# Hello Everyone

## Course Code : CSE201
## Course Title : Data Structure


**Course Teacher:**

Md. Jamal Uddin

Assistant Professor

Department of CSE,BSMRSTU,

Gopalganj-8100.

**Presented By:**

Tushar Sarkar

Student ID: 18CSE035

Department of CSE,BSMRSTU,

Gopalganj-8100.

# Quick Sort

# Outline

- Introduction to Sort
- What is Quick Sort?
- Algorithm Explain
- Example of Quick Sort
- Pseudo Code
- Complexity
- Advantages & Disadvantages
- Conclusion

# Introduction to Sort

- **Sorting** is one of the most basic functions applied to data.
- Sorting is a technique that is implemented to arrange the data in a specific order.
- It means arranging the data in a particular fashion, which can be increasing or decreasing.
- Sorting is required to ensure that the data which we use is in a particular order so that we can easily retrieve the required piece of information from the pile of data.
- If the data is unkempt and unsorted, when we want a particular piece of information, then we will have to search it one by one every time to retrieve the data.

# What is Quick Sort?

- Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays.
- A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.
- Quicksort partitions an array and then calls itself recursively twice to sort the two resulting subarrays.

# Algorithm Explain

**Step 1** – Choose the highest index value has 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

**Step 5** – while value at left is less than pivot move right

**Step 6** – while value at right is 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 ≥ right, the point where they met is new pivot

# Example of Quick Sort

| | | | | |
|---|---|---|---|---|
| If primary array is : | 3 | 2 | 5 | 4 |
| Sorting By Ascending Order : | 2 | 3 | 4 | 5 |
| Sorting By descending Order : | 5 | 4 | 3 | 2 |

# Example Explain

Step 1:

pivot=3, right=4

pivot < right, right shift

Step 2:

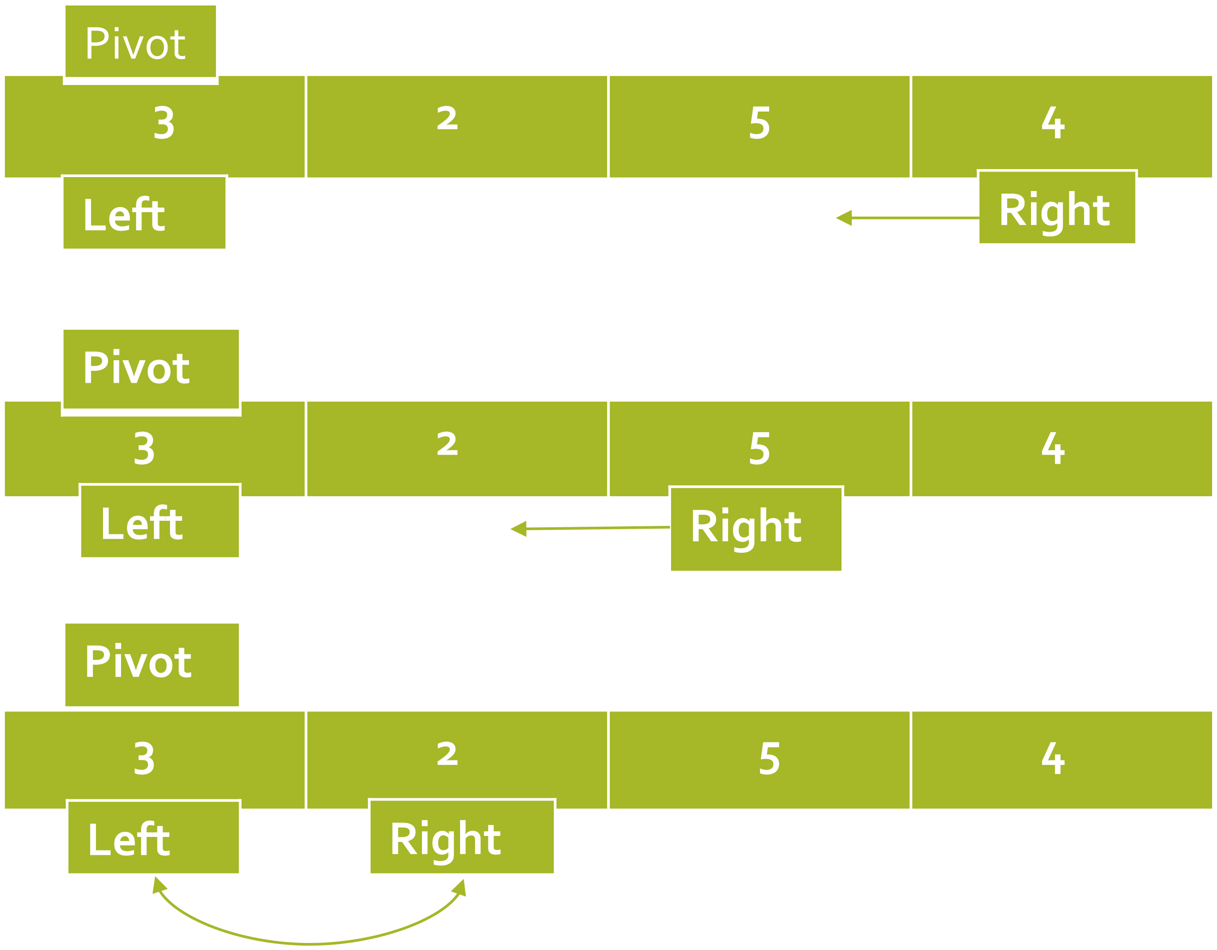
pivot=3, right=5

pivot < right, right shift

Step 3:

pivot=3, right=2

pivot > right, swap(pivot,right)



# Example Explain

Step 4:

pivot=3, left=2

pivot > left, left shift

Step 5:

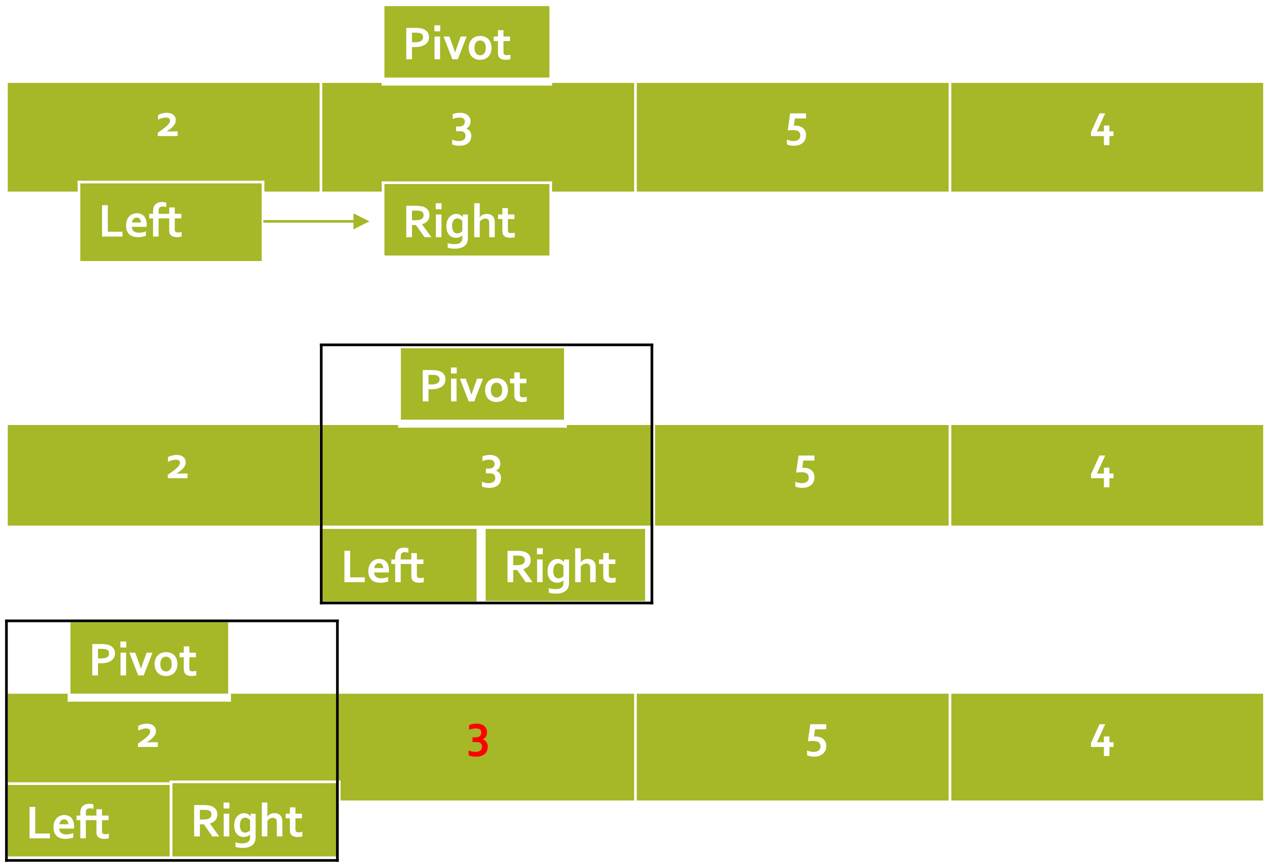
pivot=left=right=3

So 3 is sorted

Step 6:

pivot=left=right=2

So 2 is fixed



# Example Explain

Step 7:

pivot=5, right=4

pivot > right, swap(pivot,right)


Step 8:

pivot=5, left=4

pivot > left, left shift


Step 9:

pivot=left=right=5

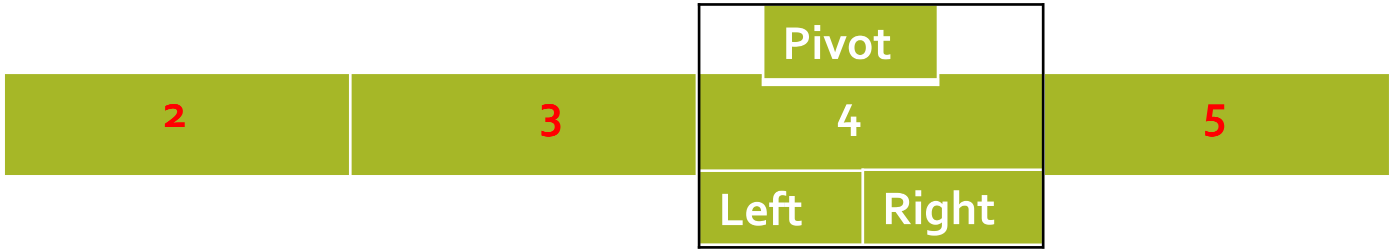
So 5 is sorted

# Example Explain

Step 10:

pivot=left=right=4

So 4 is sorted




Step 11:

Array is sorted

# Pseudo Code

```
int partition (int arr[], int low, int high){
    int i,j,pivot;

    pivot = arr[high];     // pivot
    i = (low - 1);  // Index of smaller element

    for (j = low; j <= high- 1; j++){
        if (arr[j] <= pivot)
        {
            i++;    // increment index of smaller element
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);

    return (i + 1);
}

void quickSort(int arr[], int low, int high)
{
    int pi;
    if (low < high)
    {
        pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main()
{
```

C:\WINDOWS\system32\cmd.exe - pause

```
Input  : 3 2 5 4
Output : 2 3 4 5
Press any key to continue . . .
```

# Complexity

## Best case :

- The best-case occurs the algorithm is conducted in such a way that always the median element is selected as the pivot and thus reduces the complexity.
- The following time is taken for the best case.

  $T(n)=2T(n/2)+n;$

- The solution of the above recurrence is O(*nlogn*).
- It can be solved using Master Theorem.
- So the best case of this algorithm is *nlogn* where n is the size the array.

# Complexity

**Worst Case :**

- The proposed algorithm gives a better running time than a classical quick sort algorithm.
- In this case, we go for a manual sort where we compare two elements normally.
- There might be a situation where a worst-case partitioning will be required.
- Thus, the time taken for the proposed algorithm is:

$$T(n)=T(8n/10)+T(2n/10)+cn$$

- The total time taken becomes O(nlogn).

# Advantages of Quick Sort

- ✓It is in-place since it uses only a small auxiliary stack.
- ✓It requires only n (log n) time to **sort** n items.
- ✓It has an extremely short inner loop.
- ✓This algorithm has been subjected to a thorough mathematical analysis, a very precise statement can be made about performance issues.

# Disadvantages of Quick Sort

- ✓It is recursive. Especially, if recursion is not available, the implementation is extremely complicated.
- ✓It requires quadratic (i.e., n2) time in the worst-case.
- ✓It is fragile, i.e. a simple mistake in the implementation can go unnoticed and cause it to perform badly.

# Conclusion

- **Quicksort** turns out to be the fastest **sorting** algorithm in practice.
- It has a time complexity of $\Theta( n \log( n ))$ on the average.
- However, in the (very rare) worst case **quicksort** is as slow as Bubblesort, namely in $\Theta( n^2)$.

Thank You