Quick Sort Algorithm

- Pivot: Any Reference Variable
- After first pass:
  - all the elements < pivot will be on left side
  - all the elements > pivot will be on right side

Last Pivot Sort

Steps:

1. (5, 3, 2, 1, 6)
2. (5, 3, 2, 1) + (6)
3. (1, 2, 3, 5) + (6)
4. (1, 2, 3) + (5) + (6)
5. (1, 2) + (3) + (5) + (6)
6. (1) + (2) + (3) + (5) + (6)
7. (1, 2, 3, 5, 6)

```java
package com.inclass;

import java.util.Arrays;

public class QuickSort {

    public static void main(String[] args) {
        int[] arr = new int[] {5, 3, 2, 1, 6};
        endPivot(arr, 0, arr.length - 1);
        System.out.println(Arrays.toString(arr));
    }

    static void endPivot (int[] arr, int start, int end) {
        if (start ≥ end) {
            return;
        }
        int startTemp = start;
        int pivot = end;
        while (start < pivot) {
            if (arr[start] > arr[pivot]) {
                int temp = arr[start];
                for (int i = start; i < pivot; i++) {
                    arr[i] = arr[i + 1];
                }
                arr[pivot--] = temp;
            } else {
                start++;
            }
        }
        print(arr, startTemp, end);
        endPivot(arr, startTemp, pivot - 1);
        endPivot(arr, pivot, end);
    }

    static void print (int[] arr, int start, int end) {
```

```
        System.out.print("[");
        for (int i = start; i < end; i++) {
            System.out.print(arr[i] + ", ");
        }
        System.out.println(arr[end] + "]");
    }
}
```

First Pivot Sort

Steps:

1. (5, 3, 2, 1, 6)
2. (1, 2, 3, 5) + (6)
3. (1) + (2, 3, 5) + (6)
4. (1) + (2) + (3, 5) + (6)
5. (1) + (2) + (3) + (5) + (6)
6. (1, 2, 3, 5, 6)

```
package com.inclass;

import java.util.Arrays;

public class QuickSort {

    public static void main(String[] args) {
        int[] arr = new int[] {5, 3, 2, 1, 6};
        startPivot(arr, 0, arr.length - 1);
        System.out.println(Arrays.toString(arr));
    }

    static void startPivot (int[] arr ,int start, int end) {
        if (start ≥ end) {
            return;
        }
        int endTemp = end;
        int pivot = start;
        while (end > pivot) {
            if (arr[end] < arr[pivot]) {
                int temp = arr[end];
                for (int i = end - 1; i ≥ pivot; i--) {
                    arr[i + 1] = arr[i];
                }
                arr[pivot++] = temp;
            } else {
                end--;
            }
        }
        print(arr, start, endTemp);
        startPivot(arr, start, pivot);
        startPivot(arr, pivot + 1, endTemp);
    }

    static void print (int[] arr, int start, int end) {
        System.out.print("[");
        for (int i = start; i < end; i++) {
            System.out.print(arr[i] + ", ");
```

```
        }
        System.out.println(arr[end] + "]");
    }
}
```

Middle Pivot Sort

Steps:

1. (2, 3, 5, 1, 6)
2. (1) + (2, 3, 5, 6)
3. (1) + (2, 3) + (5, 6)
4. (1) + (2) + (3) + (5) + (6)
5. (1, 2, 3, 5, 6)

```
package com.inclass;

import java.util.Arrays;

public class QuickSort {

    public static void main(String[] args) {
        int[] arr = new int[] {5, 3, 2, 1, 6};
        midPivot(arr, 0, arr.length - 1);
        System.out.println(Arrays.toString(arr));
    }

    static void midPivot (int[] arr, int start, int end) {
        if (start ≥ end) {
            return;
        }
        int startTemp = start;
        int endTemp = end;
        int pivot = start + (end - start + 1) / 2;
        while (start < pivot) {
            if (arr[start] > arr[pivot]) {
                int temp = arr[start];
                for (int i = start; i < pivot; i++) {
                    arr[i] = arr[i + 1];
                }
                arr[pivot--] = temp;
            } else {
                start++;
            }
        }
        while (end > pivot) {
            if (arr[end] < arr[pivot]) {
                int temp = arr[end];
                for (int i = end - 1; i ≥ pivot; i--) {
                    arr[i + 1] = arr[i];
                }
                arr[pivot++] = temp;
            } else {
                end--;
            }
        }
        print(arr, startTemp, endTemp);
```

```
        midPivot(arr, startTemp, pivot - 1);
        midPivot(arr, pivot, endTemp);
    }

    static void print (int[] arr, int start, int end) {
        System.out.print("[");
        for (int i = start; i < end; i++) {
            System.out.print(arr[i] + ", ");
        }
        System.out.println(arr[end] + "]");
    }
}
```

Recurrence Relation:

$$T[n] = T[k] + T[n-k-1] + O(n)$$

- Worst Case (k = 0):

  when $n \to \infty$, $O(n)$ is less dominating,

  $\Rightarrow f(n) = f(n-1) \Rightarrow f(n) = (n-1) + f(n-2) \Rightarrow f(n) = (n-1) + (n-2) + (n-3) + \ldots \Rightarrow f(n) = \frac{n(n-1)}{2} \Rightarrow f$

- Best Case (k= $\frac{n}{2}$):

  $$T[n] = T[\frac{n}{2}] + T[\frac{n}{2}] + [n-1] = 2T[\frac{n}{2}] + [n-1]$$

  Using Akra-Bazzi to find complexity: Finding p,

  $$a_1 b_1^p = 1 \ \Rightarrow 2 * \frac{1}{2} = 1 \therefore p = 1$$

  $$T(x) = \theta(x^P + x^p \int_1^x \frac{g(u)du)}{u^{p+1}} = \theta(x + x \int_1^x \frac{(u-1)du}{u^2} = \theta(x + x \int_1^x \frac{1}{u}du - x \int_1^x \frac{1}{u^2}du = \theta(x + xlog(x) - x + 1)$$