ANNY CAROLINE WALKER SILVA, 1201324404

[ALGORITMO DE ORDENAÇÃO]

BUBBLE SORT

Descrição: Têm se um vetor com N valores, cada valor é comparado com todos os outros valores, de dois em dois, pois se segundo elemento comparado é menor do que o primeiro elemento comparado, então trocam-se suas posições no vetor, e no final, após todos terem sido comparados com todos, temos o vetor ordenado.

Característica: Os menores elementos vão ficando mais próximos da posição inicial, como se fossem bolhas emergindo do fundo.

Pseudo-código:

```
(1) procedimento BubbleSort(A : vetor, N: inteiro)
(2)    para j → 1 até N-1 faça
(3)       para i → 1 até N-1 faça
(4)          se A[i] > A[i+1] então
(5)             aux → A[i];
(6)             A[i] → A[i+1];
(7)            A[i+1] → aux;
(8)          fimse
(9)       fimpara
(10)   fimpara
```

Análise: Independente da ordem do vetor de entrada (já ordenado, ou totalmente invertido, ou alternado/aleatório) o para...até...faça (loop) da linha (2) é executado N-1 vezes, e o loop para...até...faça da linha (3) é executado N-1 vezes para cada para...até...faça da linha (2), concluindo que o loop (3) é executado N^2 vezes.

BubbleSort implementado em C:

```
static int contFor1 = 0; //variável que conta quantas vezes o primeiro FOR foi executado
static int contFor2 = 0; //variável que conta quantas vezes o segundo FOR foi executado
static int contIf = 0; //variável que conta quantas vezes o IF foi adentrado

void bubbleSort (int vetor[], int n) {
   int k, j, aux;

   for (k = 1; k < n; k++) {

                        contFor1 = contFor1 + 1;

      for (j = 0; j < n - 1; j++) {

                                 contFor2 = contFor2 + 1;

         if (vetor[j] > vetor[j + 1]) {

           contIf = contIf + 1;

            aux         = vetor[j];
```

```
            vetor[j]     = vetor[j + 1];
            vetor[j + 1] = aux;

        }
      }
   }

   printf("\nCONTFOR1: %d\n", contFor1);
   printf("CONTFOR2: %d\n", contFor2);
   printf("CONTIF: %d\n", contIf);
}
```

Resultados:

	ORDENADO (EM QUANTIDADE DE ELEMENTOS)			
	10	**100**	**500**	**1000**
FOR EXTERNO	9 ou (n-1)	99 ou (n-1)	499 ou (n-1)	999 ou (n-1)
FOR INTERNO	81 ou $(n-1)^2$	9801 ou $(n-1)^2$	249001 ou $(n-1)^2$	998001 ou $(n-1)^2$
IF	0	0	0	0

Pegando o loop mais custoso e resolvendo $\frac{(n-1)\,n}{2}$ temos $\frac{n^2}{2}$ - $\frac{n}{2}$, desprezando valores ignoráveis (como inteiros), temos:

Melhor caso: O (n^2)

	INVERTIDO (EM QUANTIDADE DE ELEMENTOS)			
	10	**100**	**500**	**1000**
FOR EXTERNO	9 ou (n-1)	99 ou (n-1)	499 ou (n-1)	999 ou (n-1)
FOR INTERNO	81 ou $(n-1)^2$	9801 ou $(n-1)^2$	249001 ou $(n-1)^2$	998001 ou $(n-1)^2$
IF	45	4950	124750	499500

Pegando o loop mais custoso e resolvendo $\frac{(n-1)\,n}{2}$ temos $\frac{n^2}{2}$ - $\frac{n}{2}$, desprezando valores ignoráveis (como inteiros), temos:

Pior caso: O (n^2)

	ALEATÓRIO (EM QUANTIDADE DE ELEMENTOS)			
	10	**100**	**500**	**1000**
FOR EXTERNO	9 ou (n-1)	99 ou (n-1)	499 ou (n-1)	999 ou (n-1)
FOR INTERNO	81 ou $(n-1)^2$	9801 ou $(n-1)^2$	249001 ou $(n-1)^2$	998001 ou $(n-1)^2$

ANNY CAROLINE WALKER SILVA, 1201324404

IF	21	2665	59321	251949

Pegando o loop mais custoso e resolvendo $\frac{(n-1)\,n}{2}$ temos $\frac{n^2}{2}$ - $\frac{n}{2}$, desprezando valores ignoráveis (como inteiros), temos:

Caso médio: O (n^2)

[ALGORITMO DE ORDENAÇÃO]

INSERTION SORT

Descrição: Tem se um vetor com N elementos, a partir do primeiro o elemento, o Insertion Sort compara ele com o segundo, se esse segundo for menor, ele é inserido antes do primeiro elementos, senão, ele permanece onde está, logo, inicia-se a comparação do terceiro elemento, e esse é comparado com todos os elementos antes de si, iniciando pelo primeiro, se o terceiro elemento for menor que o primeiro, então ele é inserido no início, se for menor que algum elemento já verificado de outra posição que não a primeira, ele é inserido antes desse elemento, no final isso resultado em um vetor ordenado, consultando, verificando e inserindo um elemento por vez.

Característica: Parece como algumas pessoas organizam a carta do baralho, comparando da primeira posição até a última e ordenando quando necessário.

Pseudo-código:

```
(1) procedimento InsertionSort(A : vetor, N: inteiro)
(2)    para i → 1 até N-1 faça
(3)        eleito → A[i];
(4)        j → i–1;
(5)        enquanto j>=0 e eleito<A[j] faça
(6)                A[j+1] → A[j];
(7)                j → j-1;
(8)        fimenquanto
(9)    fimpara
```

Análise: Independente da ordem do vetor de entrada (já ordenado, ou totalmente invertido, ou alternado/aleatório) o para...até...faça (loop) da linha (2) é executado N-1 vezes, quanto ao enquanto... da linha (5), temos que ele será executado somente quando houver necessidade, o que no pior caso, resultaria em N^2 execuções.

InsertionSort implementado em C:

```
static int contFor = 0; //variável que conta quantas vezes o primeiro FOR foi executado
static int contWhile = 0; //variável que conta quantas vezes o segundo FOR foi executado

void insertionSort(int vetor[], int n) {
   int i, key, j;
   for (i = 1; i < n; i++) {
      key = vetor[i];
      j = i - 1;
                    contFor1++;
```

ANNY CAROLINE WALKER SILVA, 1201324404

```
        while (j >= 0 && vetor[j] > key) {
            vetor[j + 1] = vetor[j];
            j = j - 1;
            contFor2++;
        }
        vetor[j + 1] = key;
    }

    printf("\nCONTFOR: %d\n", contFor);
    printf("LOOPWHILE: %d\n", contWhile);
}
```

Resultados:

	ORDENADO (EM QUANTIDADE DE ELEMENTOS)			
	10	**100**	**500**	**1000**
FOR EXTERNO	9 ou n-1	99 ou n-1	499 ou n-1	999 ou n-1
WHILE	0	0	0	0

Melhor caso: O (n)

	INVERTIDO (EM QUANTIDADE DE ELEMENTOS)			
	10	**100**	**500**	**1000**
FOR EXTERNO	9 ou n-1	99 ou n-1	499 ou n-1	999 ou n-1
WHILE	45 ou $\frac{(n-1)\,n}{2}$	4950 ou $\frac{(n-1)\,n}{2}$	124750 ou $\frac{(n-1)\,n}{2}$	499500 ou $\frac{(n-1)\,n}{2}$

Resolvendo $\frac{(n-1)\,n}{2}$ temos $\frac{n^2}{2}$ - $\frac{n}{2}$, desprezando valores ignoráveis (como inteiros), temos:

Pior caso: O (n^2)

	ALEATÓRIO (EM QUANTIDADE DE ELEMENTOS)			
	10	**100**	**500**	**1000**
FOR EXTERNO	9 ou n-1	99 ou n-1	499 ou n-1	999 ou n-1
WHILE	21 < $\frac{(n-1)\,n}{2}$	2665 < $\frac{(n-1)\,n}{2}$	59321 < $\frac{(n-1)\,n}{2}$	251949 < $\frac{(n-1)\,n}{2}$

Pegando o loop mais custoso e resolvendo $\frac{(n-1)\,n}{2}$ temos $\frac{n^2}{2}$ - $\frac{n}{2}$, desprezando valores ignoráveis (como inteiros), temos:

Caso médio: O (n^2)

ANNY CAROLINE WALKER SILVA, 1201324404

[ALGORITMO DE ORDENAÇÃO]

SELECTION SORT

Descrição: É um algoritmo de ordenação baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição, e assim é feito sucessivamente com os n-1 elementos restantes, até os últimos dois elementos.

Característica: Muito simples e bruto, sempre seleciona os valores menores e os ordena a partir da esquerda.

Pseudo-código:

```
(1) procedimento SelectionSort(A : vetor, N: inteiro)
(2)   para i → 0 até N-1 faça
(3)       menor → i;
(4)       para j → i+1 ATÉ N faça
(5)             se A[menor] > A[j] então
(6)                   menor → j;
(7)             fimse
(8)       fimpara
(9)       se menor != i então
(10)            aux → A[menor];
(11)            A[menor] → A[i];
(12)            A[i] → aux;
(13)      fimse
(14)      fimpara
```

Análise: Independente da ordem do vetor de entrada (já ordenado, ou totalmente invertido, ou alternado/aleatório) o para...até...faça (loop) da linha (2) é executado N-1 vezes, quanto ao enquanto... da linha (5), temos que ele será executado somente quando houver necessidade, o que no pior caso, resultaria em N^2 execuções.

SelectionSort implementado em C:

```
static int contFor1 = 0; //variável que conta quantas vezes o primeiro FOR foi executado
static int contFor2 = 0; //variável que conta quantas vezes o segundo FOR foi executado
static int contIf = 0; //variável que conta quantas vezes o segundo IF foi executado
static int contIf2 = 0; //variável que conta quantas vezes o segundo IF foi executado

void selectionSort(int vetor[], int n) {
  int i, j, min, aux;
  for (i = 0; i < (n-1); i++)
  {
          contFor1++;
    min = i;
    for (j = (i+1); j < n; j++) {
          contFor2++;
      if(vetor[j] < vetor[min]) {
                              contIf++;
        min = j;
          }
           }
```

```
    if (vetor[i] != vetor[min]) {
            contIf2++;
      aux = vetor[i];
      vetor[i] = vetor[min];
      vetor[min] = aux;
    }
  }

            printf("\ContForExterno: %d\n", contFor1);
            printf("\nContForInterno: %d\n", contFor2);
            printf("\ncontIf1: %d\n", contIf);
            printf("\ncontIf2: %d\n", contIf2);
}
```

Resultados:

	ORDENADO (EM QUANTIDADE DE ELEMENTOS)			
	10	**100**	**500**	**1000**
FOR EXTERNO	9 ou n-1	99 ou n-1	499 ou n-1	999 ou n-1
FOR INTERNO	45 ou $\frac{(n-1)\,n}{2}$	4950 ou $\frac{(n-1)\,n}{2}$	124750 ou $\frac{(n-1)\,n}{2}$	499500 ou $\frac{(n-1)\,n}{2}$
IF INTERNO	0	0	0	0
IF EXTERNO	0	0	0	0

Resolvendo $\frac{(n-1)\,n}{2}$ temos $\frac{n^2}{2}$ - $\frac{n}{2}$, desprezando valores ignoráveis (como inteiros), temos:

Melhor caso: O (n^2)

	INVERTIDO (EM QUANTIDADE DE ELEMENTOS)			
	10	**100**	**500**	**1000**
FOR EXTERNO	9 ou n-1	99 ou n-1	499 ou n-1	999 ou n-1
FOR INTERNO	45 ou $\frac{(n-1)\,n}{2}$	4950 ou $\frac{(n-1)\,n}{2}$	124750 ou $\frac{(n-1)\,n}{2}$	499500 ou $\frac{(n-1)\,n}{2}$
IF INTERNO	25 ou $\frac{n}{4} * n$	2500 ou $\frac{n}{4} * n$	62500 ou $\frac{n}{4} * n$	250000 ou $\frac{n}{4} * n$
IF EXTERNO	5 ou $\frac{n}{2}$	50 ou $\frac{n}{2}$	250 ou $\frac{n}{2}$	500 ou $\frac{n}{2}$

Pegando o loop mais custoso e resolvendo $\frac{(n-1)\,n}{2}$ temos $\frac{n^2}{2}$ - $\frac{n}{2}$, desprezando valores ignoráveis (como inteiros), temos:

Pior caso: O (n^2)

ANNY CAROLINE WALKER SILVA, 1201324404

	ALEATÓRIO (EM QUANTIDADE DE ELEMENTOS)			
	10	**100**	**500**	**1000**
FOR EXTERNO	9 ou n-1	99 ou n-1	499 ou n-1	999 ou n-1
WHILE	45 ou $\frac{(n-1)\,n}{2}$	4950 ou $\frac{(n-1)\,n}{2}$	124750 ou $\frac{(n-1)\,n}{2}$	499500 ou $\frac{(n-1)\,n}{2}$
IF INTERNO	13	305	2262	5442
IF EXTERNO	7	95	493	991

Pegando o loop mais custoso e resolvendo $\frac{(n-1)\,n}{2}$ temos $\frac{n^2}{2}$ - $\frac{n}{2}$, desprezando valores ignoráveis (como inteiros), temos:

Caso médio: O (n^2)

RESULTADOS GERAIS:

	ALGORITMOS DE ORDENAÇÃO		
	BubbleSort	**InsertionSort**	**SelectionSort**
MELHOR CASO	**O (n^2)**	**O (n)**	**O (n^2)**
PIOR CASO	**O (n^2)**	**O (n^2)**	**O (n^2)**
CASO MÉDIO	**O (n^2)**	**O (n^2)**	**O (n^2)**