Rafael Vieira Coelho

rafaelvc2@gmail.com

Ordenação

1) Bubble Sort

- Um dos algoritmos mais simples que existem:

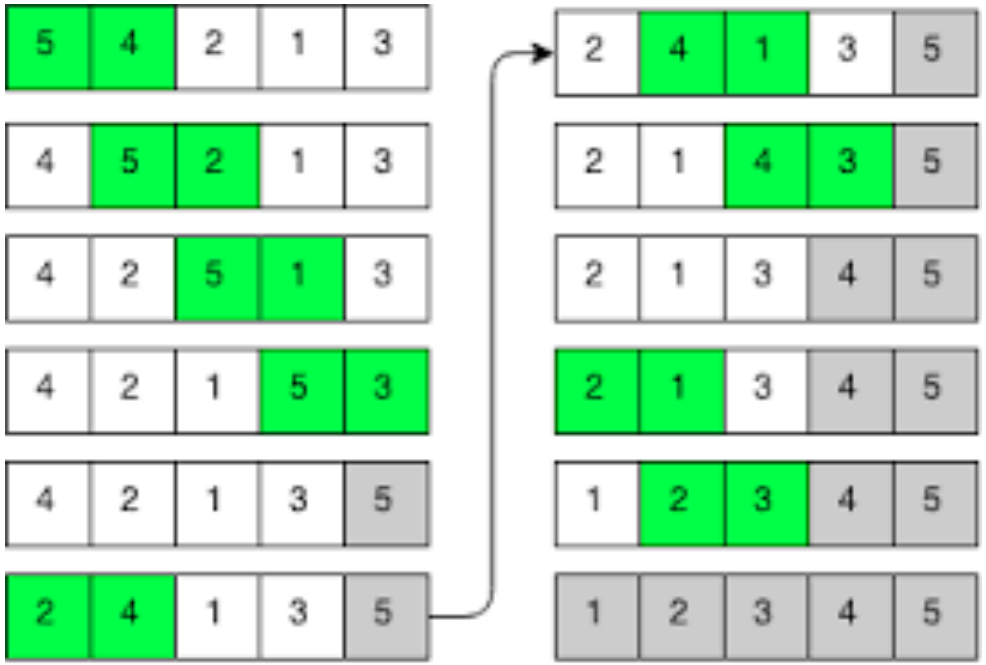
1. Percorre o vetor inteiro comparando elementos adjacentes (dois a dois)
2. Troca de lugar as posições dos elementos se eles estiverem fora de ordem
3. Repete os passos 1 e 2 com os primeiros n-1 itens, depois com os primeiros n-2 itens, até que reste apenas um item.

1) Bubble Sort

- Em Linguagem C:

```
void bubbleSort(int* vet, int length) {
    int i, j, temp;

    for (i = 0; i < length - 1; i++){
            for (j = (i+1); j < length; j++){
                if (vet[j] < vet[i]){
                    temp = vet[i];
                    vet[i] = vet[j];
                    vet[j] = temp;
                }
            }
        }
}
```



1) Bubble Sort

- Vantagens:
  - Simples de implementar e de entender.

- Desvantagens:
  - Lento;
  - Percorre toda a lista mesmo se estiver ordenada.

2) Selection Sort

1. Seleciona o menor item da lista.

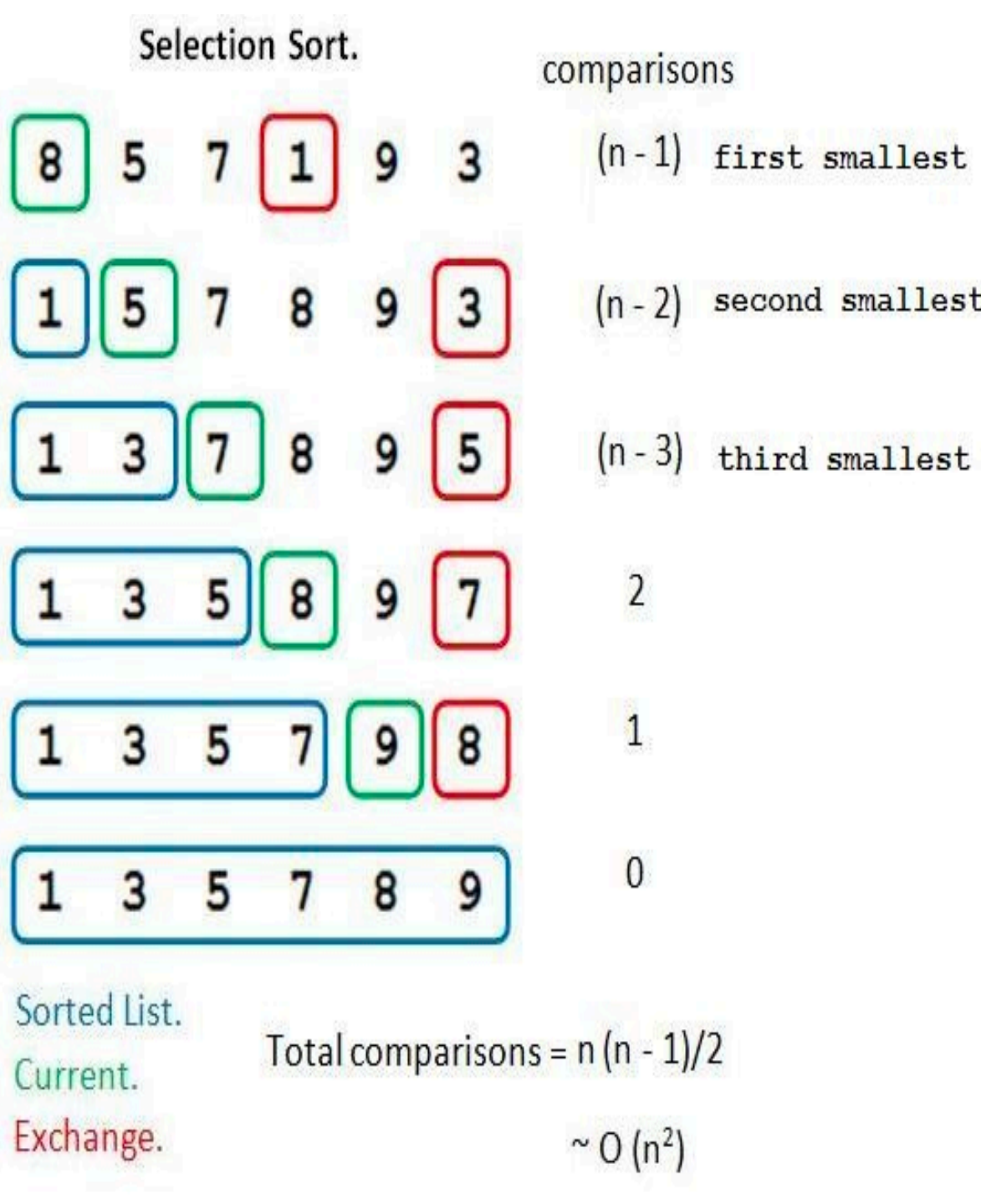
3. Troca ele com o item da primeira posição da lista.

5. Repete os passos 1 e 2 com os n - 1 itens restantes, depois com os n restantes, depois com os n - 2 itens, até que reste 2 itens, até que reste apenas um elemento.

2) Selection Sort

- Em Linguagem C:

```c
void selection_sort(int num[], int tam) {
  int i, j, min, aux;
  for (i = 0; i < (tam-1); i++)
  {
     min = i;
     for (j = (i+1); j < tam; j++) {
       if(num[j] < num[min])
         min = j;
     }
     if (num[i] != num[min]) {
       aux = num[i];
       num[i] = num[min];
       num[min] = aux;
     }
  }
}
```



2) Selection Sort

- Vantagens:
  - Ele é um algoritmo simples de ser implementado;
  - Não necessita de uma lista auxiliar;
  - Ocupa pouca memória;
  - Ele é veloz em listas de tamanho pequeno.

- Desvantagens:
  - Ele é um dos mais lentos para vetores de tamanhos grandes.

3) Insertion Sort

- Em cada passo a partir de i=2 faça:

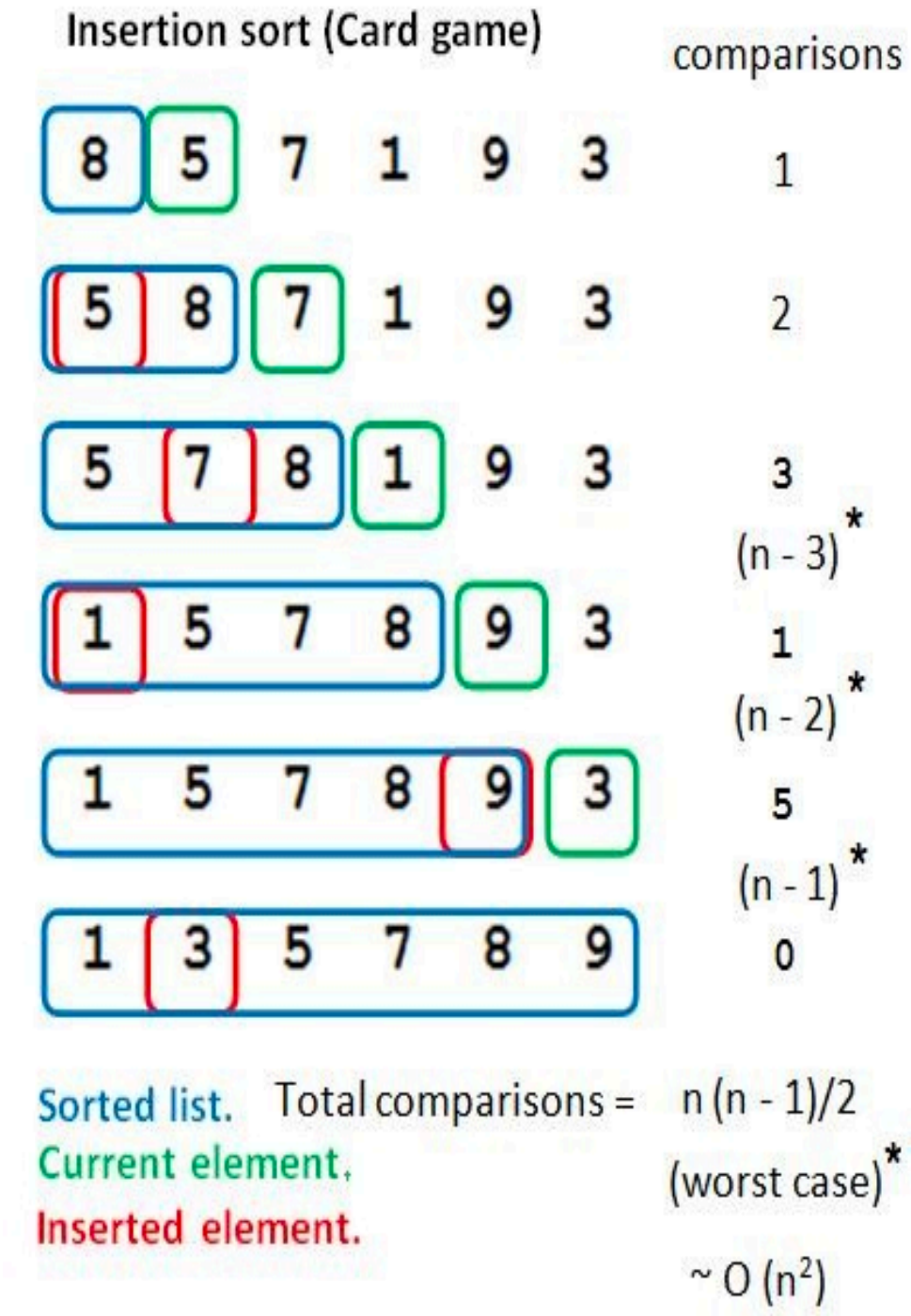
1. Selecione o i-ésimo item da lista.

3. Coloque-o no lugar apropriado na lista destino de acordo com o critério de ordenação.

3) Insertion Sort

- Em Linguagem C:

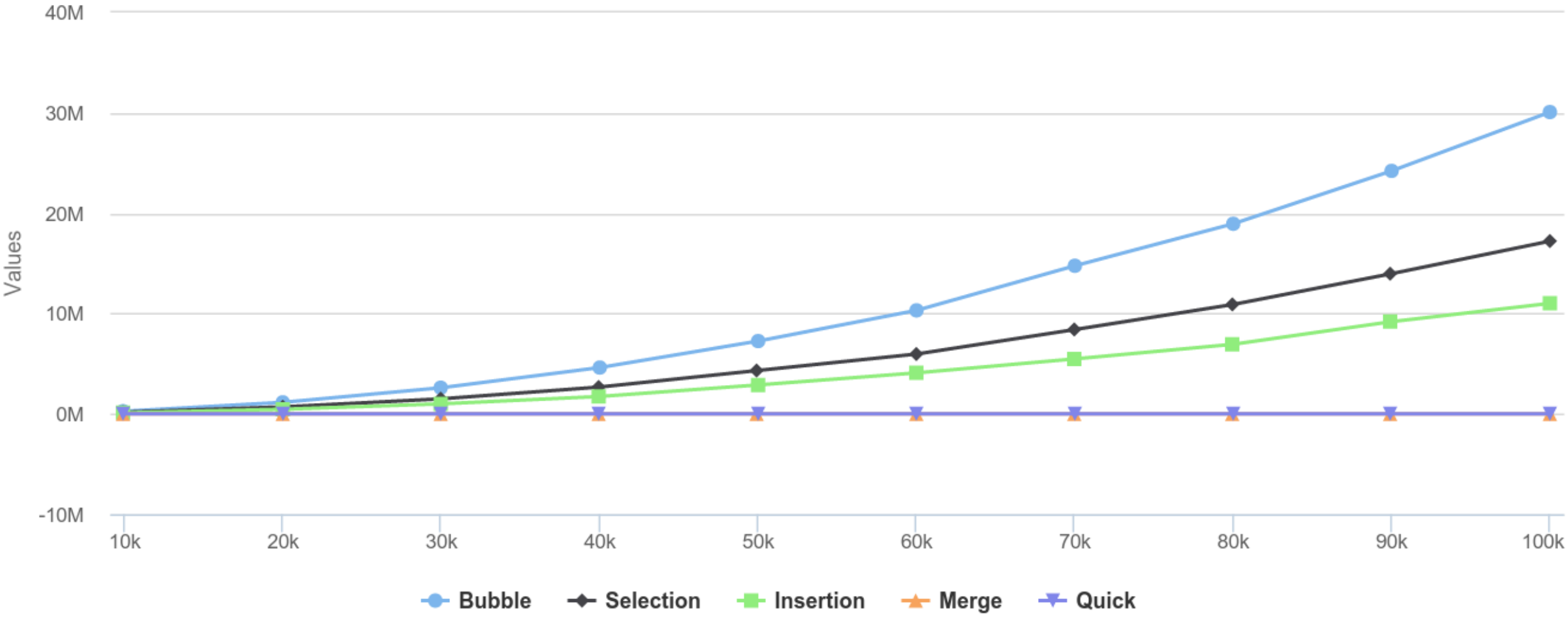
```c
void insertionSort(int* original, int length) {
    int i, j, atual;
    for (i = 1; i < length; i++) {
        atual = original[i];
        for (j = i - 1; (j >= 0) && (atual < original[j]); j--) {
            original[j + 1] = original[j];
        }
        original[j+1] = atual;
    }
}
```



3) Insertion Sort

- Vantagens:
  - É o método a ser utilizado quando a lista está "quase" ordenada
  - É um bom método quando se desejar adicionar poucos elementos em uma lista já ordenada.
- Desvantagens:
  - Alto custo para movimentar itens na lista.

Comparação


40M
30M
20M
10M
0M
-10M
Values
10k
20k
30k
40k
50k
60k
70k
80k
90k
100k
Bubble
Selection
Insertion
Merge
Quick

Links

Bolha:

- https://www.youtube.com/watch?v=lyZQPjUT5B4

Seleção:

- https://www.youtube.com/watch?v=Ns4TPTC8whw&feature=related

Inserção:

- https://www.youtube.com/watch?v=ROalU379l3U&feature=related

QuickSort:

- https://www.youtube.com/watch?v=cnzIChso3cc

Exercício:

Complemente o Código, implementando as 3 funções: **bubble_sort**, **selection_sort** e **insertion_sort**.

Todos recebem uma lista de números gerados randomicamente.

```
import time
from random import shuffle

def main():
    nitems=5000

    numbs=[i for i in range(nitems)]
    shuffle(numbs)
    start=time.clock()
    bubble_sort(numbs)
    end = time.clock()
    print('Bubble', end-start, "seconds")

    numbs=[i for i in range(nitems)]
    shuffle(numbs)
    start=time.clock()
    selection_sort(numbs)
    end = time.clock()
    print('SelectionSort', end-start, "seconds")

    numbs=[i for i in range(nitems)]
    shuffle(numbs)
    start=time.clock()
    insertion_sort(numbs)
    end = time.clock()
    print('InsertionSort', end-start, "seconds")
```