Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

AULA 04

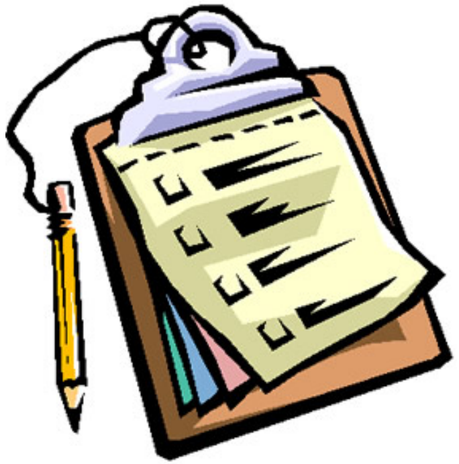
Estrutura de dados básico I (EDB1)

Prof. Msc. Janiheryson Felipe (Felipe)

Natal, RN
2023

OBJETIVO DA AULA

- Apresentar algoritmos de ordenação;
  - Conhecer o algoritmo bubble sort:
  - Conhecer o algoritmo selection sort:
  - Conhecer o algoritmo insertion sort



ALGORITMOS DE ORDENAÇÃO

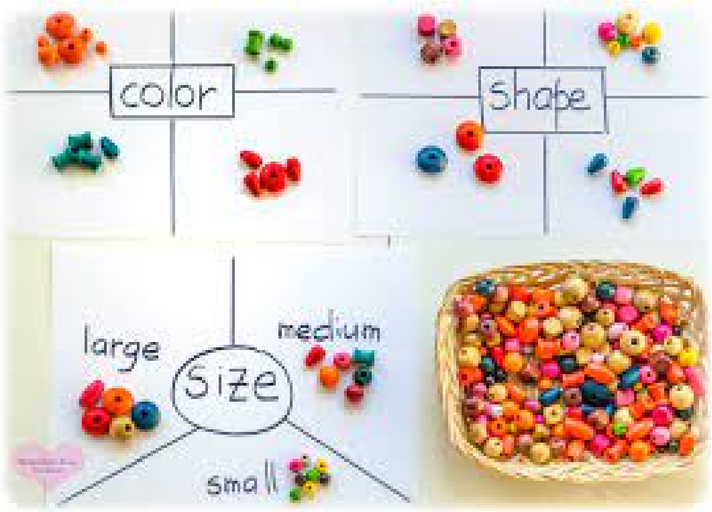
ALGORITMOS ORDENAÇÃO

Algoritmos de ordenação são procedimentos computacionais que organizam um conjunto de dados em uma determinada ordem, de acordo com um critério específico.

ALGORITMOS ORDENAÇÃO

Esses algoritmos podem ser usados para ordenar dados:

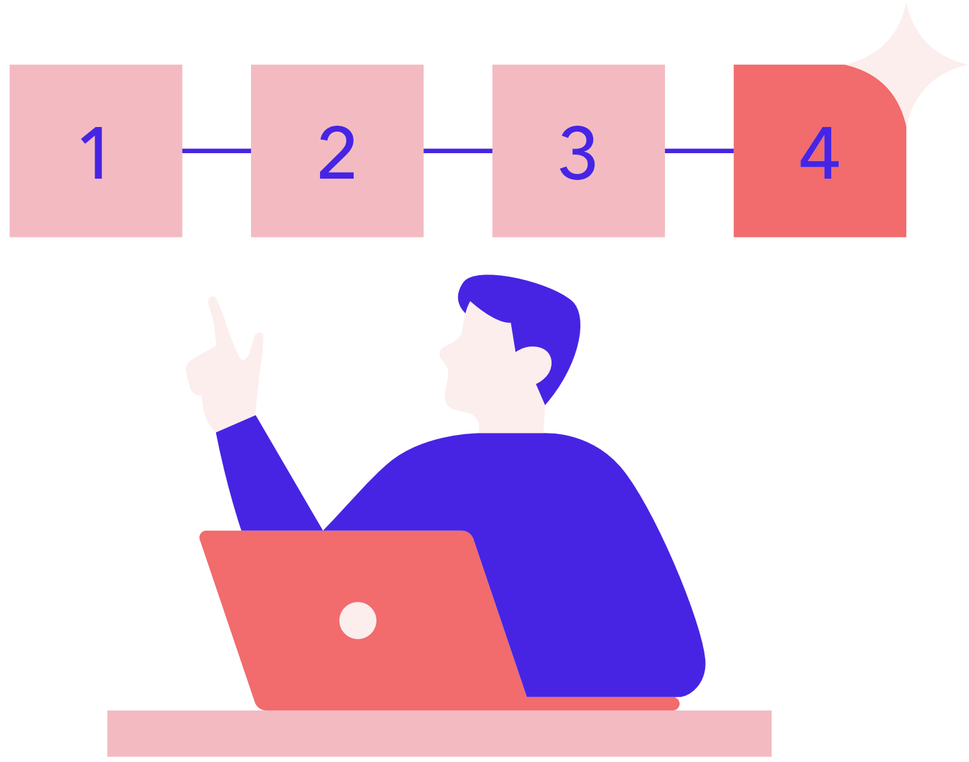
- Ordem crescente ou decrescente,
- Ordem alfabética,
- Ordem de tamanho
- Qualquer outra ordem desejada.



ALGORITMOS ORDENAÇÃO

Os principais algoritmos de ordenação são:

- Bubble Sort
- Selection Sort
- Insertion Sort:
- Merge Sort:
- Quick Sort:



BUBBLE SORT

O Bubble Sort é um algoritmo simples de ordenação que compara pares de elementos adjacentes e os troca de posição se estiverem fora de ordem, repetindo esse processo até que a lista esteja completamente ordenada.

BUBBLE SORT

O funcionamento do Bubble Sort pode ser detalhado da seguinte forma:

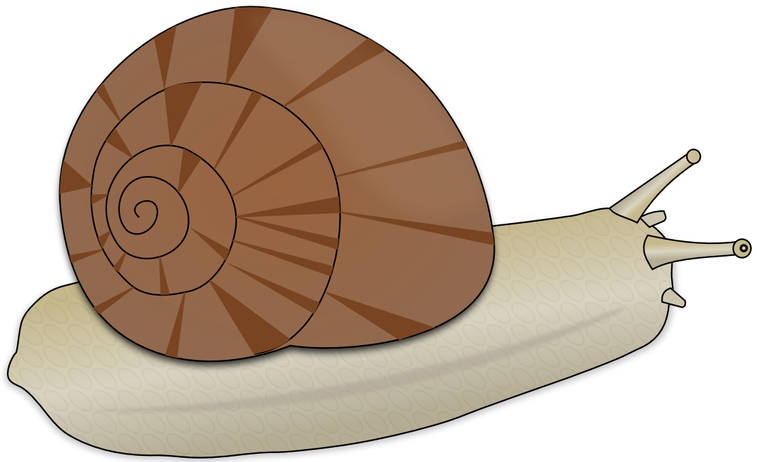
1. Começando pelo início da lista, compara o primeiro elemento com o próximo e verifica se eles estão fora de ordem. Se estiverem, troca a posição dos elementos.
2. Avança para o próximo par de elementos e repete o processo de comparação e troca, novamente verificando se estão fora de ordem.
3. Repete o processo para cada par de elementos na lista, avançando de um par para o próximo até que se chegue ao final da lista.
4. Quando se chega ao final da lista, o último elemento estará na posição correta, então o processo é reiniciado a partir do início da lista, repetindo a mesma sequência de comparações e trocas de elementos.
5. O processo continua até que nenhum par de elementos na lista esteja mais fora de ordem, ou seja, até que a lista esteja completamente ordenada.

BUBBLE SORT

O Bubble Sort tem um desempenho relativamente simples, mas pode ser ineficiente para listas muito grandes, pois sua complexidade de tempo é $O(n^2)$, o que significa que o tempo de execução cresce proporcionalmente ao quadrado do número de elementos na lista.

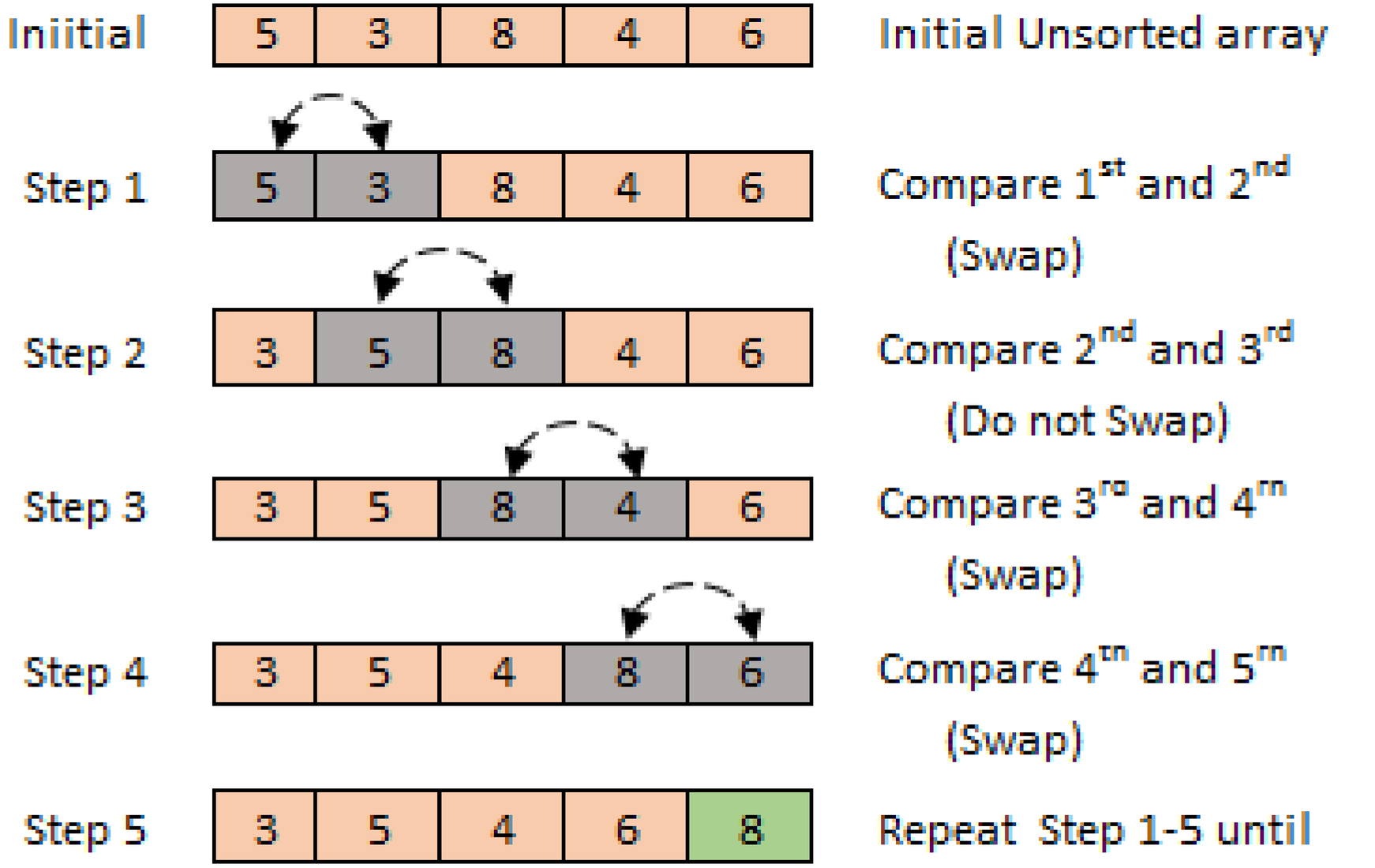
BUBBLE SORT

Por isso, para listas grandes, outros algoritmos de ordenação mais eficientes, como o Merge Sort ou Quick Sort, são geralmente preferidos.



BUBBLE SORT

Bubble sort example

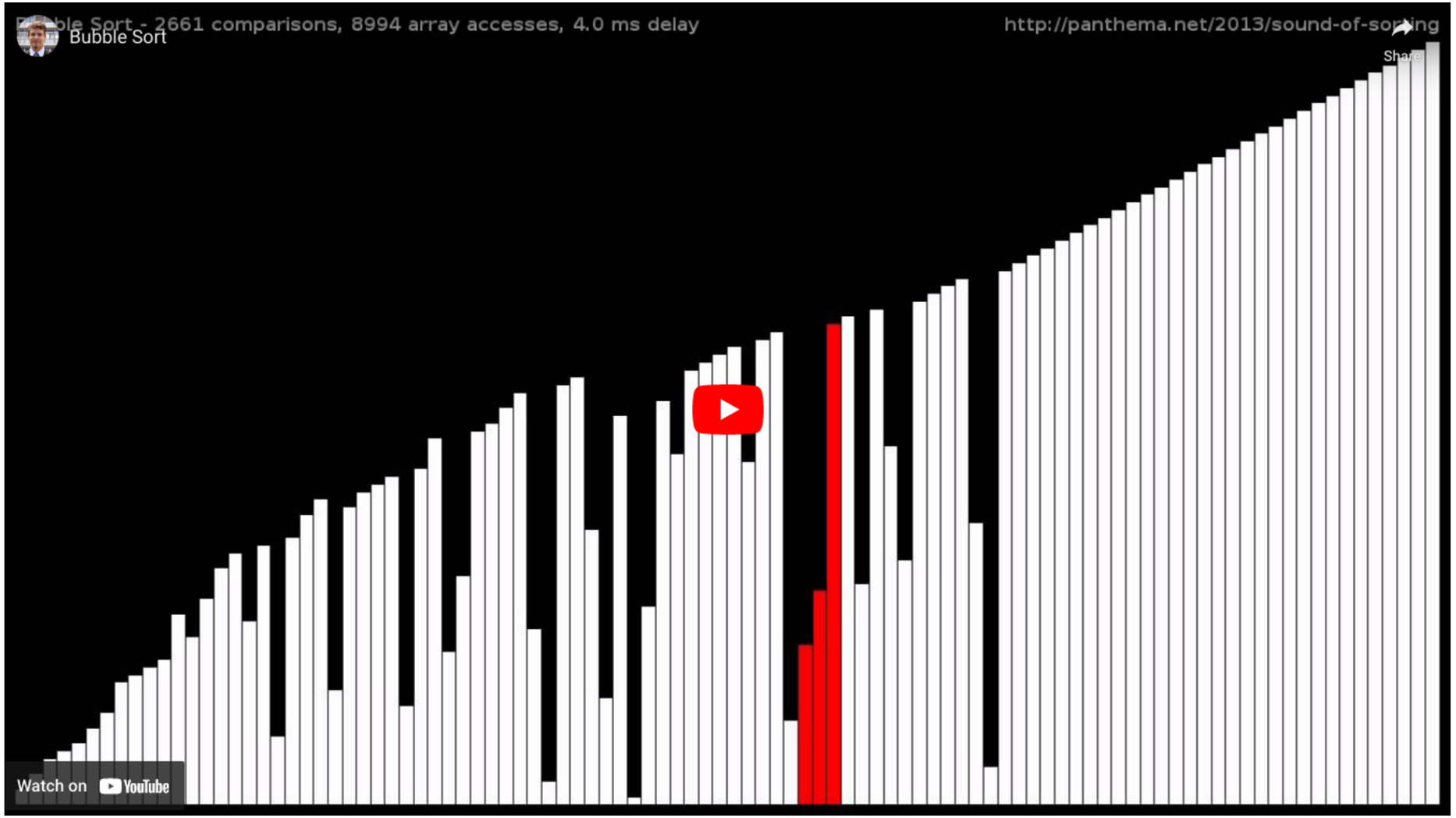


BUBBLE SORT

```
void bubble_sort(int lista[], int n) {
    for(int i = 0; i < n-1; i++) {
        for(int j = 0; j < n-i-1; j++) {
            if(lista[j] > lista[j+1]) {
                int temp = lista[j]; lista[j] = lista[j+1]; lista[j+1
            }
        }
    }
}
```

```
int main() {
    int lista[] = {5, 3, 8, 2, 1, 9};
    int n = sizeof(lista) / sizeof(lista[0]);
    bubble_sort(lista, n);
    cout << "Lista ordenada: ";
    for(int i = 0; i < n; i++) {
        cout << lista[i] << " ";
    }
    return 0;
```

BUBBLE SORT


Bubble Sort - 2661 comparisons, 8994 array accesses, 4.0 ms delay
Bubble Sort
http://panthema.net/2013/sound-of-sorting
Share
Watch on YouTube

SELECTION SORT

O Selection Sort é um algoritmo de ordenação simples e intuitivo que percorre uma lista de elementos e os organiza em ordem crescente ou decrescente, dependendo da implementação. O funcionamento do Selection Sort consiste em selecionar o menor elemento da lista não ordenada e colocá-lo na posição correta na lista ordenada.

SELECTION SORT

o algoritmo percorre a lista em busca do menor elemento, e quando encontra, o troca com o primeiro elemento da lista não ordenada. Em seguida, o algoritmo passa a considerar apenas os elementos restantes na lista não ordenada e repete o processo até que toda a lista esteja ordenada.

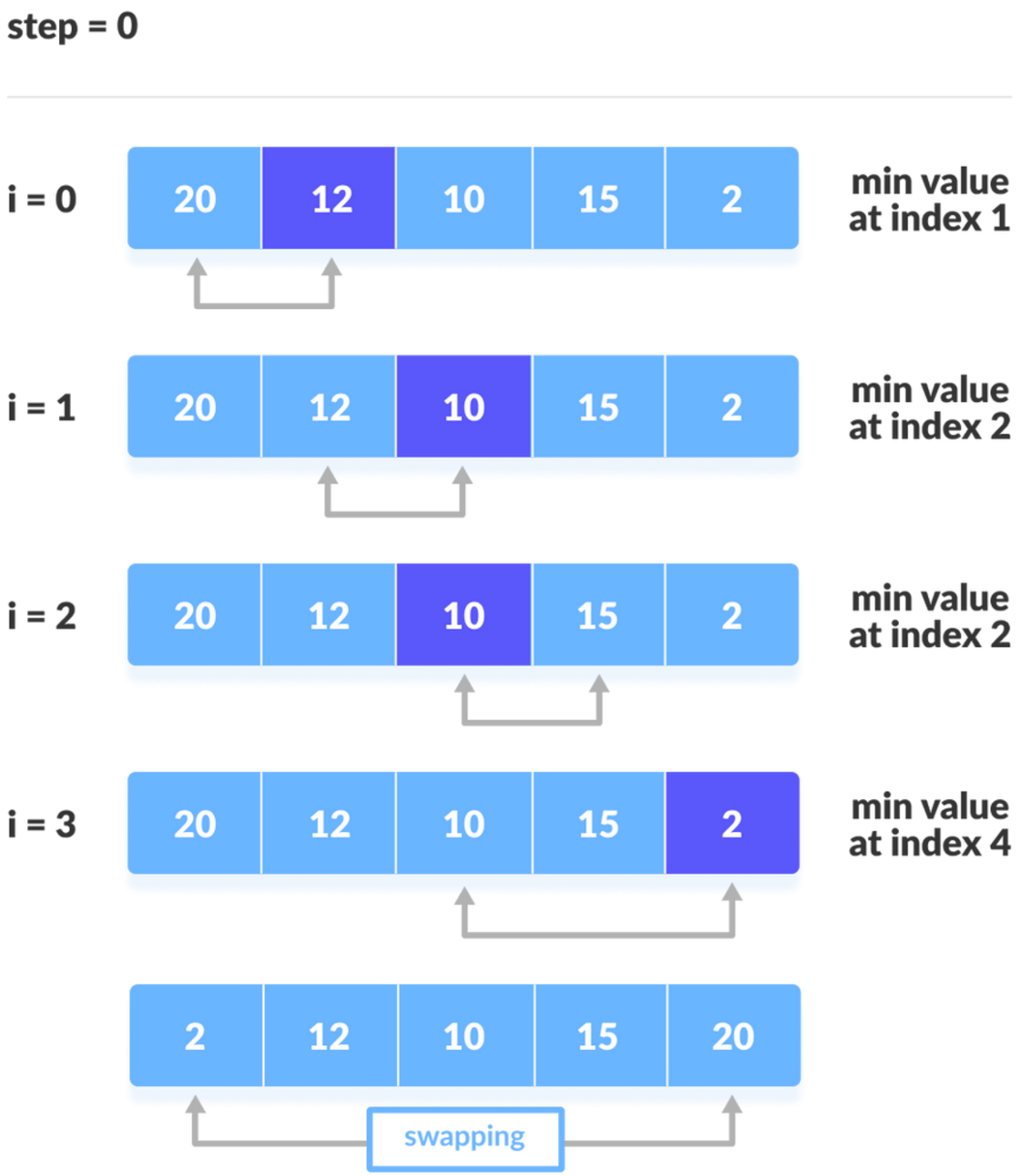
SELECTION SORT

O Selection Sort é um algoritmo de complexidade $O(n^2)$, o que significa que seu tempo de execução aumenta quadraticamente com o tamanho da lista. Isso faz com que ele não seja uma boa opção para ordenar grandes conjuntos de dados. No entanto, ele pode ser útil para ordenar pequenos conjuntos de dados

SELECTION SORT

Apesar de não ser muito eficiente em termos de tempo de execução, o Selection Sort é fácil de implementar e requer pouco espaço de memória, tornando-o uma opção viável em algumas situações.

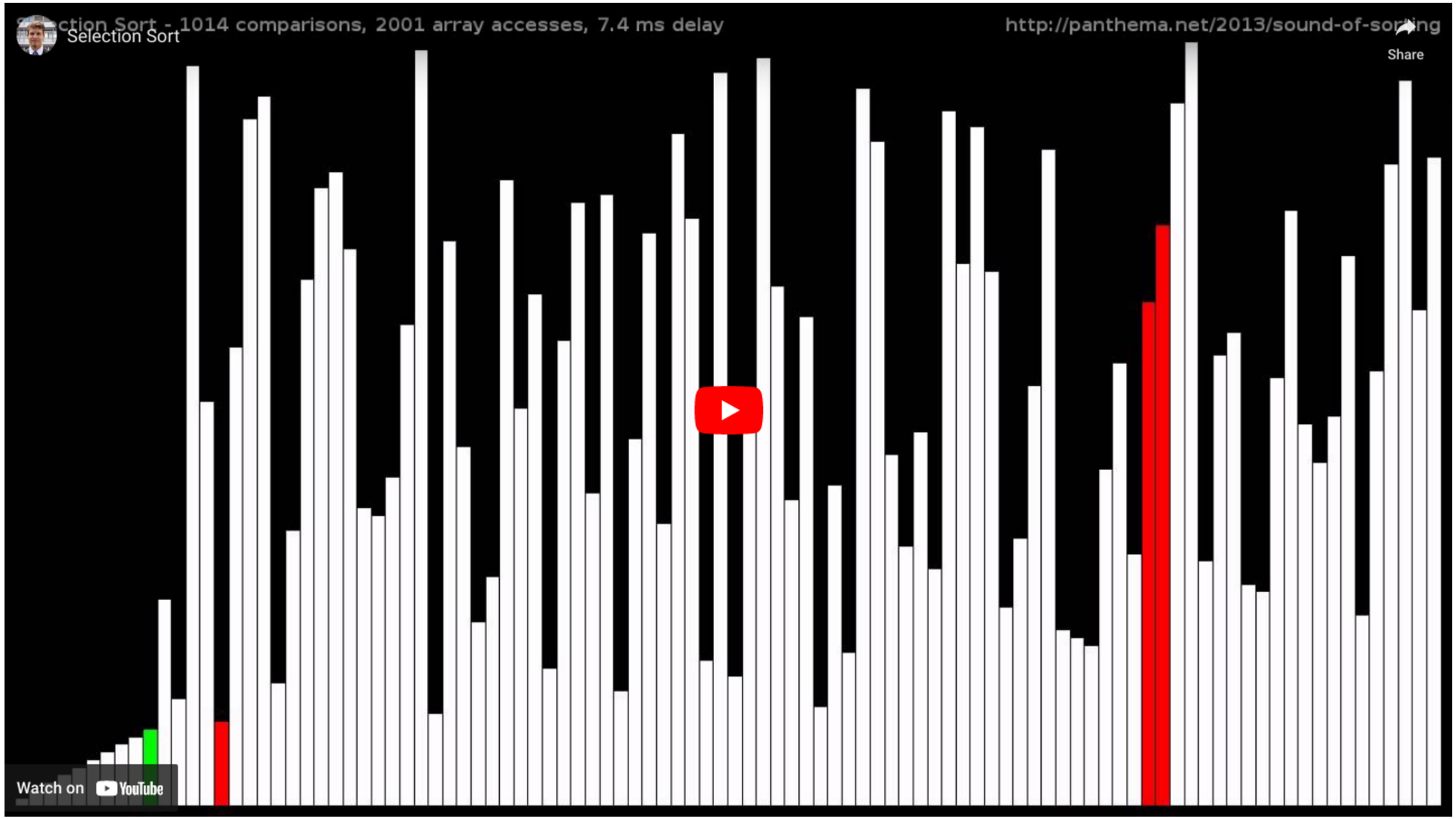
SELECTION SORT



SELECTION SORT

```c
void selectionSort(int arr[], int n) {
    int i, j, min_idx;
    // Percorre a lista de elementos
    for (i = 0; i < n-1; i++) {
        // Encontra o menor elemento na lista não ordenada
        min_idx = i;
        for (j = i+1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        // Troca o menor elemento com o primeiro elemento na lista
        int temp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = temp;
    }
}
```

SELECTION SORT



INSERCTION SORT

O Insertion Sort é um algoritmo de ordenação que percorre uma lista de elementos e os organiza em ordem crescente ou decrescente, dependendo da implementação. Este algoritmo é bastante eficiente para ordenar pequenas quantidades de elementos, mas é menos eficiente para ordenar grandes quantidades.

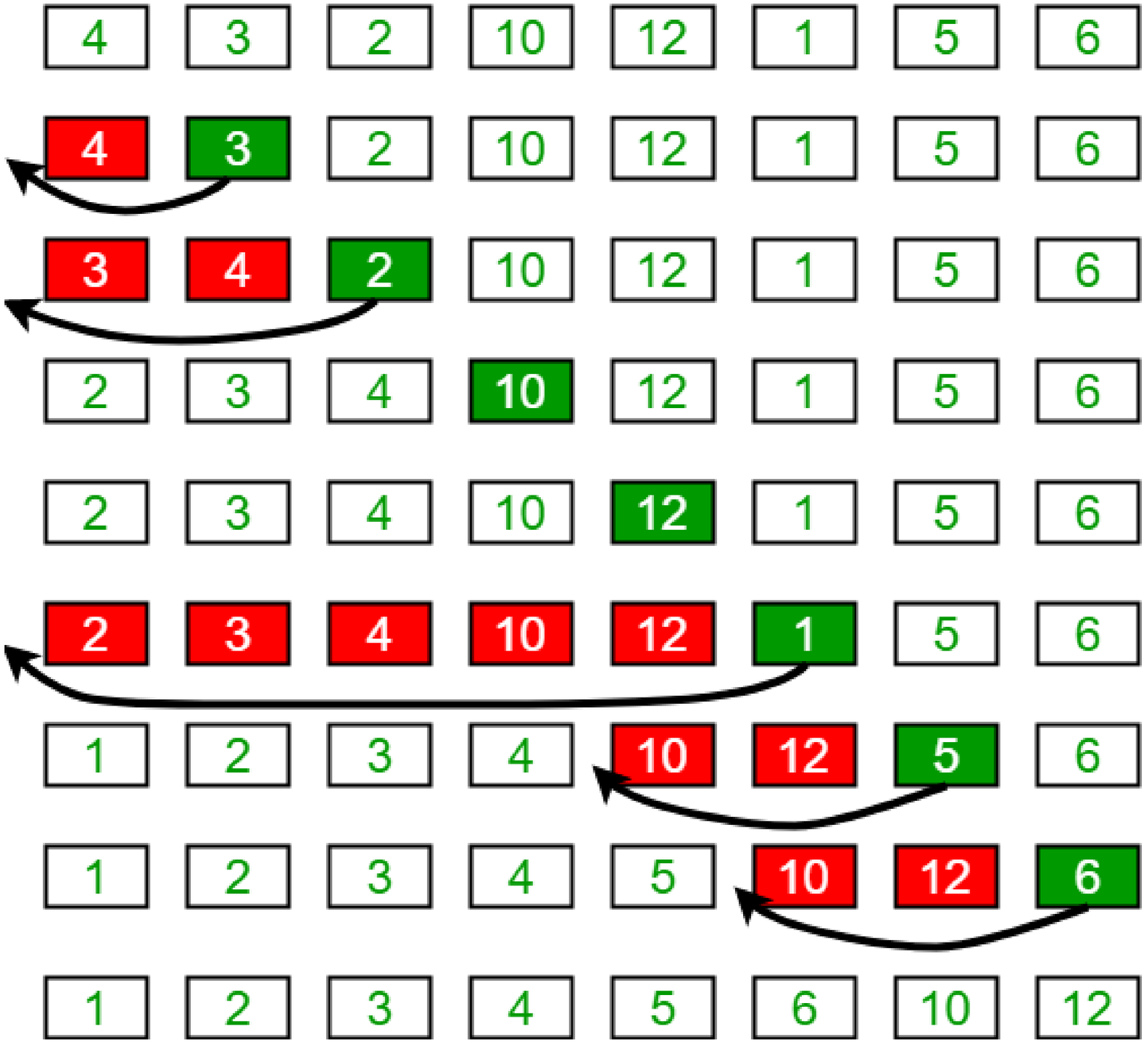
INSERCTION SORT

O funcionamento do Insertion Sort consiste em percorrer a lista de elementos a partir do segundo elemento e comparar cada elemento com seus antecessores, inserindo-o na posição correta. Para isso, o algoritmo compara o elemento atual com os elementos anteriores, movendo-os para a direita enquanto eles forem maiores do que o elemento atual, até que a posição correta seja encontrada.

INSERCTION SORT

O Insertion Sort é um algoritmo de complexidade $O(n^2)$, o que significa que seu tempo de execução aumenta quadraticamente com o tamanho da lista. No entanto, ele é eficiente para ordenar pequenas quantidades de elementos e pode ser mais rápido do que outros algoritmos de ordenação para casos específicos, como quando a lista já está quase ordenada.

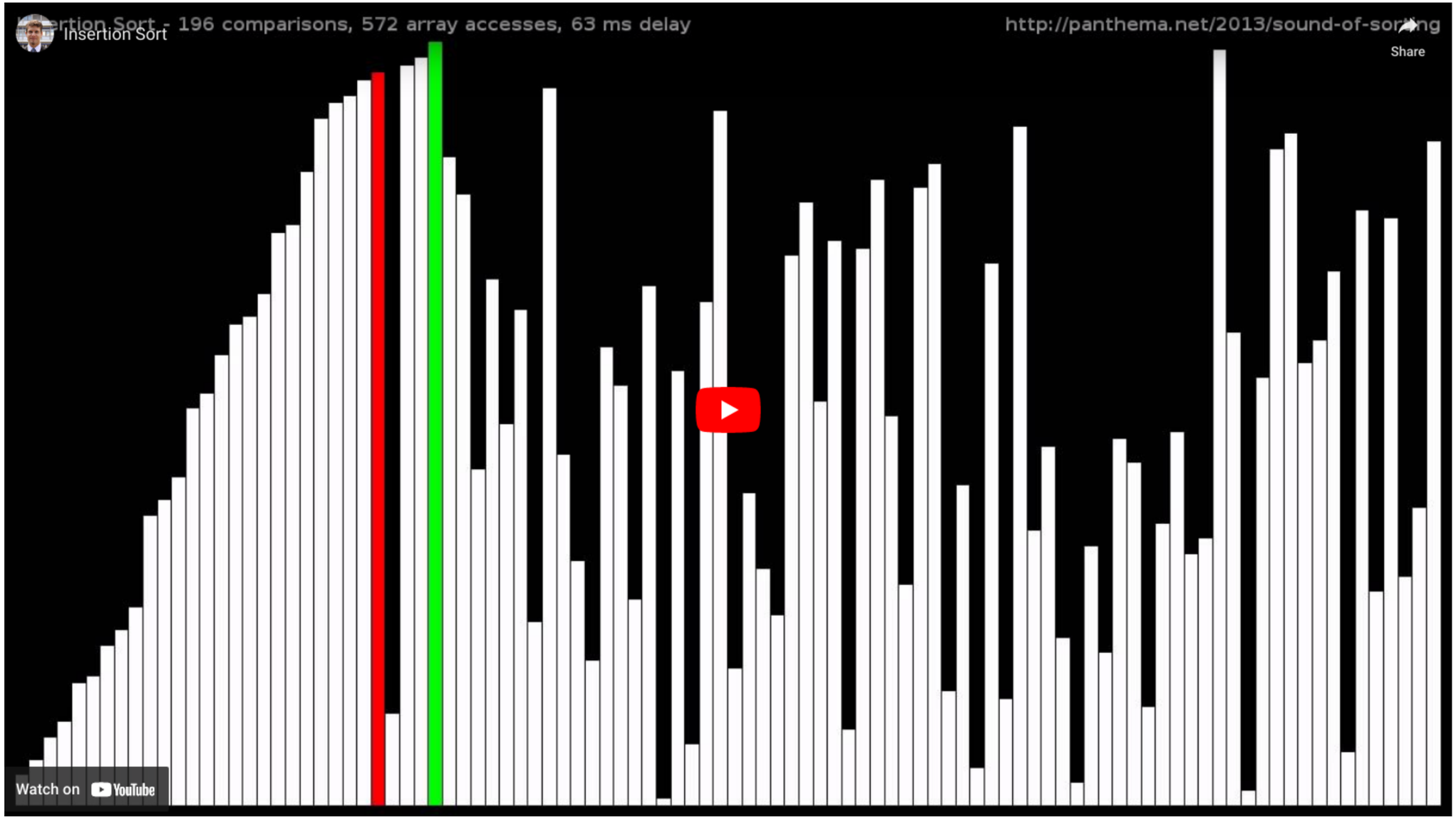
INSERCTION SORT



INSERCTION SORT

```c
void insertionSort(int arr[], int n) {
    int i, j, key;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j = j - 1;
        }
        arr[j+1] = key;
    }
}
```

INSERCTION SORT



DÚVIDAS???