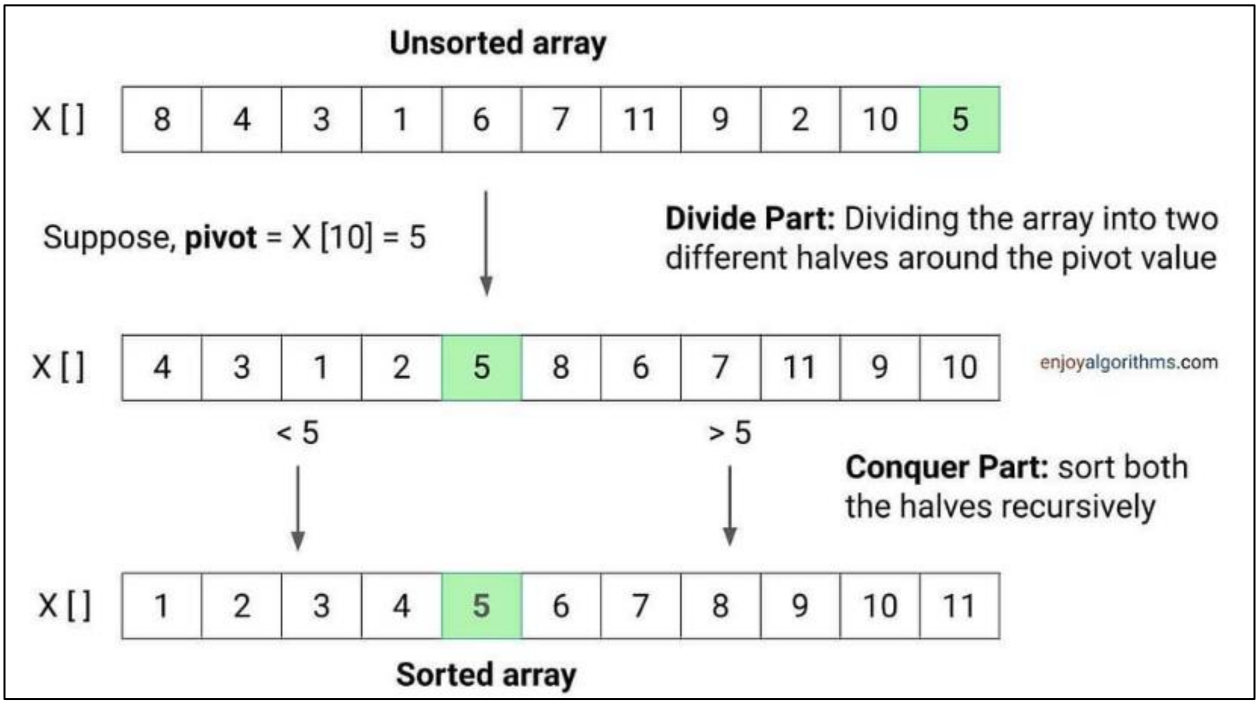
ADVANCE SORTING TECHNIQUE: QUICK SORT

Quick Sort is a highly efficient sorting algorithm that follows the Divide and Conquer principle. It works by selecting a "pivot" element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively.

KEY STEPS IN QUICK SORT:

1. **Choose a Pivot:** Select an element from the array as the pivot (various strategies can be used for selecting the pivot, such as the first element, the last element, or the median).
2. **Partitioning:** Rearrange the array such that elements less than the pivot are on its left, and elements greater than the pivot are on its right.
3. **Recursion:** Recursively apply the same process to the left and right sub-arrays.


Unsorted array
X [] 8 4 3 1 6 7 11 9 2 10 5
Suppose, **pivot** = X [10] = 5
Divide Part: Dividing the array into two different halves around the pivot value
X [] 4 3 1 2 5 8 6 7 11 9 10
enjoyalgorithms.com
< 5
> 5
Conquer Part: sort both the halves recursively
X [] 1 2 3 4 5 6 7 8 9 10 11
Sorted array

TIME COMPLEXITY:

Worst Case: O(n²) (This occurs when the smallest or largest element is always chosen as the pivot.)

ADVANTAGES OF QUICK SORT

- Fast: On average, it has a time complexity of O(n log n).
- In-Place: It requires little additional memory.
- Efficient: It performs well on average and is often faster than other O(n log n) algorithms.

DISADVANTAGES OF QUICK SORT:

- Worst Case Performance: Its worst-case time complexity is $O(n^2)$, which can happen when the pivot is consistently chosen poorly.
- Not Stable: It does not preserve the relative order of equal elements.

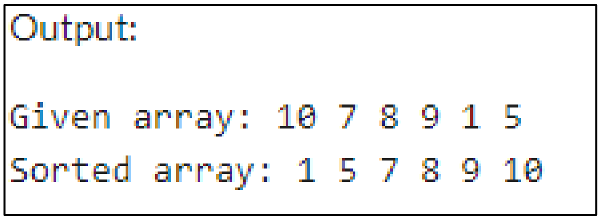
KEY FEATURES OF QUICK SORT:

- In-Place Sorting: Quick sort is an in-place sorting algorithm, meaning it requires only a small, constant amount of additional storage space.
- Efficient for Large Datasets: It is generally faster than other O(n log n) algorithms like merge sort and heap sort, especially for large datasets, due to better cache performance and fewer data movements.
- Not Stable: Quick sort is not a stable sort, meaning it can change the relative order of equal elements.

CODE

```
#include <iostream>
using namespace std;

// Function to swap two elements
void swap(int& a, int& b) {
   int temp = a;
   a = b;
   b = temp;
}
```

```
// Function to partition the array
int partition(int arr[], int low, int high) {
    // Choosing the rightmost element as pivot
    int pivot = arr[high];
    int i = (low - 1); // Index of the smaller element

    // Rearranging elements based on the pivot
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++; // Increment index of smaller element
            swap(arr[i], arr[j]); // Swap current element with the smaller element
        }
    }
    swap(arr[i + 1], arr[high]); // Place the pivot element in the correct position
    return (i + 1); // Return the partitioning index
}

// Function to perform quick sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // Partitioning index
        int pi = partition(arr, low, high);

        // Recursively sort elements before and after partitioning
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

// Helper function to print the array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Main function to test quick sort
int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int arrSize = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array: ";
    printArray(arr, arrSize);

    quickSort(arr, 0, arrSize - 1);

    cout << "Sorted array: ";
    printArray(arr, arrSize);

    return 0;
}
```



Output:

```
Given array: 10 7 8 9 1 5
Sorted array: 1 5 7 8 9 10
```