QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. It is also called partition-exchange sort. This algorithm divides the list into three main parts:

1. Elements less than the Pivot element
2. Pivot element
3. Elements greater than the pivot element

Pivot element can be any element from the array, it can be the first element, the last element or any random element. The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

In the array {52, 37, 63, 14, 17, 8, 6, 25}, we take 25 as pivot. So after the first pass, the list will be changed like this.

{6 8 17 14 **25** 63 37 52}

Hence after the first pass, pivot will be set at its position, with all the elements smaller to it on its left and all the elements larger than to its right. Now 6 8 17 14 and 63 37 52 are considered as two separate sunarrays, and same recursive logic will be applied on them, and we will keep doing this until the complete array is sorted.

Following are the steps involved in quick sort algorithm:

1. After selecting an element as pivot, which is the last index of the array in our case, we divide the array for the first time.
2. In quick sort, we call this partitioning. It is not simple breaking down of array into 2 subarrays, but in case of partitioning, the array elements are so positioned that all the elements smaller than the pivot will be on the left side of the pivot and all the elements greater than the pivot will be on the right side of it.
3. And the pivot element will be at its final sorted position.
4. The elements to the left and right, may not be sorted.
5. Then we pick subarrays, elements on the left of pivot and elements on the right of pivot, and we perform partitioning on them by choosing a pivot in the subarrays.

int Partition (int arr[], int low, int high)

{

int pivot, i, j,t ;

pivot = arr[high];

i = (low - 1);

for (j = low; j <= (high - 1); j++)

{

if (arr[j] <= pivot)

{

i++;

t = arr[i];

arr[i] = arr[j];

arr[j] = t;

}

}

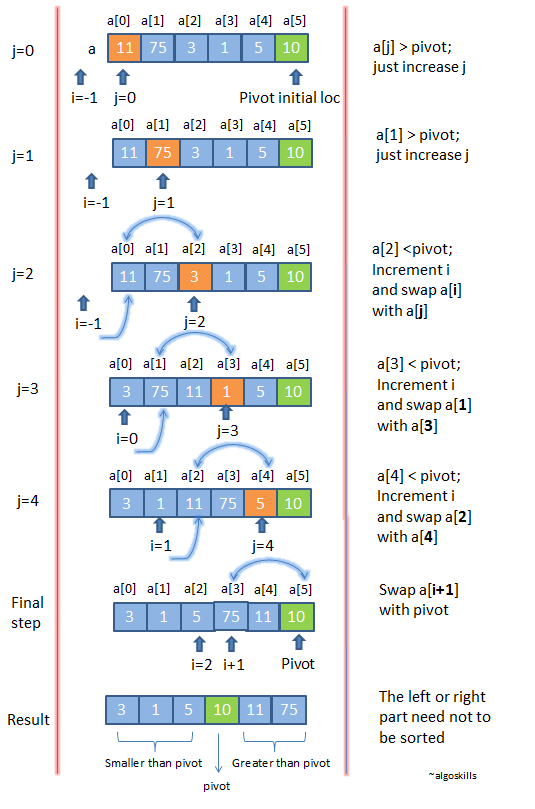
t = arr[i+1];

arr[i+1] = arr[high];

arr[i+1] = t;

return (i + 1);

}



arr[] = {10, 80, 30, 90, 40, 50, 70}

Indexes: 0 1 2 3 4 5 6

low = 0, high = 6, pivot = arr[h] = 70

Initialize index of smaller element, i = -1

Traverse elements from j = low to high-1

j = 0 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])

i = 0

arr[] = {10, 80, 30, 90, 40, 50, 70} // No change as i and j // are same

j = 1 : Since arr[j] > pivot, do nothing

// No change in i and arr[]

j = 2 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])

i = 1

arr[] = {10, 30, 80, 90, 40, 50, 70} // We swap 80 and 30

j = 3 : Since arr[j] > pivot, do nothing

// No change in i and arr[]

j = 4 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])

i = 2

arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 and 40 Swapped

j = 5 : Since arr[j] <= pivot, do i++ and swap arr[i] with arr[j]

i = 3

arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 and 50 Swapped

We come out of loop because j is now equal to high-1.

Finally we place pivot at correct position by swapping

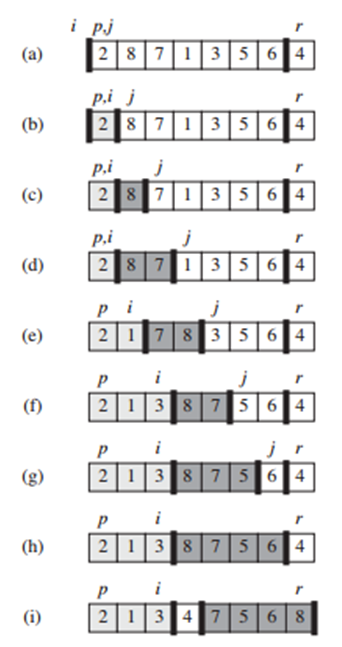
arr[i+1] and arr[high] (or pivot)

arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 and 70 Swapped

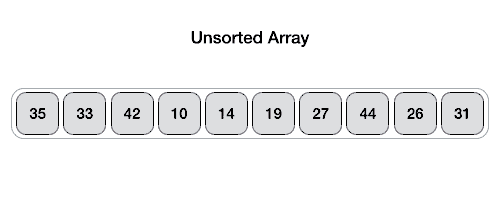
Now 70 is at its correct place. All elements smaller than

70 are before it and all elements greater than 70 are after

it.



How Quick Sort algorithm works

[](file:///C:\Users\hp\Desktop\quick_sort_partition_animation.gif)

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

int pivot = partition(arr, low, high);

quickSort(arr, low, pivot - 1);

quickSort(arr, pivot + 1, high);

}

}

int main()

{

int arr[100],n,i;

printf("Enter Element of the array");

scanf("%d",&n)

for(i=0; i<n; i++)

scanf("%d",&arr[i]);

quickSort(arr, 0, n - 1);

printf( "Sorted array: \n");

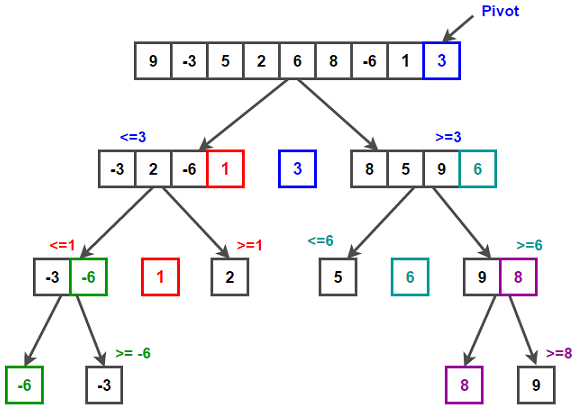
for(i=0; i<n; i++)

printf("%d\t",arr[i]);

return 0;

}

Recursion Example



low = 7,

high = 8

low = 0,

high = 1

low = 0, high = 3

low = 5, high = 8

low = 0, high = 8

quicksort(A, 8, 8);

partition(A, 5, 8);

quicksort(A,0,8)

Partition(A, 0, 1);

quicksort(A, 0, -1);

partition(A, 7, 8);

quicksort(A, 7, 6);

quiclsort(A, 1, 1);

quicksort(A, 7, 8);

quicksort(A, 5, 5);

quicksort(A, 3,3);

quicksort(A, 0, 1);

quicksort(A, 0, 3);

Partition(A, 0, 3);

quicksort(A, 5, 8);

partition(A, 0, 8);

**Best Case O(n logn)**

**Average Case O(n logn)**

**Worst Case O(n^2)**