

Analysis of the Quick Sort Algorithm

- **Code :**

```
import java.util.*;

public class Quicksort {

    public void q1(int arr[], int low, int high) {

        if (low < high) {

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

            q1(arr,low,pi - 1);

            q1(arr,pi + 1,high);

        }

    }

    public 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++;

                int temp = arr[i];

                arr[i] = arr[j];

                arr[j] = temp;

            }

        }

        int temp = arr[i + 1];

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

        arr[high] = temp;

    }

}
```

```

        return (i + 1);
    }

    public static void main(String[] args) {
        int arr[] = {45,21,6,129,234,5,2,1};

        System.out.println("Before Sorting : ");

        for (int i = 0; i < arr.length; i++)
            System.out.print(arr[i] + " ");

        Quicksort qs = new Quicksort();

        qs.q1(arr, 0, arr.length - 1);

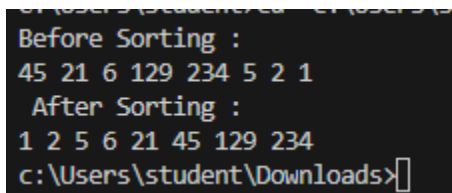
        System.out.println("\n After Sorting : ");

        for (int i = 0; i < arr.length; i++)
            System.out.print(arr[i] + " ");

    }
}

```

- **Output :**



```

Before Sorting :
45 21 6 129 234 5 2 1
After Sorting :
1 2 5 6 21 45 129 234
c:\Users\student\Downloads>

```

- **Overview of Quick Sort**

Quick Sort is a divide-and-conquer algorithm that sorts an array by selecting a 'pivot' element 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 Components of the Code**

1. Partition Function:

- Input: An array (`arr`), and indices (`low` and `high`).
- Process: Selects the last element as the pivot.
 - Rearranges the array so that all elements less than or equal to the pivot are on its left, and all elements greater than it are on its right.
- Returns the index of the pivot after partitioning.

2. Recursive Quick Sort Function (`q1`):

- Input: The same array (`arr`), along with `low` and `high` indices.
- Process:
 - Calls itself recursively on the left and right sub-arrays formed by the pivot index returned from the partition function.
- Base Case: The recursion stops when `low` is not less than `high`.

3. Main Method:

- Initializes an array, prints it before sorting, calls the Quick Sort method, and then prints the sorted array.

- **Time Complexity Analysis**

1. Best Case: $O(n \log n)$

- Occurs when the pivot divides the array into two equal halves consistently. This leads to $\log n$ levels of recursion, with each level taking linear time to process.

2. Average Case: $O(n \log n)$

- On average, Quick Sort performs well due to random distribution of elements.

3. Worst Case: $O(n^2)$

- This occurs when the smallest or largest element is always chosen as the pivot (e.g., already sorted arrays). In such cases, one side of the partition will always be empty, leading to n levels of recursion with each level taking linear time.

- **Space Complexity**

- The space complexity of Quick Sort is $O(\log n)$ due to recursive stack space in case of balanced partitions. However, in worst-case scenarios (unbalanced partitions), it can go up to $O(n)$.