Bubble Sort

```
void BubbleSort(int a[], int array_size)
{
        int i, j, temp;
        for (i = 0; i < (array_size - 1); ++i)
        {
                for (j = 0; j < array_size - 1 - i; ++j)
                {
                        if (a[j] > a[j+1])
                        {
                                temp = a[j+1];
                                a[j+1] = a[j];
                                a[j] = temp;
                        }
                }
        }
}
```

Selection Sort

```
void SelectionSort(int a[], int array_size)
{
        int i;
        for (i = 0; i < array_size - 1; ++i)
        {
                int j, min, temp;
                min = i;
                for (j = i+1; j < array_size; ++j)
                {
                        if (a[j] < a[min])
                        min = j;
                }
```

```
            temp = a[i];
            a[i] = a[min];
            a[min] = temp;
        }
}
```

Insertion Sort

```
void insertionSort(int a[], int array_size)
{
        int i, j, index;
        for (i = 1; i < array_size; ++i)
        {
                index = a[i];
                for (j = i; j > 0 && a[j-1] > index; j--)
                        a[j] = a[j-1];
                a[j] = index;
        }
}
```

Merge Sort

```
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l+(r-l)/2;

        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
```

Quick Sort

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

int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

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