Merge Sort Algorithm

Steps:

1. Divide Array into 2 parts
2. Sort both parts via Recursion
3. Merge the sorted parts

- Dividing Arrays:
  1. {5, 3, 2, 1, 4}
  2. {5, 3} + {2, 1, 4}
  3. {5} + {3} + {2} + {1, 4}
  4. {5} + {3} + {2} + {1} + {4}
- Merging Arrays:
  1. {5} + {3} + {2} + {1} + {4}
  2. {5} + {3} + {2} + {1, 4}
  3. {3, 5} + {1, 2, 4}
  4. {1, 2, 3, 4, 5}

Not-In-Place Merge Sort

```java
package com.inclass;

import java.util.Arrays;

public class MergeSort {
    public static void main(String[] args) {
        int[] arr = {7, 9 ,1, 3, 5, 2, 4};
        System.out.println(Arrays.toString(mergeSort(arr)));
    }

    static int[] mergeSort (int[] arr) {
        if (arr.length == 1) {
            return arr;
        }

        int mid = arr.length / 2;

        int[] left = mergeSort(Arrays.copyOfRange(arr, 0, mid));
        int[] right = mergeSort(Arrays.copyOfRange(arr, mid, arr.length));

        return merge(left, right);
    }

    static int[] merge (int[] left, int[] right) {
        int[] mix = new int[left.length + right.length];

        int i = 0;
        int j = 0;

        while (i < left.length && j < right.length) {
            if (left[i] < right[j]) {
                mix[i + j] = left[i];
                i++;
```

```
            } else {
                mix[i + j] = right[j];
                j++;
            }
        }

        // it may be possible that one of the arrays is not complete
        // copy the remaining elements
        while (i < left.length) {
            mix[i + j] = left[i];
            i++;
        }

        while (j < right.length) {
            mix[i + j] = right[j];
            j++;
        }

        return mix;
    }
}
```

Time Complexity Analysis:

Total Number of Levels = $N/2^k$

$\Rightarrow 1 = N/2^k \Rightarrow 2^k = N \Rightarrow k = log_2 N$

$$\therefore O[N * log_2 N]$$

Space Complexity: O(N)

Recurrence Relation:

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

Using Akra-Bazzi to find complexity:

Finding p,

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

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

In-Place Merge Sort

```
package com.inclass;

import java.util.Arrays;

public class InplaceMergeSort {
    public static void main(String[] args) {
        int[] arr = {5, 4, 1, 2, 3};
        mergeSort(arr, 0 , arr.length - 1);
        System.out.println(Arrays.toString(arr));
    }
```

```
    static void mergeSort (int[] arr, int start, int end) {
        if (start ≥ end) {
            return;
        }
        int mid = start + (end - start + 1) / 2;
        mergeSort(arr, start, mid - 1);
        mergeSort(arr, mid, end);
        merge(arr, start, end);
    }

    static void merge (int[] arr, int start, int end) {
        int mid = start + (end - start + 1) / 2;
        while (start < mid && mid ≤ end) {
            if (arr[start] > arr[mid]) {
                for (int i = mid - 1; i ≥ start; i--) {
                    int temp = arr[i + 1];
                    arr[i + 1] = arr[i];
                    arr[i] = temp;
                }
                start++;
                mid++;
            } else {
                start++;
            }
        }
    }
}
```