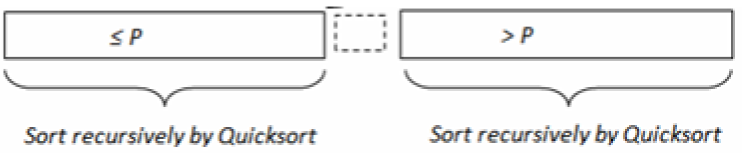
Quick Sort: [Best: O(nlogn), Worst:O(N^2)]

Basic idea

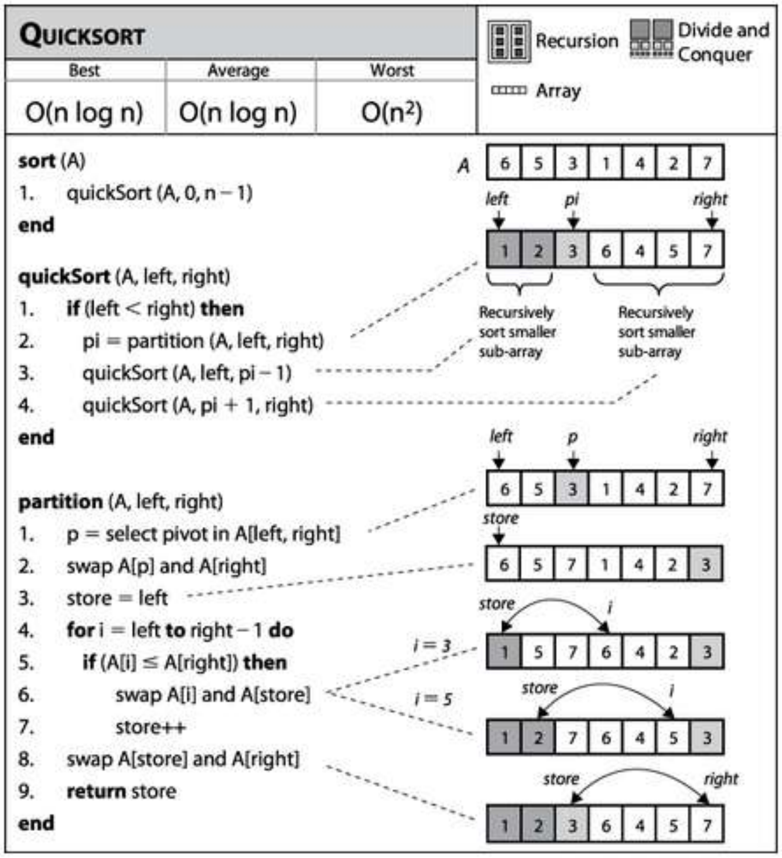
1. Pick one element in the array, which will be the pivot.
2. Make one pass through the array, called a partition step, re-arranging the entries so that:
   - the pivot is in its proper place.
   - entries smaller than the pivot are to the left of the pivot.
   - entries larger than the pivot are to its right.
3. Recursively apply quicksort to the part of the array that is to the left of the pivot, and to the right part of the array.



- In-place – Constant amount of extra memory.
- Pretty fast and efficient algorithm.
- Similar to mergesort - divide-and-conquer recursive algorithm
- One of the fastest sorting algorithms
- Average running time O(NlogN)
- Worst-case running time O(N2)

Algorithm:

```
algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p – 1)
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
    pivot := A[hi]
i := lo         // place for swapping
    for j := lo to hi – 1 do
        if A[j] ≤ pivot then
            swap A[i] with A[j]
i := i + 1
    swap A[i] with A[hi]
    return i
```

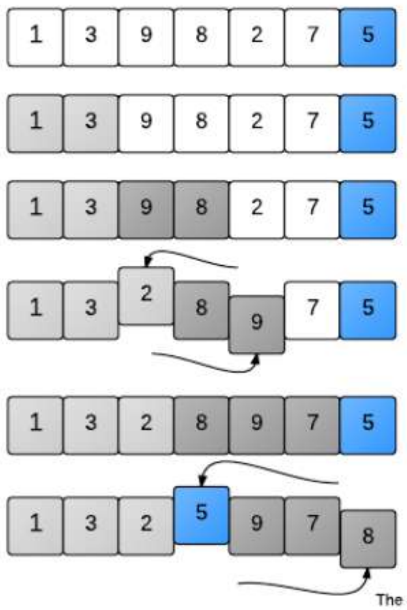


QUICKSORT

Best	Average	Worst
O(n log n)	O(n log n)	O(n²)

```
sort (A)
    quickSort (A, 0, n – 1)
end

quickSort (A, left, right)
    if (left < right) then
        pi = partition (A, left, right)
        quickSort (A, left, pi – 1)
        quickSort (A, pi + 1, right)
end

partition (A, left, right)
    p = select pivot in A[left, right]
    swap A[p] and A[right]
    store = left
    for i = left to right – 1 do
        if (A[i] ≤ A[right]) then
            swap A[i] and A[store]
            store++
    swap A[store] and A[right]
    return store
end
```

Partitioning an array



The Initial Array, where the pivot has been marked.

The first two elements are each compared with the pivot (and they are "swapped" with themself).

The next two element are greater than the pivot so they remain where they are.

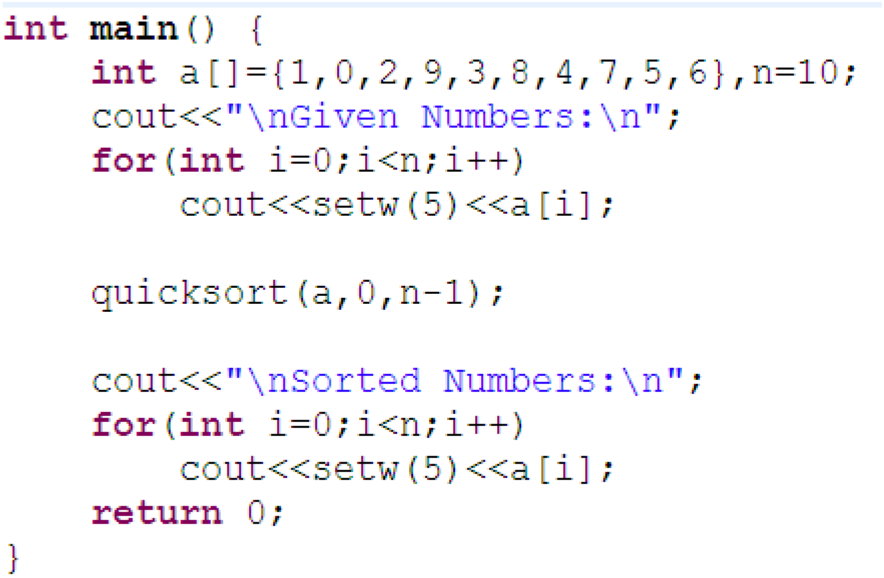
The 2 is smaller than the pivot, so it is swapped with the first element available.

The 7 is larger, so it remains where it is.

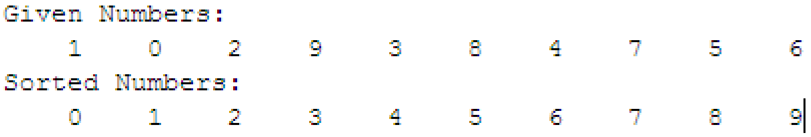
Finally, the pivot is swapped into the correct location.

The array has now been partitioned, and the index of the pivot can be returned.

```
// Quick Sort - Divide & Conquer - Recursive O(nlogn)
#include <iostream>
#include <iomanip>
using namespace std;
int partition(int a[],int start,int end)
{    int pivot=a[end];
     int pindex=start;
     for(int i=start; i<end; i++)
     {    if (a[i]<=pivot)
          {    swap(a[i],a[pindex]);
               pindex++;
          }
     }
     swap(a[pindex],a[end]);
     return pindex;
}

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



```
int main() {
    int a[]={1,0,2,9,3,8,4,7,5,6},n=10;
    cout<<"\nGiven Numbers:\n";
    for(int i=0;i<n;i++)
        cout<<setw(5)<<a[i];

    quicksort(a,0,n-1);

    cout<<"\nSorted Numbers:\n";
    for(int i=0;i<n;i++)
        cout<<setw(5)<<a[i];
    return 0;
}
```



```
Given Numbers:
    1    0    2    9    3    8    4    7    5    6
Sorted Numbers:
    0    1    2    3    4    5    6    7    8    9
```