Q1.

1] Stable(Merge Sort):-

```c
#include <stdio.h>
#include <stdlib.h>

#define SIZE 10

void mergelist(int a[], int lb1, int ub1, int lb2, int ub2);

void mergesort(int a[], int low, int high) {
    if (low == high)
        return;
    else {
        int mid = (low + high) / 2;
        mergesort(a, low, mid);     // left subarray
        mergesort(a, mid + 1, high); // right subarray
        mergelist(a, low, mid, mid + 1, high);
    }
}

void mergelist(int a[], int lb1, int ub1, int lb2, int ub2) {
    int i = lb1;
    int j = lb2;
    int k = 0;
    int c[SIZE];

    while (i <= ub1 && j <= ub2) {
        if (a[i] < a[j])
            c[k++] = a[i++];
        else
            c[k++] = a[j++];
    }

    while (j <= ub2) {
        c[k++] = a[j++];
    }
    while (i <= ub1)
        c[k++] = a[i++];

    for (i = lb1, k = 0; i <= ub2; i++, k++)
        a[i] = c[k];
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
```

```
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    for (int i = 0; i < arr_size; i++)
        printf("%d ", arr[i]);
    printf("\n");

    mergesort(arr, 0, arr_size - 1);

    printf("Sorted array is \n");
    for (int i = 0; i < arr_size; i++)
        printf("%d ", arr[i]);
    printf("\n");

    return 0;
}
```

```
Given array is
12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\AoA>
```

2] Fast(Quick Sort):-

```
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

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

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i - 1], &arr[j]);
        }
    }

    swap(&arr[i], &arr[high]);
    return i;
}
```

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

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {1,45,34,87,56,23};
    int arr_size = 6;

    printf("Given array is \n");
    printArray(arr, arr_size);

    quickSort(arr, 0, arr_size - 1);

    printf("Sorted array is \n");
    printArray(arr, arr_size);

    return 0;
}
```

```
Given array is
1 45 34 87 56 23
Sorted array is
1 23 34 45 56 87
PS C:\Users\Mark Lopes\Desktop\college\Sem_4\AoA>
```

Postlab:-

Mark Lopes
9913

FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

5.E comps A Batch-C Lab2 postlab

Q.1 Complexity of merge sorts

→ In merge sort, the array is recursively sorted into two halves at the same time.

$\therefore T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

By Master method,

$f(n) = n$

$a = 2, b = 2$

$n^{\log_b a} = n^{\log_2 2} = n$

$\therefore f(n) = n^{\log_b a}$

$\therefore T(n) = f(n) \times \log n$

$\boxed{T(n) = n \log_2 n}$

Q.2 derive complexity of quick sort for all cases.

→ 1. Worst case ~~Average case~~

This case occurs when, the array is already sorted or nearly sorted.

$\therefore T(n) = T(n-1) + O(n) \quad \cdots T(n) = 1 \text{ when } n = 1$

$= T(n-2) + (n-1) + n$

$= T(n-3) + (n-2) + (n-1) + n$

$= T(n-k) + (n-(k-1)) + (n-(k-2)) + \cdots n$

~~$= T(1) + (n-$~~

$\therefore n-k = 1 \quad \therefore \boxed{k = n-1}$

$\therefore T(n) = T(1) + (n-(n-1-1)) + (n-(n-1-2)) + \dots n$

$= T(1) + (2) + 3 + 4 \dots n$

$= 1 + 2 + 3 + \dots n$

$= \frac{n(n+1)}{2}$

$\therefore \boxed{T(n) = O(n^2)}$

2. Best case

→ This case occurs when the pivot element constantly divides the array into two equal halves.

$\therefore T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

By Master method

$f(n) = n \;,\; a = 2, b = 2$

$n^{\log_b a} = n^{\log_2 2} = n$

$\therefore f(n) = n^{\log_b a}$

$\therefore T(n) = f(n) \times \log_2 n$

$\boxed{T(n) = n \log_2 n}$

Q.3 What changes will you do to make quick sort to randomised quick sort?

→ * Select Random pivot element:-

Rather than choosing a first or last element as pivot element, choose a random element from the array. This helps achieve a more balanced partition.

* Randomised quick sort helps mitigate the risk of worst case scenarios.