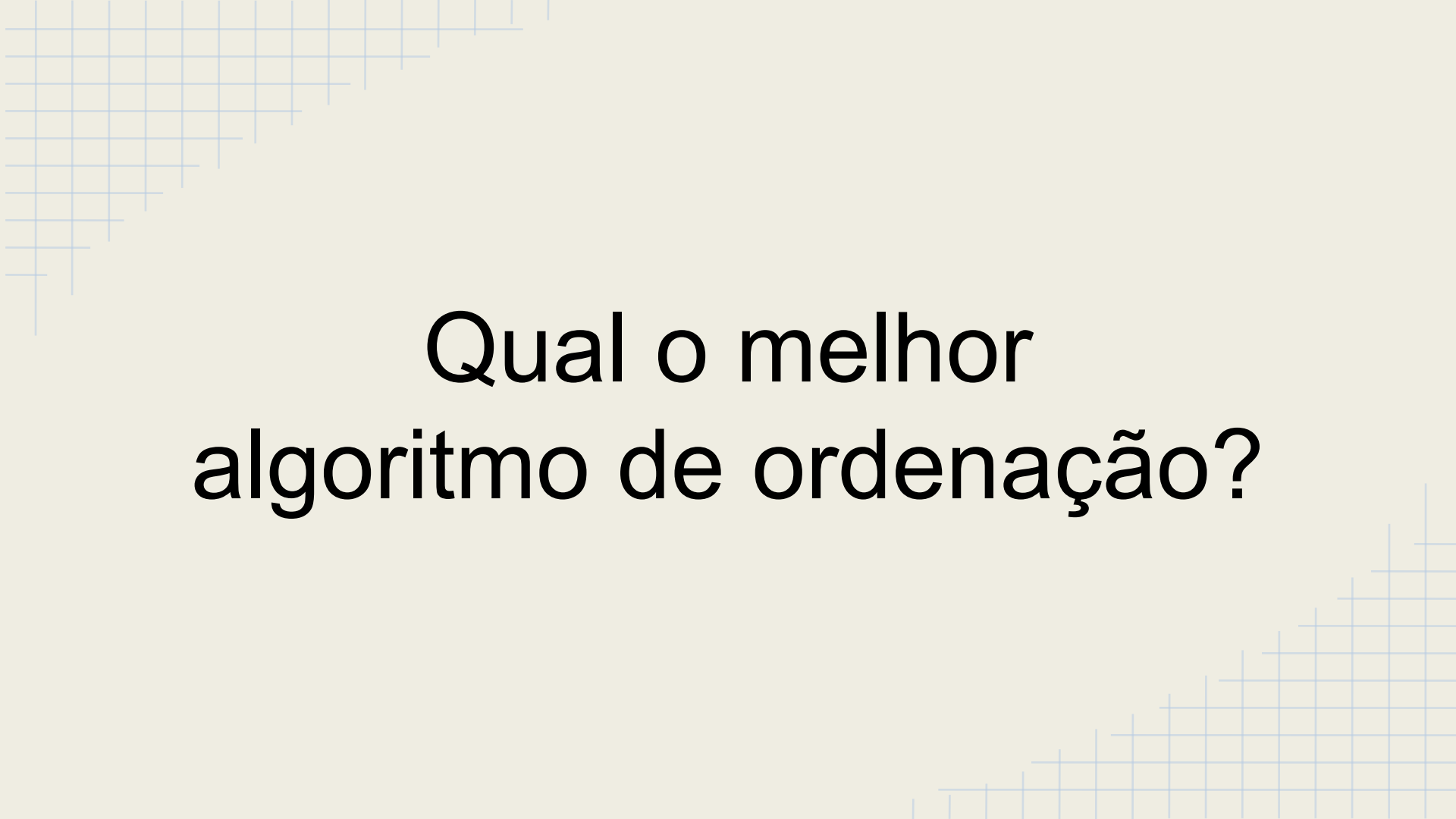
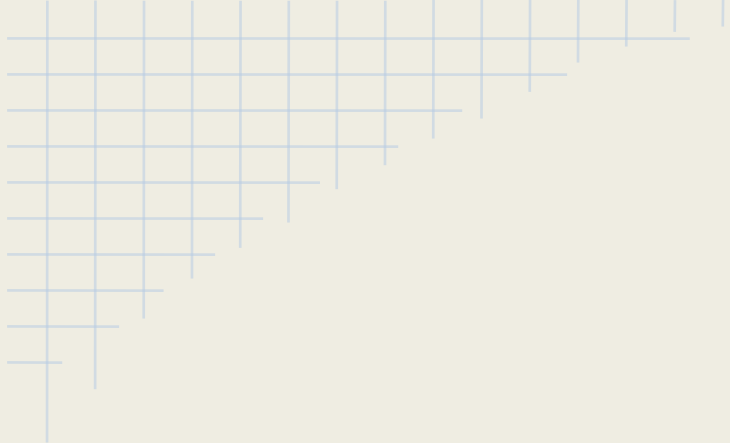


Métodos de Ordenação

Andressa Andrade, Guilherme Bernal, Renata Antunes e Rodrigo Fernandes



Qual o melhor
algoritmo de ordenação?



Não sei.

Isso depende.



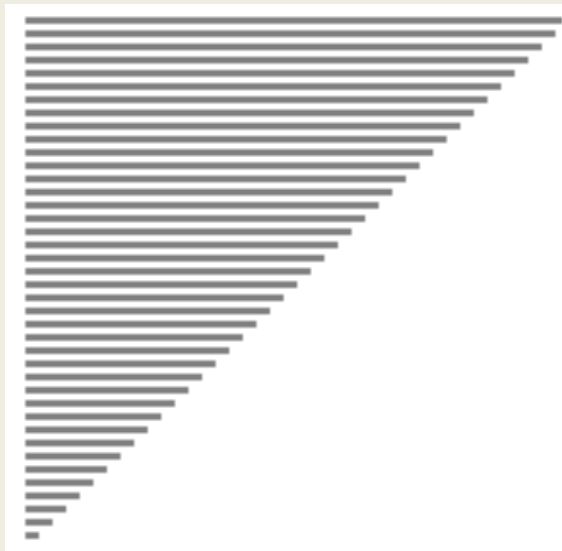
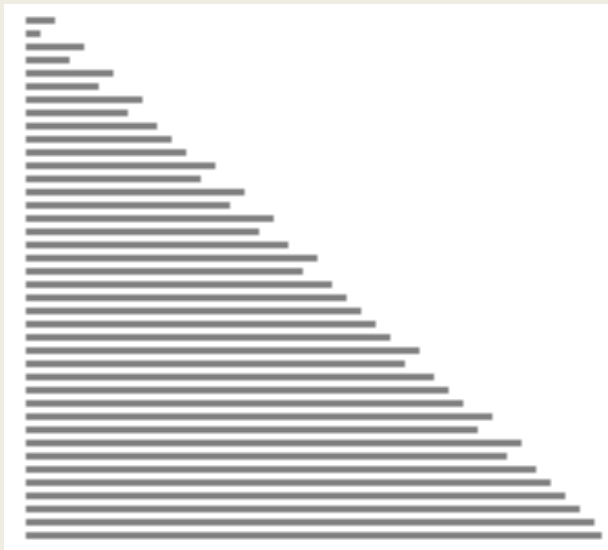
Selection Sort

```
1 void selection_sort(int list[], int size) {  
2     int i, k, min, aux;  
3  
4     for (i = 0; i < (size-1); i++) {  
5         min = i;  
6  
7         for (k = (i+1); k < size; k++) {  
8             if(list[k] < list[min]) {  
9                 min = k;  
10            }  
11        }  
12  
13        if (i != min) {  
14            aux = list[i];  
15            list[i] = list[min];  
16            list[min] = aux;  
17        }  
18    }  
19 }
```

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

Selection Sort

- Não se adapta ao dado, sempre $O(n^2)$
- Minimiza swaps



Bubble Sort

- Ordenação por troca
- Um dos algoritmos mais simples
- Custo caro

```
5 void bubbleSort(int* list, int size) {  
6     for (int i = 0; i < (size - 1); i++) {  
7         for (int j = 0; j < size - (i + 1); j++) {  
8             if (list[j] > list[j + 1]) {  
9                 swap(list[j], list[j + 1]);  
10            }  
11        }  
12    }  
13 }
```

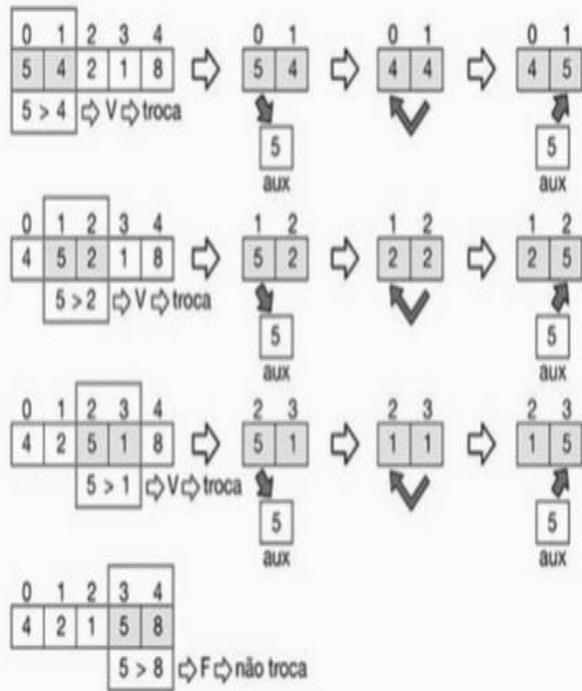
Bubble Sort

- Complexidade de tempo:
 - Melhor caso: $O(n)$
 - Caso médio: $O(n^2)$
 - Pior caso: $O(n^2)$

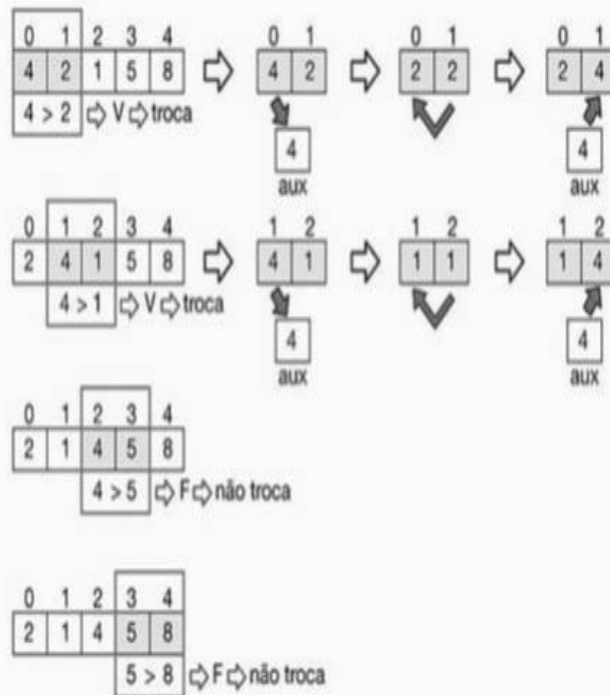
6 5 3 1 8 7 2 4

Bubble Sort

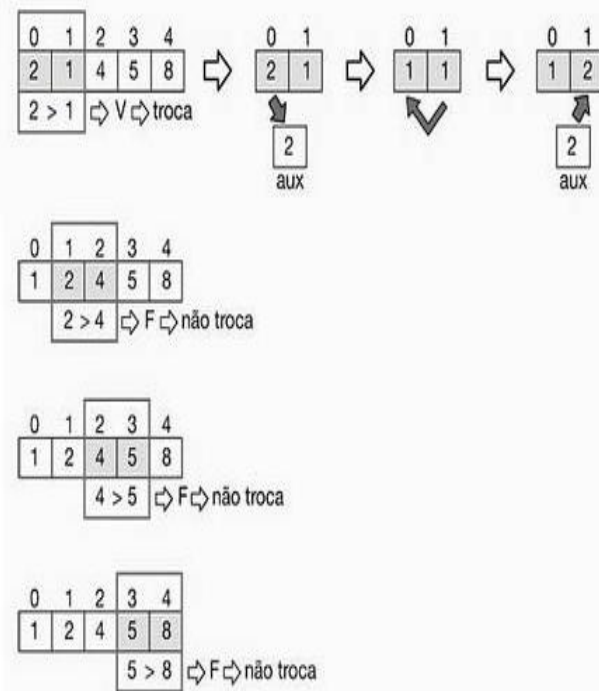
1ª execução do laço



2ª execução do laço



3ª execução do laço



Bubble Sort

4ª execução do laço

0	1	2	3	4
1	2	4	5	8
1 > 2 ⇒ F ⇒ não troca				

0	1	2	3	4
1	2	4	5	8
2 > 4 ⇒ F ⇒ não troca				

0	1	2	3	4
1	2	4	5	8
4 > 5 ⇒ F ⇒ não troca				

0	1	2	3	4
1	2	4	5	8
5 > 8 ⇒ F ⇒ não troca				

5ª execução do laço

Apesar de o vetor já estar ordenado, mais uma execução do laço será realizada.

0	1	2	3	4
1	2	4	5	8
1 > 2 ⇒ F ⇒ não troca				

0	1	2	3	4
1	2	4	5	8
2 > 4 ⇒ F ⇒ não troca				

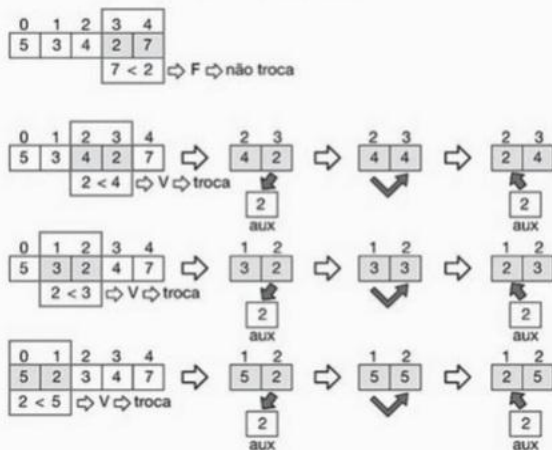
0	1	2	3	4
1	2	4	5	8
4 > 5 ⇒ F ⇒ não troca				

0	1	2	3	4
1	2	4	5	8
5 > 8 ⇒ F ⇒ não troca				

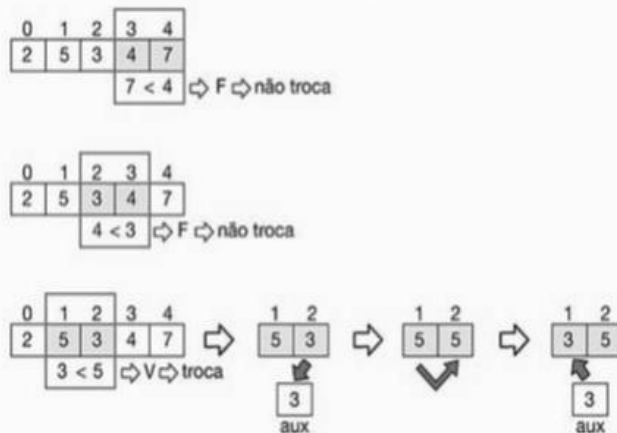
Bubble Sort

- Bubble Sort melhorado

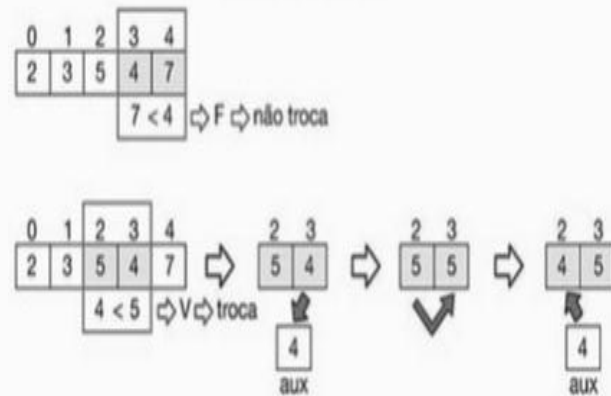
1ª execução do laço



2ª execução do laço

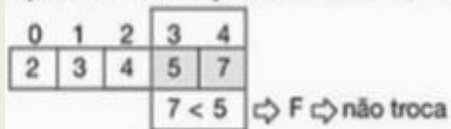


3ª execução do laço



4ª execução do laço

Apesar de o vetor já estar ordenado, mais uma execução do laço será realizada.



Insertion Sort

- Ordenação por inserção
- Mais rápido que o Bubble Sort

```
1
2 void insertionSort(int* list, int size) {
3     for (int j = 1; j < size; j++) {
4         int aux = list[j];
5         int i = j - 1;
6         while (i >= 0 && list[i] > aux) {
7             list[i+1] = list[i];
8             i--;
9         }
10        list[i+1] = aux;
11    }
12 }
```

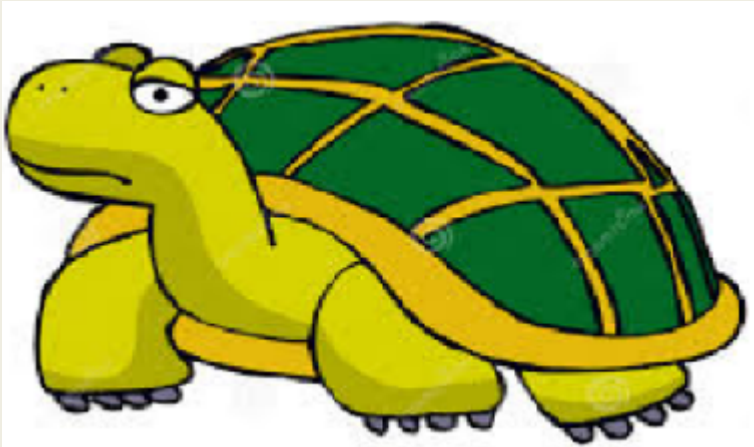
Insertion Sort

- Complexidade de tempo:
 - Melhor caso: $O(n)$
 - Caso médio: $O(n^2)$
 - Pior caso: $O(n^2)$

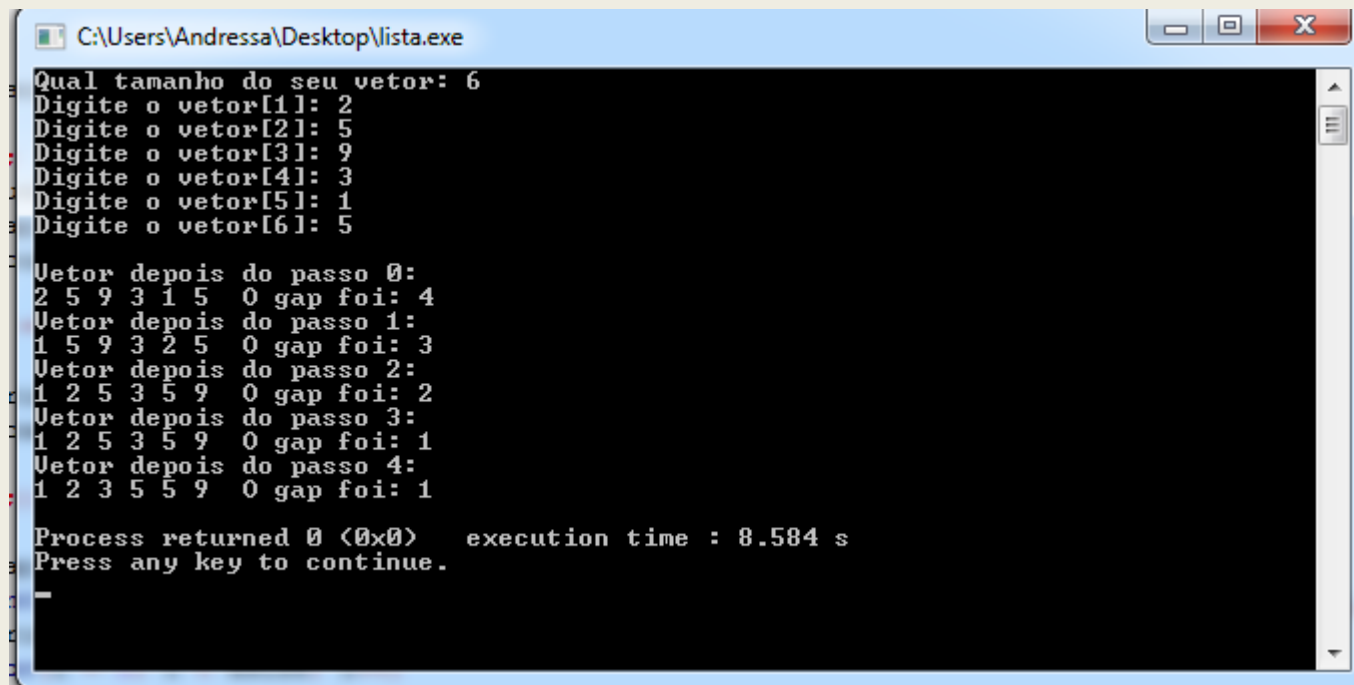
6 5 3 1 8 7 2 4

Comb Sort

- O método
- O gap



O método



```
C:\Users\Andressa\Desktop\lista.exe
Qual tamanho do seu vetor: 6
Digite o vetor[1]: 2
Digite o vetor[2]: 5
Digite o vetor[3]: 9
Digite o vetor[4]: 3
Digite o vetor[5]: 1
Digite o vetor[6]: 5

Vetor depois do passo 0:
2 5 9 3 1 5  0 gap foi: 4
Vetor depois do passo 1:
1 5 9 3 2 5  0 gap foi: 3
Vetor depois do passo 2:
1 2 5 3 5 9  0 gap foi: 2
Vetor depois do passo 3:
1 2 5 3 5 9  0 gap foi: 1
Vetor depois do passo 4:
1 2 3 5 5 9  0 gap foi: 1

Process returned 0 (0x0)   execution time : 8.584 s
Press any key to continue.
-
```

Combsort 11

- (9, 6, 4, 3, 2, 1)
- (10, 7, 5, 3, 2, 1)
- (11, 8, 6, 4, 3, 2, 1)

```
C:\Users\Andressa\Desktop\combsort.exe

Vetor passo 4:
2 3 6 5 4 5 7 4 1 12 25 36 25 14 6 25 8 9 20 25 26 282 58 96 87 41 98 58 98 74
0 gap foi 11

Vetor passo 5:
1 3 6 5 4 5 6 4 2 9 20 25 25 14 7 25 8 12 25 36 26 74 58 96 87 41 98 58 98 282
0 gap foi 8

Vetor passo 6:
1 3 2 5 4 5 6 4 6 9 8 12 25 14 7 25 20 25 25 36 26 58 58 96 87 41 98 74 98 282
0 gap foi 6

Vetor passo 7:
1 3 2 4 4 5 6 5 6 9 7 12 20 14 8 25 25 25 25 36 26 41 58 74 87 58 98 96 98 282
0 gap foi 4

Vetor passo 8:
1 3 2 4 4 5 6 5 6 9 7 8 20 14 12 25 25 25 25 36 26 41 58 74 87 58 98 96 98 282
0 gap foi 3

Vetor passo 9:
1 3 2 4 4 5 6 5 6 8 7 9 12 14 20 25 25 25 25 36 26 41 58 58 87 74 98 96 98 282
0 gap foi 2

Vetor passo 10:
1 2 3 4 4 5 5 6 6 7 8 9 12 14 20 25 25 25 25 26 36 41 58 58 74 87 96 98 98 282
0 gap foi 1

Vetor ordenado:
```

Melhorando a Eficiencia

```
static int novogap(int gap) {  
    gap = (gap * 10) / 13;  
    if (gap == 9 || gap == 10)  
    {  
        gap = 11;  
    }  
    if (gap < 1)  
    {  
        gap = 1;  
    }  
    return gap;  
}
```


Shell Sort

- O método



O método

```
C:\Users\Andressa\Desktop\shellsort.exe
Qual tamanho do seu vetor: 8
Digite o vetor[1]: 12
Digite o vetor[2]: 43
Digite o vetor[3]: 1
Digite o vetor[4]: 6
Digite o vetor[5]: 56
Digite o vetor[6]: 23
Digite o vetor[7]: 52
Digite o vetor[8]: 9

Vetor passo 0
12 43 1 6 56 23 52 9
0 gap foi 4

Vetor passo 1
12 23 1 6 56 43 52 9
0 gap foi 1

Vetor ordenado:
1 6 9 12 23 43 52 56

Process returned 0 (0x0)   execution time : 12.714 s
Press any key to continue.
```

Primeiro passo

- A primeira coisa que ele faz é pegar o tamanho dos "pulos" (gap, em inglês) para “montar” diversos vetores de menor tamanho (subvetores dentro do vetor inicial).

```
do {  
    gap = 3 * gap + 1;  
} while(gap < size);
```

Segundo passo

```
do {  
    gap /= 3;  
    for(i = gap; i < size; i++) {  
        value = vet[i];  
        j = i - gap;  
        while(j >= 0 && value < vet[j]) {  
            vet[j + gap] = vet[j];  
            j -= gap;  
        }  
        vet[j + gap] = value;  
    }  
} while(gap > 1);
```

Merge Sort

```
1 void merge_sort(int list[], int size) {  
2     int mid;  
3  
4     if (size > 1) {  
5         mid = size / 2;  
6         merge_sort(list, mid);  
7         merge_sort(list + mid, size - mid);  
8         merge(list, size);  
9     }  
10 }
```

Merge Sort

6 5 3 1 8 7 2 4

- Não se adapta aos dados, sempre $O(n \log n)$
- Gasto adicional de memória

Quick Sort

QuickSort(list):

left, right := Partition(list)

QuickSort(left)

QuickSort(right)

Partition(list):

pivot := PickPivot(list)

Swap(list[pivot], list[last])

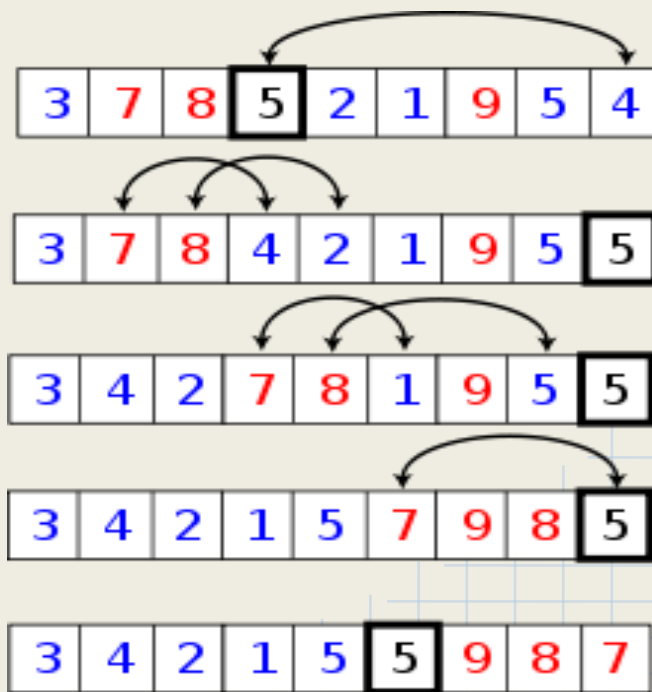
index := 0

For i from 0 to length

 If list[i] < pivotValue

 Swap(list[i], list[index++])

Swap(list[last], list[index])



Quick Sort e Merge Sort

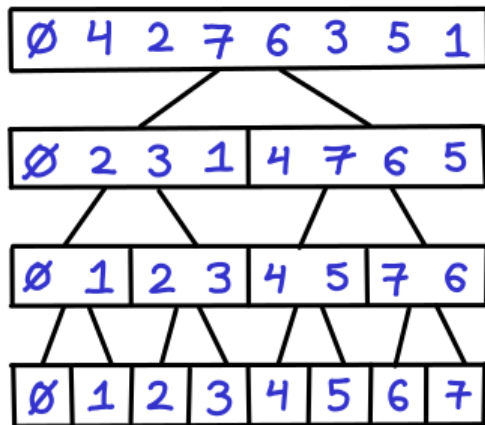
Complexidade de tempo:

- Melhor Caso: $O(n \log n)$
- Pior Caso: $O(n^2)$

Complexidade de tempo:

- Melhor Caso: $O(n \log n)$
- Pior Caso: $O(n \log n)$

QUICKSORT



MERGESORT

