Nomes: Pedro Bergamin e Rebeca Gomes

ESPECIFICAÇÃO DO PROBLEMA

`selection_sort:`

A ordenação por seleção ou selection sort consiste em selecionar o menor item e colocar na primeira posição, selecionar o segundo menor item e colocar na segunda posição, segue estes passos até que reste um único elemento. Para todos os casos (melhor, médio e pior caso) possui complexidade $C(n) = O(n^2)$ e não é um algoritmo estável.

ENTRADAS:

```
void selection_sort(int *array, int size)
```

```
* @param array -> Array to be sorted
```

```
* @param size -> Tamanho do array
```

`insertion_sort`

Insertion Sort ou ordenação por inserção é o método que percorre um vetor de elementos da esquerda para a direita e à medida que avança vai ordenando os elementos à esquerda. Possui complexidade $C(n) = O(n)$ no melhor caso e $C(n) = O(n^2)$ no caso médio e pior caso. É considerado um método de ordenação estável.

ENTRADAS:

```
void insertion_sort(int *array, int size)
```

```
* @param array -> Array to be sorted
```

```
* @param size -> Tamanho do array
```

`shell_sort`

O método Shell Sort é uma extensão do algoritmo de ordenação por inserção. Ele permite a troca de registros distantes um do outro, diferente do algoritmo de ordenação por inserção que possui a troca de itens adjacentes para determinar o ponto de inserção. A complexidade do algoritmo é desconhecida, ninguém ainda foi capaz de encontrar uma fórmula fechada para sua função de complexidade e o método não é estável.

ENTRADAS:

```
void shell_sort(int *array, int size)
```

```
* @param array -> Array to be sorted
```

```
* @param size -> Tamanho do array
```

quick_sort

Provavelmente é o mais utilizado. Possui complexidade C(n) = O(n²) no pior caso e C(n) = O(n log n) no melhor e médio caso e não é um algoritmo estável. Basicamente a operação do algoritmo pode ser resumida na seguinte estratégia: divide sua lista de entrada em duas sub-listas a partir de um pivô, para em seguida realizar o mesmo procedimento nas duas listas menores até uma lista unitária.

ENTRADAS:

```
        void quick_sort(int array[], int left, int right)
* @param array -> Array to be sorted
* @param left -> Elemento Pivô
* @param right -> Tamanho do array
```

heap_sort

A classificação de heap é uma técnica de classificação baseada em comparação baseada na estrutura de dados Binary Heap. É semelhante à ordenação por seleção, onde primeiro encontramos o elemento máximo e colocamos o elemento máximo no final. Repetimos o mesmo processo para os elementos restantes.

ENTRADAS:

```
        void heap_sort(int array[], int n)
* @param a -> Array to be sorted
* @param n -> Tamanho do array
```

merge_sort

Esse algoritmo divide o problema em pedaços menores, resolve cada pedaço e depois junta (merge) os resultados. O vetor será dividido em duas partes iguais, que serão cada uma divididas em duas partes, e assim até ficar um ou dois elementos cuja ordenação é trivial.

ENTRADAS:

```
        void merge_sort(int*a, int n)
* @param a -> Array to be sorted
* @param n -> Tamanho do array
```

OS TESTES ABAIXO VALEM PARA TODAS AS SITUAÇÕES DE SORT ACIMA, INCLUINDO O QUICK SORT, POIS POR DEFAULT SEU PIVÔ ESTÁ EM “0”.

CLASSE DE EQUIVALÊNCIA			
VARIAVEL DE ENTRADA		CLASSES DE EQUIVALÊNCIA VÁLIDAS	CLASSES DE EQUIVALÊNCIA INVÁLIDAS
VETOR NÃO ESTÁ VAZIO	array	Contém números inteiros	Não contém números inteiros
		Pode ser uma lista vazia	
TIPO DO VETOR		É um número inteiro finito	Não é um número inteiro finito
TAMANHO DO VETOR	size	size >= 0	size < 0

CASOS DE TESTES DA CLASSE DE EQUIVALÊNCIA	
VARIAVEL DE ENTRADA	SAIDA
{“3,5,6,3,1”, 5}	VÁLIDO, {1,3,3,5,6}
{“a,d,g,f ”, 4}	INVÁLIDO, {type error}, deve ser um vetor de números inteiros
{“2.4,1.2”, 2}	INVÁLIDO, {type error}, deve ser um vetor de números inteiros
{“3,5,6,3,1”, }	INVALIDO, {size error}, não possui um número
{“3,5,6,3,1”, -1}	INVALIDO, {size error}, não é maior ou igual a 0

CASOS DE TESTE DA CLASSE DE VALOR LIMITE	
VARIAVEL DE ENTRADA	SAIDA
{“”, 0}	VÁLIDO, { }
{“1,2,3,4,5”, 5}	VÁLIDO, {1,2,3,4,5}
{“3,3,3,1,1”, 5}	VÁLIDO, {1,1,3,3,3}
{“3,1,1,1,3 ”, 5}	VÁLIDO, {1,1,1,3,3}
{“3,1,1,1,3,..... ”, N(∞)}	INVÁLIDO, {size error}
{“3.3,1,1,1,3 ”, 5}	INVÁLIDO, {type error}

CASOS DE TESTES SISTEMÁTICO	
VARIAVEL DE ENTRADA	SAIDA
{“3,5,6,3,1”, 5}	VÁLIDO, {2,3,3,5,6}
{“3, -5,6,3,1”, 5}	VÁLIDO, {-5,2,3,3,6}
{“”, 0}	VÁLIDO, {}
{“2”, 1}	VÁLIDO, {2}
{“9,4,2,1 ”, 2}	VÁLIDO, {4,9,2,1}
{“3,3,3,1,1”, 5}	VÁLIDO, {1,1,3,3,3}
{“3,1,1,1,3 ”, 5}	VÁLIDO, {1,1,1,3,3}
{“3.2,5.5,6.2,3.1,1”, 5}	INVÁLIDO, {type error}
{“A,S,B,H”, 4}	INVÁLIDO, {type error}
{“4,3,1”, -1}	INVALIDO, {size error}
{“4,3,1”, 5}	INVALIDO, {size error}
{“3,5,6,3,1”, }	INVALIDO, {size error}
{“3,5,6,3,A”, }	INVALIDO, {type error} E {size error}