Exercício 1

```c
// Nome: Felipe B Castro
// RA: 2311292

#include <stdio.h>

void selectionSort(int arr[], int n) {
    for (int i = 0; i < n / 2; i++) {
        int min_index = i;
        int max_index = i;

        // Encontra o mínimo e o máximo dentro do intervalo [i, n - i - 1]
        for (int j = i + 1; j < n - i; j++) {
            if (arr[j] < arr[min_index])
                min_index = j;
            else if (arr[j] > arr[max_index])
                max_index = j;
        }
        // Troca o mínimo com o primeiro elemento não ordenado
        if (min_index != i) {
            int temp = arr[min_index];
            arr[min_index] = arr[i];
            arr[i] = temp;
        }
        // Se o máximo foi movido para a posição de i, atualiza o índice do máximo
        if (max_index == i)
            max_index = min_index;
        // Troca o máximo com o último elemento não ordenado
        int temp = arr[n - i - 1];
        arr[n - i - 1] = arr[max_index];
        arr[max_index] = temp;
    }
}

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

```c
int main() {
    int arr[] = {8, 6, 7, 4, 5, 3, 2, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Array original:\n");
    printArray(arr, n);

    selectionSort(arr, n);

    printf("Array ordenado:\n");
    printArray(arr, n);

    return 0;
}
```

```
Run

Array original:
[8][6][7][4][5][3][2][1]
Array ordenado:
[1][2][3][4][5][6][7][8]
```

Exercício 2

```c
#include <stdio.h>

// Bubble Sort
void bubbleSort(int arr[], int n) {
    int comparacoes = 0;
    int trocas = 0;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            comparacoes++;
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                trocas++;
            }
        }
    }
    printf("Bubble Sort:\n");
    printf("Comparacoes: %d\n", comparacoes);
    printf("Trocas: %d\n", trocas);
}

// Selection Sort
void selectionSort(int arr[], int n) {
    int comparacoes = 0;
    int trocas = 0;
    for (int i = 0; i < n - 1; i++) {
        int min_index = i;
        for (int j = i + 1; j < n; j++) {
            comparacoes++;
            if (arr[j] < arr[min_index]) {
                min_index = j;
            }
        }
        if (min_index != i) {
            int temp = arr[min_index];
            arr[min_index] = arr[i];
            arr[i] = temp;
            trocas++;
```

```c
        }
    }
    printf("\nSelection Sort:\n");
    printf("Comparacoes: %d\n", comparacoes);
    printf("Trocas: %d\n", trocas);
}

int main() {
    int arr[] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
    int n = sizeof(arr) / sizeof(arr[0]);

    int arr_copy[n]; // fazer uma cópia do vetor original
    for (int i = 0; i < n; i++) {
        arr_copy[i] = arr[i];
    }

    bubbleSort(arr_copy, n); // Ordena uma cópia do vetor original

    printf("Vetor desordenado:\n");
    for (int i = 0; i < n; i++) {
        printf("[%d]", arr[i]);
    }
    printf("\n");

    selectionSort(arr, n); // Selection Sort no vetor desordenado original

    return 0;
}
```

```
Bubble Sort:
Comparacoes: 45
Trocas: 45

Vetor desordenado:
[9][8][7][6][5][4][3][2][1][0]

Selection Sort:
Comparacoes: 45
Trocas: 5
```

Análise de Eficiência:

Bubble Sort:
- Comparações: 45
- Trocas: 45

Selection Sort:
- Comparações: 45
- Trocas: 5

- Em termos de comparações, ambos os algoritmos fazem o mesmo número de comparações (45) para o vetor dado.
- No entanto, em termos de trocas, o Bubble Sort faz muitas mais trocas (45) em comparação com o Selection Sort (5).
- Portanto, para este vetor de entrada específico, o Selection Sort é mais eficiente em termos de trocas, já que realiza significativamente menos trocas em comparação com o Bubble Sort.