ALGORITMOS SEMANA 5

1)

```
int* quickSort(int arr_count, int* arr, int* result_count) {
    *result_count = arr_count;
    int *left = malloc(arr_count * sizeof(int));
    int *equal = malloc(arr_count * sizeof(int));
    int *right = malloc(arr_count * sizeof(int));

    //Storing the pivot
    int pivot = arr[0];

    //Dividing
    int index_left = 0, index_equal = 0, index_right = 0;
    for (int i = 0; i < arr_count; i++)
    {
        if (arr[i] < pivot)
            left[index_left++] = arr[i];
        else if (arr[i] == pivot)
            equal[index_equal++] = arr[i];
        else
            right[index_right++] = arr[i];

}
    //Group all together
    int *a = malloc(arr_count * sizeof(int));
    memcpy(a, left, index_left * sizeof(int));
    memcpy(a + index_left, equal, index_equal * sizeof(int));
    memcpy(a + index_left + index_equal, right, index_right *
sizeof(int));

    //Getting free of our Arrays
    free(left);
    free(equal);
    free(right);

    return a;
}
```

---

2)

```
int *quickSort(int arr_count, int *arr)
{
    //This condition is not necessary, but we get our
    //result faster than doing a new quicksort
    if (arr_count == 2)
    {
        if (arr[0] < arr[1])
            return arr;
        else
        {
            int b = arr[0];
            arr[0] = arr[1];
            arr[1] = b;
            return arr;
        }
    }

    int *left = malloc(arr_count * sizeof(int));
    int *equal = malloc(arr_count * sizeof(int));
    int *right = malloc(arr_count * sizeof(int));

    //Storing the pivot
    int pivot = arr[0];

    //Dividing
    int index_left = 0, index_equal = 0, index_right = 0;
    for (int i = 0; i < arr_count; i++)
    {
        if (arr[i] < pivot)
            left[index_left++] = arr[i];
        else if (arr[i] == pivot)
            equal[index_equal++] = arr[i];
        else
            right[index_right++] = arr[i];
    }

    if (index_left > 1)
        left = quickSort(index_left, left);

    if (index_right > 1)
```

```
        right = quickSort(index_right, right);

    //Group all together
    int *a = malloc(arr_count * sizeof(int));
    memcpy(a, left, index_left * sizeof(int));
    memcpy(a + index_left, equal, index_equal * sizeof(int));
    memcpy(a + index_left + index_equal, right, index_right *
sizeof(int));

    //Getting free of our Arrays
    free(left);
    free(equal);
    free(right);

    return a;
}

int findMedian(int arr_count, int *arr)
{
    int result_count;
    int *a = quickSort(arr_count, arr);
    return (arr_count % 2) == 1 ? a[arr_count / 2] : ((a[arr_count
/ 2] + a[(arr_count / 2) + 1]) / 2);
}
```

---

d1)

```
int *countingSort(int arr_count, int *arr, int *result_count)
{
    *result_count = 100;
    int *a = malloc(*result_count * sizeof(int));

    for (int i = 0; i < *result_count; i++)
        a[arr[i]]++;

    return a;
}
```

---

d2)

```
int *countingSort(int arr_count, int *arr, int *result_count)
{
    *result_count = arr_count;
    const int domain_array = 100;
    int *count = malloc(domain_array * sizeof(int));

    for (int i = 0; i < arr_count; i++)
        count[arr[i]]++;

    int *a = malloc(arr_count * sizeof(int));
    int index = 0;
    for (int i = 0; i < domain_array; i++)
    {
        for (int j = 0; j < count[i]; j++)
        {
            a[index] = i;
            index++;
        }
    }

    return a;
}
```