Algorithms & Code (Java)

1. Search

1.1. Binary Search

- For sorted arrays
- O(lg(n))

```
// Refactored from: https://www.geeksforgeeks.org/binary-search/

public static int binsearch(int[] arr, int val) {
    int left = 0, right = arr.length - 1, mid = 0;
    while (right >= left) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == val) {      return mid; }
        else if (arr[mid] > val) {  right = mid - 1; }
        else {                      left = mid + 1; }
    }
    return -1;
}
```

2. Sorting

2.1. Merge Sort

- Time & Space Complexity: O(nlog(n))
- Not in-place
- Stable
- Comparison based sorting algorithm
- Extra Notes:
    - Can be used to count inversions
    - Example of divide & conquer

```
// Refactored from: https://www.geeksforgeeks.org/java-program-for-
merge-sort/
```

```
public static void merge(int[] arr, int l, int m, int r) {
    // Find sizes of two subarrays to be merged
    int L_N = m - l + 1;
    int R_N = r - m;

    /* Create temp arrays */
    int L[] = new int [L_N];
    int R[] = new int [R_N];

    /*Copy data to temp arrays*/
    for (int i=0; i<L_N; ++i) L[i] = arr[l + i];
    for (int j=0; j<R_N; ++j) R[j] = arr[m + 1 + j];

    /* Merge the sorted sub-arrays */
    int i = 0, j = 0;
    int k = l;      // Index of current sub-arr
    while (i < L_N && j < R_N) {
        if (L[i] <= R[j])   { arr[k++] = L[i++]; }
        else                { arr[k++] = R[j++]; }
    }

    /* Copy remaining elements of L[] and R[] if any */
    while (i < n1) { arr[k++] = L[i++]; }
    while (j < n2) { arr[k++] = R[j++]; }
}

// Usage: merge_sort(arr, 0, arr.length - 1);
public static void merge_sort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        merge_sort(arr, left, mid);
        merge_sort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
```

2.2. Heap Sort

2.3. Quick Sort

```
public static void swap(int arr[], int i, int j) {
    int tmp = arr[i];
    arr[i] = arr[j];
```

```
    arr[j] = tmp;
}

public static int partition(int arr[], int l, int r, int pivot) {
    int i = l; // index of smaller element
    for (int j = l; j < r; ++j) {
        if (arr[j] <= pivot)
            swap(arr, i++, j);
    }

    // swap arr[i] with pivot at arr[r]
    swap(arr, i, r);
    return i;
}

// Usage: quickSort(arr, 0, arr.length - 1);
public static void quickSort(int arr[], int l, int r) {
    if (l < r) {
        // Swap random pivot to right
        swap(arr, rand.nextInt(r - l) + l, r);

        // Get pivot index
        int pivotIdx = partition(arr, l, r, arr[r]);

        // Recursively sort elements on left & right
        quickSort(arr, l, pivotIdx - 1);
        quickSort(arr, pivotIdx + 1, r);
    }
}
```

2.4. Counting Sort

2.5. Radix Sort

2.6. Bucket Sort