
s?