PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA

TRABALHO PRÁTICO 1 — TP1
TESTE E CONFIABILIDADE DE SISTEMAS

ALUNOS: DIEGO HENRIQUE SILVA OLIVEIRA E LEONARDO BARBOSA DA ROSA

PORTO ALEGRE, OUTUBRO DE 2022

CONTEÚDO

- ➡️ Classes de equivalência válidas e inválidas
- ➡️ 42 testes para cada algoritmo de ordenação
- ➡️ 270 testes funcionais
- ➡️ 336 testes no total
- ➡️ Análise de cobertura de código (*code coverage*)

CLASSES DE EQUIVALÊNCIA

Variável de Entrada	Classes de Equivalência Válidas	Classes de Equivalência Inválidas
a	*a* ser um ponteiro para um vetor de inteiros	*a* não ser um ponteiro; *a* ser um ponteiro para um vetor de qualquer outro tipo (char, float, double, struct, etc.).
length	*length* ser do tipo *int* && satisfazer às condições abaixo: 2 ≤ *length* ≤ 20 *&& length* ser igual ao tamanho do vetor apontado por a	*length* não ser do tipo *int*; *length* < 2 || *length* > 20; *length* ser diferente do tamanho do vetor apontado por *a*.
type	*type* ser um ponteiro para *char* && ser igual a um dos valores abaixo: *type* == “On" *type* == "On2" *type* == *“*Onlogn"	*type* não ser um ponteiro; *type* não ser um ponteiro para *char;* *type* ser diferente de “On”, “On2" e “Onlogn”.
algorithm	*algorithm* ser do tipo *int* && ser igual a um dos valores abaixo: *algorithm* == COUNTING (ou seu equivalente: *algorithm == 0*) *algorithm* == RADIX (ou seu equivalente: *algorithm == 1*) *algorithm* == BUBBLE (ou seu equivalente: *algorithm == 2*) *algorithm* == INSERTION (ou seu equivalente: *algorithm == 3*) *algorithm* == SELECTION (ou seu equivalente: *algorithm == 4*) *algorithm* == HEAP (ou seu equivalente: *algorithm == 5*) *algorithm* == MERGE (ou seu equivalente: *algorithm == 6*) *algorithm* == QUICK (ou seu equivalente: *algorithm == 7*)	*algorithm* não ser do tipo *int*; *algorithm* ser um *enum* inválido, ou seja, diferente de COUNTING, RADIX, BUBBLE, INSERTION, SELECTION, HEAP, MERGE e QUICK; *algorithm* ser igual a um inteiro não definido na sequência do *enum* do código original.

COUNTING SORT — TESTES

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
0	CountingSortOnArrayCheck	[4, 5, 3, 1, 2]	5	“On"	COUNTING	*a* == [1, 2, 3, 4, 5]
1	CountingSortOn2ArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	COUNTING	*a* == [4, 5, 3, 1, 2]
2	CountingSortOnlognArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	COUNTING	*a* == [4, 5, 3, 1, 2]
3	CountingSortOnFuncReturn	[4, 5, 3, 1, 2]	5	“On"	COUNTING	retorno de sort == 0
4	CountingSortOn2FuncReturn	[4, 5, 3, 1, 2]	5	“On2”	COUNTING	retorno de sort == 1
5	CountingSortOnlognFuncReturn	[4, 5, 3, 1, 2]	5	“Onlogn”	COUNTING	retorno de sort == 1
6	CountingSortOnLowercased	[4, 5, 3, 1, 2]	5	“on"	COUNTING	retorno de sort == 1
7	CountingSortOnLowercasedArrayCheck	[4, 5, 3, 1, 2]	5	“on"	COUNTING	*a* == [4, 5, 3, 1, 2]
8	CountingSortOnUppercased	[4, 5, 3, 1, 2]	5	“ON”	COUNTING	retorno de sort == 1
9	CountingSortOnUppercasedArrayCheck	[4, 5, 3, 1, 2]	5	“ON”	COUNTING	*a* == [4, 5, 3, 1, 2]
10	CountingSortOnToggled	[4, 5, 3, 1, 2]	5	“oN”	COUNTING	retorno de sort == 1
11	CountingSortOnToggledArrayCheck	[4, 5, 3, 1, 2]	5	“oN”	COUNTING	*a* == [4, 5, 3, 1, 2]
12	CountingSortOnWithSpace	[4, 5, 3, 1, 2]	5	“On “	COUNTING	retorno de sort == 1
13	CountingSortOnWithSpaceArrayCheck	[4, 5, 3, 1, 2]	5	“On “	COUNTING	*a* == [4, 5, 3, 1, 2]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
14	CountingSortOnEmpty	[4, 5, 3, 1, 2]	5	“"	COUNTING	retorno de sort == 1
15	CountingSortOnEmptyArrayCheck	[4, 5, 3, 1, 2]	5	“"	COUNTING	*a* == [4, 5, 3, 1, 2]
16	CountingSortOnLowerBound	[8, 2]	2	“On"	COUNTING	retorno de sort == 0
17	CountingSortOnLowerBoundArrayCheck	[8, 2]	2	“On"	COUNTING	*a* == [2, 8]
18	CountingSortOnUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“On"	COUNTING	retorno de sort == 0
19	CountingSortOnUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“On"	COUNTING	a == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
20	CountingSortOnInvalidLowerBound	[8]	1	“On”	COUNTING	retorno de sort == 1
21	CountingSortOnInvalidUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“On”	COUNTING	retorno de sort == 1
22	CountingSortOnInvalidUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“On"	COUNTING	Mesmo vetor de entrada, sem modificação

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
23	CountingSortOnLengthZero	[ ]	0	“On"	COUNTING	retorno de sort == 1
24	CountingSortOnNullArray	NULL	0	“On”	COUNTING	retorno de sort == 1
25	CountingSortOnMuchBiggerUpperBound	Vetor com 100 elementos fora de ordem	100	“On”	COUNTING	retorno de sort == 1
26	CountingSortOnMuchBiggerUpperBoundArrayCheck	Vetor com 100 elementos fora de ordem	100	“On"	100	Mesmo vetor de entrada, sem modificação
27	CountingSortInvalidAlgorithm	[4, 5, 3, 1, 2]	5	“On"	100	retorno de sort == 1
28	CountingSortInvalidAlgorithmArrayCheck	[4, 5, 3, 1, 2]	5	“On"	100	*a* == [4, 5, 3, 1, 2]
29	CountingSortInvalidAlgorithm2	[4, 5, 3, 1, 2]	5	“On"	-1	retorno de sort == 1
30	CountingSortInvalidAlgorithm2ArrayCheck	[4, 5, 3, 1, 2]	5	“On"	-1	*a* == [4, 5, 3, 1, 2]

COUNTING SORT
TESTES COM ERRO NA EXECUÇÃO

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
31	CountingSortOnUpperBoundContentsArrayCheck	[2147483447, 2044483647, 56237, 21, 0, 2147483647, 13131313, 1947483647, 13, 21474647]	10	“On"	COUNTING	a == [0, 13, 21, 56237, 13131313, 21474647, 1947483647, 2044483647, 2147483447, 2147483647]
32	CountingSortOnBiggerThanUpperBoundContentsArrayCheck	[2147483648, 2147483646, 2147483647]	3	“On”	COUNTING	a == [-2147483648, 2147483646, 2147483647]
33	CountingSortOnFloatArrayCheck	[8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]	18	“On”	COUNTING	a == [8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]
34	CountingSortOnNegAndPosValuesArrayCheck	[5, -2, 3, 0, -1, 2, 1, 4, -3]	9	“On"	COUNTING	a == [-3, -2, -1, 0, 1, 2, 3, 4, 5]
35	CountingSortOnNegativeValuesArrayCheck	[-8, -18, -1, -5, -2, -19, -10, -4, -3, -15, -11, -6, -14, -7, -9, -17, -12, -13, -16]	19	“On"	COUNTING	a == [-19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
36	CountingSortOnNullString	[4, 5, 3, 1, 2]	5	NULL	COUNTING	retorno de sort == 1
37	CountingSortOnNullStringArrayCheck	[4, 5, 3, 1, 2]	5	NULL	COUNTING	a == [4, 5, 3, 1, 2]
38	CountingSortOnNegativeLength	[4, 5, 3, 1, 2]	-1	“On"	COUNTING	retorno de sort == 1
39	CountingSortOnNegativeLengthArrayCheck	[4, 5, 3, 1, 2]	-1	“On”	COUNTING	a == [4, 5, 3, 1, 2]
40	CountingSortOnNullArrayWithInvalidLength	NULL	5	“On”	COUNTING	retorno de sort == 1
41	CountingSortOnNullArrayWithInvalidLengthArrayCheck	NULL	5	“On"	COUNTING	a == NULL

RADIX SORT — TESTES

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
42	RadixSortOnArrayCheck	[4, 5, 3, 1, 2]	5	“On"	RADIX	*a* == [1, 2, 3, 4, 5]
43	RadixSortOn2ArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	RADIX	*a* == [4, 5, 3, 1, 2]
44	RadixSortOnlognArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	RADIX	*a* == [4, 5, 3, 1, 2]
45	RadixSortOnFuncReturn	[4, 5, 3, 1, 2]	5	“On"	RADIX	retorno de sort == 0
46	RadixSortOn2FuncReturn	[4, 5, 3, 1, 2]	5	“On2”	RADIX	retorno de sort == 1
47	RadixSortOnlognFuncReturn	[4, 5, 3, 1, 2]	5	“Onlogn”	RADIX	retorno de sort == 1
48	RadixSortOnLowercased	[4, 5, 3, 1, 2]	5	“on"	RADIX	retorno de sort == 1
49	RadixSortOnLowercasedArrayCheck	[4, 5, 3, 1, 2]	5	“on"	RADIX	*a* == [4, 5, 3, 1, 2]
50	RadixSortOnUppercased	[4, 5, 3, 1, 2]	5	“ON”	RADIX	retorno de sort == 1
51	RadixSortOnUppercasedArrayCheck	[4, 5, 3, 1, 2]	5	“ON”	RADIX	*a* == [4, 5, 3, 1, 2]
52	RadixSortOnToggled	[4, 5, 3, 1, 2]	5	“oN”	RADIX	retorno de sort == 1
53	RadixSortOnToggledArrayCheck	[4, 5, 3, 1, 2]	5	“oN”	RADIX	*a* == [4, 5, 3, 1, 2]
54	RadixSortOnWithSpace	[4, 5, 3, 1, 2]	5	“On “	RADIX	retorno de sort == 1
55	RadixSortOnWithSpaceArrayCheck	[4, 5, 3, 1, 2]	5	“On “	RADIX	*a* == [4, 5, 3, 1, 2]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
56	RadixSortOnEmpty	[4, 5, 3, 1, 2]	5	“"	RADIX	retorno de sort == 1
57	RadixSortOnEmptyArrayCheck	[4, 5, 3, 1, 2]	5	“"	RADIX	*a* == [4, 5, 3, 1, 2]
58	RadixSortOnLowerBound	[8, 2]	2	“On"	RADIX	retorno de sort == 0
59	RadixSortOnLowerBoundArrayCheck	[8, 2]	2	“On"	RADIX	*a* == [2, 8]
60	RadixSortOnUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“On"	RADIX	retorno de sort == 0
61	RadixSortOnUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“On"	RADIX	a == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
62	RadixSortOnInvalidLowerBound	[8]	1	“On”	RADIX	retorno de sort == 1
63	RadixSortOnInvalidUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“On”	RADIX	retorno de sort == 1
64	RadixSortOnInvalidUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“On"	RADIX	Mesmo vetor de entrada, sem modificação

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
65	RadixSortOnLengthZero	[ ]	0	“On"	RADIX	retorno de sort == 1
66	RadixSortOnNullArray	NULL	0	“On”	RADIX	retorno de sort == 1
67	RadixSortOnMuchBiggerUpperBound	Vetor com 100 elementos fora de ordem	100	“On”	RADIX	retorno de sort == 1
68	RadixSortOnMuchBiggerUpperBoundArrayCheck	Vetor com 100 elementos fora de ordem	100	“On"	100	Mesmo vetor de entrada, sem modificação
69	RadixSortInvalidAlgorithm	[4, 5, 3, 1, 2]	5	“On"	100	retorno de sort == 1
70	RadixSortInvalidAlgorithmArrayCheck	[4, 5, 3, 1, 2]	5	“On"	100	*a* == [4, 5, 3, 1, 2]
71	RadixSortInvalidAlgorithm2	[4, 5, 3, 1, 2]	5	“On"	-1	retorno de sort == 1
72	RadixSortInvalidAlgorithm2ArrayCheck	[4, 5, 3, 1, 2]	5	“On"	-1	*a* == [4, 5, 3, 1, 2]

RADIX SORT
TESTES COM ERRO NA EXECUÇÃO

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
73	RadixSortOnBiggerThanUpperBoundContentsArrayCheck	[2147483648, 2147483646, 2147483647]	3	“On”	RADIX	a == [-2147483648, 2147483646, 2147483647]
74	RadixSortOnNegAndPosValuesArrayCheck	[5, -2, 3, 0, -1, 2, 1, 4, -3]	9	“On"	RADIX	a == [-3, -2, -1, 0, 1, 2, 3, 4, 5]
75	RadixSortOnNegativeValuesArrayCheck	[-8, -18, -1, -5, -2, -19, -10, -4, -3, -15, -11, -6, -14, -7, -9, -17, -12, -13, -16]	19	“On"	RADIX	a == [-19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1]
76	RadixSortOnNullString	[4, 5, 3, 1, 2]	5	NULL	RADIX	retorno de sort == 1
77	RadixSortOnNullStringArrayCheck	[4, 5, 3, 1, 2]	5	NULL	RADIX	a == [4, 5, 3, 1, 2]
78	RadixSortOnNegativeLength	[4, 5, 3, 1, 2]	-1	“On"	RADIX	retorno de sort == 1
79	RadixSortOnNegativeLengthArrayCheck	[4, 5, 3, 1, 2]	-1	“On”	RADIX	a == [4, 5, 3, 1, 2]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
80	RadixSortOnNullArrayWithInvalidLength	NULL	5	“On”	RADIX	retorno de sort == 1
81	RadixSortOnNullArrayWithInvalidLength ArrayCheck	NULL	5	“On"	RADIX	a == NULL
82	RadixSortOnUpperBoundContentsArray Check	[2147483447, 2044483647, 56237, 21, 0, 2147483647, 13131313, 1947483647, 13, 21474647]	10	“On"	RADIX	a == [0, 13, 21, 56237, 13131313, 21474647, 1947483647, 2044483647, 2147483447, 2147483647]
83	RadixSortOnFloatArrayCheck	[8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]	18	“On"	RADIX	a == [8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]

BUBBLE SORT — TESTES

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
84	BubbleSortOn2ArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	BUBBLE	*a* == [1, 2, 3, 4, 5]
85	BubbleSortOnArrayCheck	[4, 5, 3, 1, 2]	5	“On”	BUBBLE	*a* == [4, 5, 3, 1, 2]
86	BubbleSortOnlognArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	BUBBLE	*a* == [4, 5, 3, 1, 2]
87	BubbleSortOn2FuncReturn	[4, 5, 3, 1, 2]	5	“On2”	BUBBLE	retorno de sort == 0
88	BubbleSortOnFuncReturn	[4, 5, 3, 1, 2]	5	“On”	BUBBLE	retorno de sort == 1
89	BubbleSortOnlognFuncReturn	[4, 5, 3, 1, 2]	5	“Onlogn”	BUBBLE	retorno de sort == 1
90	BubbleSortOn2Lowercased	[4, 5, 3, 1, 2]	5	“on2”	BUBBLE	retorno de sort == 1
91	BubbleSortOn2LowercasedArrayCheck	[4, 5, 3, 1, 2]	5	“on2”	BUBBLE	*a* == [4, 5, 3, 1, 2]
92	BubbleSortOn2Uppercased	[4, 5, 3, 1, 2]	5	“ON2”	BUBBLE	retorno de sort == 1
93	BubbleSortOn2UppercasedArrayCheck	[4, 5, 3, 1, 2]	5	“ON2”	BUBBLE	*a* == [4, 5, 3, 1, 2]
94	BubbleSortOn2Toggled	[4, 5, 3, 1, 2]	5	“oN2”	BUBBLE	retorno de sort == 1
95	BubbleSortOn2ToggledArrayCheck	[4, 5, 3, 1, 2]	5	“oN2”	BUBBLE	*a* == [4, 5, 3, 1, 2]
96	BubbleSortOn2WithSpace	[4, 5, 3, 1, 2]	5	“On2 “	BUBBLE	retorno de sort == 1
97	BubbleSortOn2WithSpaceArrayCheck	[4, 5, 3, 1, 2]	5	“On2 “	BUBBLE	*a* == [4, 5, 3, 1, 2]

		Casos de Teste				
Número do Teste	**Nome do Teste**	**Entrada**				**Saída Esperada**
		a	***length***	***type***	***algorithm***	
98	BubbleSortOn2Empty	[4, 5, 3, 1, 2]	5	“"	BUBBLE	retorno de sort == 1
99	BubbleSortOn2EmptyArrayCheck	[4, 5, 3, 1, 2]	5	“"	BUBBLE	*a* == [4, 5, 3, 1, 2]
100	BubbleSortOn2LowerBound	[8, 2]	2	“On2”	BUBBLE	retorno de sort == 0
101	BubbleSortOn2LowerBoundArrayCheck	[8, 2]	2	“On2”	BUBBLE	*a* == [2, 8]
102	BubbleSortOn2UpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“On2”	BUBBLE	retorno de sort == 0
103	BubbleSortOn2UpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“On2”	BUBBLE	a == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
104	BubbleSortOn2InvalidLowerBound	[8]	1	“On2”	BUBBLE	retorno de sort == 1
105	BubbleSortOn2InvalidUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“On2”	BUBBLE	retorno de sort == 1
106	BubbleSortOn2InvalidUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“On2”	BUBBLE	Mesmo vetor de entrada, sem modificação

		Casos de Teste				
Número do Teste	**Nome do Teste**	**Entrada**				**Saída Esperada**
		a	***length***	***type***	***algorithm***	
107	BubbleSortOn2LengthZero	[ ]	0	“On2”	BUBBLE	retorno de sort == 1
108	BubbleSortOn2NullArray	NULL	0	“On2”	BUBBLE	retorno de sort == 1
109	BubbleSortOn2MuchBiggerUpperBound	Vetor com 100 elementos fora de ordem	100	“On2”	BUBBLE	retorno de sort == 1
110	BubbleSortOn2MuchBiggerUpperBoundArrayCheck	Vetor com 100 elementos fora de ordem	100	“On2”	100	Mesmo vetor de entrada, sem modificação
111	BubbleSortInvalidAlgorithm	[4, 5, 3, 1, 2]	5	“On2”	100	retorno de sort == 1
112	BubbleSortInvalidAlgorithmArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	100	*a* == [4, 5, 3, 1, 2]
113	BubbleSortInvalidAlgorithm2	[4, 5, 3, 1, 2]	5	“On2”	-1	retorno de sort == 1
114	BubbleSortInvalidAlgorithm2ArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	-1	*a* == [4, 5, 3, 1, 2]
115	BubbleSortOn2UpperBoundContentsArrayCheck	[2147483447, 2044483647, 56237, 21, 0, 2147483647, 13131313, 1947483647, 13, 21474647]	10	“On2”	BUBBLE	a == [0, 13, 21, 56237, 13131313, 21474647, 1947483647, 2044483647, 2147483447, 2147483647]
116	BubbleSortOn2BiggerThanUpperBoundContentsArrayCheck	[2147483648, 2147483646, 2147483647]	3	“On2”	BUBBLE	a == [-2147483648, 2147483646, 2147483647]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	***length***	***type***	***algorithm***	
117	BubbleSortOn2NegAndPosValuesArrayCheck	[5, -2, 3, 0, -1, 2, 1, 4, -3]	9	“On2”	BUBBLE	a == [-3, -2, -1, 0, 1, 2, 3, 4, 5]
118	BubbleSortOn2NegativeValuesArrayCheck	[-8, -18, -1, -5, -2, -19, -10, -4, -3, -15, -11, -6, -14, -7, -9, -17, -12, -13, -16]	19	“On2”	BUBBLE	a == [-19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1]

BUBBLE SORT
TESTES COM ERRO NA EXECUÇÃO

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
119	BubbleSortOn2NullString	[4, 5, 3, 1, 2]	5	NULL	BUBBLE	retorno de sort == 1
120	BubbleSortOn2NullStringArrayCheck	[4, 5, 3, 1, 2]	5	NULL	BUBBLE	a == [4, 5, 3, 1, 2]
121	BubbleSortOn2NegativeLength	[4, 5, 3, 1, 2]	-1	“On2”	BUBBLE	retorno de sort == 1
122	BubbleSortOn2NegativeLengthArrayCheck	[4, 5, 3, 1, 2]	-1	“On2”	BUBBLE	a == [4, 5, 3, 1, 2]
123	BubbleSortOn2NullArrayWithInvalidLength	NULL	5	“On2”	BUBBLE	retorno de sort == 1
124	BubbleSortOn2NullArrayWithInvalidLengthArrayCheck	NULL	5	“On2”	BUBBLE	a == NULL
125	BubbleSortOn2FloatArrayCheck	[8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]	18	“On2”	BUBBLE	a == [8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]

INSERTION SORT — TESTES

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
126	InsertionSortOn2ArrayCheck	[4, 5, 3, 1, 2]	5	“On2"	INSERTION	*a* == [1, 2, 3, 4, 5]
127	InsertionSortOnArrayCheck	[4, 5, 3, 1, 2]	5	“On”	INSERTION	*a* == [4, 5, 3, 1, 2]
128	InsertionSortOnlognArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	INSERTION	*a* == [4, 5, 3, 1, 2]
129	InsertionSortOn2FuncReturn	[4, 5, 3, 1, 2]	5	“On2"	INSERTION	retorno de sort == 0
130	InsertionSortOnFuncReturn	[4, 5, 3, 1, 2]	5	“On”	INSERTION	retorno de sort == 1
131	InsertionSortOnlognFuncReturn	[4, 5, 3, 1, 2]	5	“Onlogn”	INSERTION	retorno de sort == 1
132	InsertionSortOn2Lowercased	[4, 5, 3, 1, 2]	5	“on2”	INSERTION	retorno de sort == 1
133	InsertionSortOn2LowercasedArrayCheck	[4, 5, 3, 1, 2]	5	“on2”	INSERTION	*a* == [4, 5, 3, 1, 2]
134	InsertionSortOn2Uppercased	[4, 5, 3, 1, 2]	5	“ON2”	INSERTION	retorno de sort == 1
135	InsertionSortOn2UppercasedArrayCheck	[4, 5, 3, 1, 2]	5	“ON2”	INSERTION	*a* == [4, 5, 3, 1, 2]
136	InsertionSortOn2Toggled	[4, 5, 3, 1, 2]	5	“oN2”	INSERTION	retorno de sort == 1
137	InsertionSortOn2ToggledArrayCheck	[4, 5, 3, 1, 2]	5	“oN2”	INSERTION	*a* == [4, 5, 3, 1, 2]
138	InsertionSortOn2WithSpace	[4, 5, 3, 1, 2]	5	“On2 “	INSERTION	retorno de sort == 1
139	InsertionSortOn2WithSpaceArrayCheck	[4, 5, 3, 1, 2]	5	“On2 “	INSERTION	*a* == [4, 5, 3, 1, 2]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
140	InsertionSortOn2Empty	[4, 5, 3, 1, 2]	5	“"	INSERTION	retorno de sort == 1
141	InsertionSortOn2EmptyArrayCheck	[4, 5, 3, 1, 2]	5	“"	INSERTION	*a* == [4, 5, 3, 1, 2]
142	InsertionSortOn2LowerBound	[8, 2]	2	“On2”	INSERTION	retorno de sort == 0
143	InsertionSortOn2LowerBoundArrayCheck	[8, 2]	2	“On2”	INSERTION	*a* == [2, 8]
144	InsertionSortOn2UpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“On2”	INSERTION	retorno de sort == 0
145	InsertionSortOn2UpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“On2”	INSERTION	a == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
146	InsertionSortOn2InvalidLowerBound	[8]	1	“On2”	INSERTION	retorno de sort == 1
147	InsertionSortOn2InvalidUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“On2”	INSERTION	retorno de sort == 1
148	InsertionSortOn2InvalidUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“On2”	INSERTION	Mesmo vetor de entrada, sem modificação

		Casos de Teste				
Número do Teste	**Nome do Teste**	**Entrada**				**Saída Esperada**
		a	***length***	***type***	***algorithm***	
149	InsertionSortOn2LengthZero	[ ]	0	“On2”	INSERTION	retorno de sort == 1
150	InsertionSortOn2NullArray	NULL	0	“On2”	INSERTION	retorno de sort == 1
151	InsertionSortOn2MuchBiggerUpperBound	Vetor com 100 elementos fora de ordem	100	“On2”	INSERTION	retorno de sort == 1
152	InsertionSortOn2MuchBiggerUpperBoundArrayCheck	Vetor com 100 elementos fora de ordem	100	“On2”	100	Mesmo vetor de entrada, sem modificação
153	InsertionSortInvalidAlgorithm	[4, 5, 3, 1, 2]	5	“On2”	100	retorno de sort == 1
154	InsertionSortInvalidAlgorithmArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	100	*a* == [4, 5, 3, 1, 2]
155	InsertionSortInvalidAlgorithm2	[4, 5, 3, 1, 2]	5	“On2”	-1	retorno de sort == 1
156	InsertionSortInvalidAlgorithm2ArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	-1	*a* == [4, 5, 3, 1, 2]
157	InsertionSortOn2UpperBoundContentsArrayCheck	[2147483447, 2044483647, 56237, 21, 0, 2147483647, 13131313, 1947483647, 13, 21474647]	10	“On2”	INSERTION	a == [0, 13, 21, 56237, 13131313, 21474647, 1947483647, 2044483647, 2147483447, 2147483647]
158	InsertionSortOn2BiggerThanUpperBoundContentsArrayCheck	[2147483648, 2147483646, 2147483647]	3	“On2”	INSERTION	a == [-2147483648, 2147483646, 2147483647]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
159	InsertionSortOn2NegAndPosValuesArrayCheck	[5, -2, 3, 0, -1, 2, 1, 4, -3]	9	“On2”	INSERTION	a == [-3, -2, -1, 0, 1, 2, 3, 4, 5]
160	InsertionSortOn2NegativeValuesArrayCheck	[-8, -18, -1, -5, -2, -19, -10, -4, -3, -15, -11, -6, -14, -7, -9, -17, -12, -13, -16]	19	“On2”	INSERTION	a == [-19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1]

INSERTION SORT
TESTES COM ERRO NA EXECUÇÃO

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
161	InsertionSortOn2NullString	[4, 5, 3, 1, 2]	5	NULL	INSERTION	retorno de sort == 1
162	InsertionSortOn2NullStringArrayCheck	[4, 5, 3, 1, 2]	5	NULL	INSERTION	a == [4, 5, 3, 1, 2]
163	InsertionSortOn2NegativeLength	[4, 5, 3, 1, 2]	-1	“On2”	INSERTION	retorno de sort == 1
164	InsertionSortOn2NegativeLengthArrayCheck	[4, 5, 3, 1, 2]	-1	“On2”	INSERTION	a == [4, 5, 3, 1, 2]
165	InsertionSortOn2NullArrayWithInvalidLength	NULL	5	“On2”	INSERTION	retorno de sort == 1
166	InsertionSortOn2NullArrayWithInvalidLengthArrayCheck	NULL	5	“On2”	INSERTION	a == NULL
167	InsertionSortOn2FloatArrayCheck	[8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]	18	“On2”	INSERTION	a == [8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]

SELECTION SORT — TESTES

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
168	SelectionSortOn2ArrayCheck	[4, 5, 3, 1, 2]	5	“On2"	SELECTION	*a* == [1, 2, 3, 4, 5]
169	SelectionSortOnArrayCheck	[4, 5, 3, 1, 2]	5	“On”	SELECTION	*a* == [4, 5, 3, 1, 2]
170	SelectionSortOnlognArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	SELECTION	*a* == [4, 5, 3, 1, 2]
171	SelectionSortOn2FuncReturn	[4, 5, 3, 1, 2]	5	“On2"	SELECTION	retorno de sort == 0
172	SelectionSortOnFuncReturn	[4, 5, 3, 1, 2]	5	“On”	SELECTION	retorno de sort == 1
173	SelectionSortOnlognFuncReturn	[4, 5, 3, 1, 2]	5	“Onlogn”	SELECTION	retorno de sort == 1
174	SelectionSortOn2Lowercased	[4, 5, 3, 1, 2]	5	“on2”	SELECTION	retorno de sort == 1
175	SelectionSortOn2LowercasedArrayCheck	[4, 5, 3, 1, 2]	5	“on2”	SELECTION	*a* == [4, 5, 3, 1, 2]
176	SelectionSortOn2Uppercased	[4, 5, 3, 1, 2]	5	“ON2”	SELECTION	retorno de sort == 1
177	SelectionSortOn2UppercasedArrayCheck	[4, 5, 3, 1, 2]	5	“ON2”	SELECTION	*a* == [4, 5, 3, 1, 2]
178	SelectionSortOn2Toggled	[4, 5, 3, 1, 2]	5	“oN2”	SELECTION	retorno de sort == 1
179	SelectionSortOn2ToggledArrayCheck	[4, 5, 3, 1, 2]	5	“oN2”	SELECTION	*a* == [4, 5, 3, 1, 2]
180	SelectionSortOn2WithSpace	[4, 5, 3, 1, 2]	5	“On2 “	SELECTION	retorno de sort == 1
181	SelectionSortOn2WithSpaceArrayCheck	[4, 5, 3, 1, 2]	5	“On2 “	SELECTION	*a* == [4, 5, 3, 1, 2]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
182	SelectionSortOn2Empty	[4, 5, 3, 1, 2]	5	“"	SELECTION	retorno de sort == 1
183	SelectionSortOn2EmptyArrayCheck	[4, 5, 3, 1, 2]	5	“"	SELECTION	*a* == [4, 5, 3, 1, 2]
184	SelectionSortOn2LowerBound	[8, 2]	2	“On2”	SELECTION	retorno de sort == 0
185	SelectionSortOn2LowerBoundArrayCheck	[8, 2]	2	“On2”	SELECTION	*a* == [2, 8]
186	SelectionSortOn2UpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“On2”	SELECTION	retorno de sort == 0
187	SelectionSortOn2UpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“On2”	SELECTION	a == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
188	SelectionSortOn2InvalidLowerBound	[8]	1	“On2”	SELECTION	retorno de sort == 1
189	SelectionSortOn2InvalidUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“On2”	SELECTION	retorno de sort == 1
190	SelectionSortOn2InvalidUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“On2”	SELECTION	Mesmo vetor de entrada, sem modificação

		Casos de Teste				
Número do Teste	**Nome do Teste**	**Entrada**				**Saída Esperada**
		a	***length***	***type***	***algorithm***	
191	SelectionSortOn2LengthZero	[ ]	0	“On2”	SELECTION	retorno de sort == 1
192	SelectionSortOn2NullArray	NULL	0	“On2”	SELECTION	retorno de sort == 1
193	SelectionSortOn2MuchBiggerUpperBound	Vetor com 100 elementos fora de ordem	100	“On2”	SELECTION	retorno de sort == 1
194	SelectionSortOn2MuchBiggerUpperBoundArrayCheck	Vetor com 100 elementos fora de ordem	100	“On2”	100	Mesmo vetor de entrada, sem modificação
195	SelectionSortInvalidAlgorithm	[4, 5, 3, 1, 2]	5	“On2”	100	retorno de sort == 1
196	SelectionSortInvalidAlgorithmArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	100	*a* == [4, 5, 3, 1, 2]
197	SelectionSortInvalidAlgorithm2	[4, 5, 3, 1, 2]	5	“On2”	-1	retorno de sort == 1
198	SelectionSortInvalidAlgorithm2ArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	-1	*a* == [4, 5, 3, 1, 2]
199	SelectionSortOn2UpperBoundContentsArrayCheck	[2147483447, 2044483647, 56237, 21, 0, 2147483647, 13131313, 1947483647, 13, 21474647]	10	“On2”	SELECTION	a == [0, 13, 21, 56237, 13131313, 21474647, 1947483647, 2044483647, 2147483447, 2147483647]
200	SelectionSortOn2BiggerThanUpperBoundContentsArrayCheck	[2147483648, 2147483646, 2147483647]	3	“On2”	SELECTION	a == [-2147483648, 2147483646, 2147483647]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
201	SelectionSortOn2NegAndPosValuesArrayCheck	[5, -2, 3, 0, -1, 2, 1, 4, -3]	9	“On2”	SELECTION	a == [-3, -2, -1, 0, 1, 2, 3, 4, 5]
202	SelectionSortOn2NegativeValuesArrayCheck	[-8, -18, -1, -5, -2, -19, -10, -4, -3, -15, -11, -6, -14, -7, -9, -17, -12, -13, -16]	19	“On2”	SELECTION	a == [-19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1]

SELECTION SORT
TESTES COM ERRO NA EXECUÇÃO

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
203	SelectionSortOn2NullString	[4, 5, 3, 1, 2]	5	NULL	SELECTION	retorno de sort == 1
204	SelectionSortOn2NullStringArrayCheck	[4, 5, 3, 1, 2]	5	NULL	SELECTION	a == [4, 5, 3, 1, 2]
205	SelectionSortOn2NegativeLength	[4, 5, 3, 1, 2]	-1	“On2”	SELECTION	retorno de sort == 1
206	SelectionSortOn2NegativeLengthArrayCheck	[4, 5, 3, 1, 2]	-1	“On2”	SELECTION	a == [4, 5, 3, 1, 2]
207	SelectionSortOn2NullArrayWithInvalidLength	NULL	5	“On2”	SELECTION	retorno de sort == 1
208	SelectionSortOn2NullArrayWithInvalidLengthArrayCheck	NULL	5	“On2”	SELECTION	a == NULL
209	SelectionSortOn2FloatArrayCheck	[8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]	18	“On2”	SELECTION	a == [8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]

HEAP SORT — TESTES

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
210	HeapSortOnlognArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	HEAP	*a* == [1, 2, 3, 4, 5]
211	HeapSortOnArrayCheck	[4, 5, 3, 1, 2]	5	“On”	HEAP	*a* == [4, 5, 3, 1, 2]
212	HeapSortOn2ArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	HEAP	*a* == [4, 5, 3, 1, 2]
213	HeapSortOnlognFuncReturn	[4, 5, 3, 1, 2]	5	“Onlogn”	HEAP	retorno de sort == 0
214	HeapSortOnFuncReturn	[4, 5, 3, 1, 2]	5	“On”	HEAP	retorno de sort == 1
215	HeapSortOn2FuncReturn	[4, 5, 3, 1, 2]	5	“On2”	HEAP	retorno de sort == 1
216	HeapSortOnlognLowercased	[4, 5, 3, 1, 2]	5	“onlogn”	HEAP	retorno de sort == 1
217	HeapSortOnlognLowercasedArrayCheck	[4, 5, 3, 1, 2]	5	“onlogn”	HEAP	*a* == [4, 5, 3, 1, 2]
218	HeapSortOnlognUppercased	[4, 5, 3, 1, 2]	5	“ONLOGN”	HEAP	retorno de sort == 1
219	HeapSortOnlognUppercasedArrayCheck	[4, 5, 3, 1, 2]	5	“ONLOGN”	HEAP	*a* == [4, 5, 3, 1, 2]
220	HeapSortOnlognToggled	[4, 5, 3, 1, 2]	5	“oNlOgN”	HEAP	retorno de sort == 1
221	HeapSortOnlognToggledArrayCheck	[4, 5, 3, 1, 2]	5	“oNlOgN”	HEAP	*a* == [4, 5, 3, 1, 2]
222	HeapSortOnlognWithSpace	[4, 5, 3, 1, 2]	5	“Onlogn “	HEAP	retorno de sort == 1
223	HeapSortOnlognWithSpaceArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn “	HEAP	*a* == [4, 5, 3, 1, 2]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
224	HeapSortOnlognEmpty	[4, 5, 3, 1, 2]	5	“"	HEAP	retorno de sort == 1
225	HeapSortOnlognEmptyArrayCheck	[4, 5, 3, 1, 2]	5	“"	HEAP	*a* == [4, 5, 3, 1, 2]
226	HeapSortOnlognLowerBound	[8, 2]	2	“Onlogn”	HEAP	retorno de sort == 0
227	HeapSortOnlognLowerBoundArrayCheck	[8, 2]	2	“Onlogn”	HEAP	*a* == [2, 8]
228	HeapSortOnlognUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“Onlogn”	HEAP	retorno de sort == 0
229	HeapSortOnlognUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“Onlogn”	HEAP	a == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
230	HeapSortOnlognInvalidLowerBound	[8]	1	“Onlogn”	HEAP	retorno de sort == 1
231	HeapSortOnlognInvalidUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“Onlogn”	HEAP	retorno de sort == 1
232	HeapSortOnlognInvalidUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“Onlogn”	HEAP	Mesmo vetor de entrada, sem modificação

		Casos de Teste				
Número do Teste	**Nome do Teste**	**Entrada**				**Saída Esperada**
		a	***length***	***type***	***algorithm***	
233	HeapSortOnlognMuchBiggerUpperBound	Vetor com 100 elementos fora de ordem	100	“Onlogn”	HEAP	retorno de sort == 1
234	HeapSortOnlognMuchBiggerUpperBoundArrayCheck	Vetor com 100 elementos fora de ordem	100	“Onlogn”	100	Mesmo vetor de entrada, sem modificação
235	HeapSortInvalidAlgorithm	[4, 5, 3, 1, 2]	5	“Onlogn”	100	retorno de sort == 1
236	HeapSortInvalidAlgorithmArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	100	*a* == [4, 5, 3, 1, 2]
237	HeapSortInvalidAlgorithm2	[4, 5, 3, 1, 2]	5	“Onlogn”	-1	retorno de sort == 1
238	HeapSortInvalidAlgorithm2ArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	-1	*a* == [4, 5, 3, 1, 2]
239	HeapSortOnlognUpperBoundContentsArrayCheck	[2147483447, 2044483647, 56237, 21, 0, 2147483647, 13131313, 1947483647, 13, 21474647]	10	“Onlogn”	HEAP	a == [0, 13, 21, 56237, 13131313, 21474647, 1947483647, 2044483647, 2147483447, 2147483647]
240	HeapSortOnlognBiggerThanUpperBoundContentsArrayCheck	[2147483648, 2147483646, 2147483647]	3	“Onlogn”	HEAP	a == [-2147483648, 2147483646, 2147483647]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
241	HeapSortOnlognNegAndPosValuesArrayCheck	[5, -2, 3, 0, -1, 2, 1, 4, -3]	9	“Onlogn”	HEAP	a == [-3, -2, -1, 0, 1, 2, 3, 4, 5]
242	HeapSortOnlognNegativeValuesArrayCheck	[-8, -18, -1, -5, -2, -19, -10, -4, -3, -15, -11, -6, -14, -7, -9, -17, -12, -13, -16]	19	“Onlogn”	HEAP	a == [-19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1]

HEAP SORT
TESTES COM ERRO NA EXECUÇÃO

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
243	HeapSortOnlognNullString	[4, 5, 3, 1, 2]	5	NULL	HEAP	retorno de sort == 1
244	HeapSortOnlognNullStringArrayCheck	[4, 5, 3, 1, 2]	5	NULL	HEAP	a == [4, 5, 3, 1, 2]
245	HeapSortOnlognNegativeLength	[4, 5, 3, 1, 2]	-1	“Onlogn”	HEAP	retorno de sort == 1
246	HeapSortOnlognNegativeLengthArrayCheck	[4, 5, 3, 1, 2]	-1	“Onlogn”	HEAP	a == [4, 5, 3, 1, 2]
247	HeapSortOnlognNullArrayWithInvalidLength	NULL	5	“Onlogn”	HEAP	retorno de sort == 1
248	HeapSortOnlognNullArrayWithInvalidLengthArrayCheck	NULL	5	“Onlogn”	HEAP	a == NULL
249	HeapSortOnlognLengthZero	[ ]	0	“Onlogn”	HEAP	retorno de sort == 1
250	HeapSortOnlognNullArray	NULL	0	“Onlogn”	HEAP	retorno de sort == 1
251	HeapSortOnlognFloatArrayCheck	[8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]	18	“Onlogn”	HEAP	a == [8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]

MERGE SORT — TESTES

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
252	MergeSortOnlognArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	MERGE	*a* == [1, 2, 3, 4, 5]
253	MergeSortOnArrayCheck	[4, 5, 3, 1, 2]	5	“On”	MERGE	*a* == [4, 5, 3, 1, 2]
254	MergeSortOn2ArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	MERGE	*a* == [4, 5, 3, 1, 2]
255	MergeSortOnlognFuncReturn	[4, 5, 3, 1, 2]	5	“Onlogn”	MERGE	retorno de sort == 0
256	MergeSortOnFuncReturn	[4, 5, 3, 1, 2]	5	“On”	MERGE	retorno de sort == 1
257	MergeSortOn2FuncReturn	[4, 5, 3, 1, 2]	5	“On2”	MERGE	retorno de sort == 1
258	MergeSortOnlognLowercased	[4, 5, 3, 1, 2]	5	“onlogn”	MERGE	retorno de sort == 1
259	MergeSortOnlognLowercasedArrayCheck	[4, 5, 3, 1, 2]	5	“onlogn”	MERGE	*a* == [4, 5, 3, 1, 2]
260	MergeSortOnlognUppercased	[4, 5, 3, 1, 2]	5	“ONLOGN”	MERGE	retorno de sort == 1
261	MergeSortOnlognUppercasedArrayCheck	[4, 5, 3, 1, 2]	5	“ONLOGN”	MERGE	*a* == [4, 5, 3, 1, 2]
262	MergeSortOnlognToggled	[4, 5, 3, 1, 2]	5	“oNlOgN”	MERGE	retorno de sort == 1
263	MergeSortOnlognToggledArrayCheck	[4, 5, 3, 1, 2]	5	“oNlOgN”	MERGE	*a* == [4, 5, 3, 1, 2]
264	MergeSortOnlognWithSpace	[4, 5, 3, 1, 2]	5	“Onlogn “	MERGE	retorno de sort == 1
265	MergeSortOnlognWithSpaceArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn “	MERGE	*a* == [4, 5, 3, 1, 2]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
266	MergeSortOnlognEmpty	[4, 5, 3, 1, 2]	5	“"	MERGE	retorno de sort == 1
267	MergeSortOnlognEmptyArrayCheck	[4, 5, 3, 1, 2]	5	“"	MERGE	*a* == [4, 5, 3, 1, 2]
268	MergeSortOnlognLowerBound	[8, 2]	2	“Onlogn”	MERGE	retorno de sort == 0
269	MergeSortOnlognLowerBoundArrayCheck	[8, 2]	2	“Onlogn”	MERGE	*a* == [2, 8]
270	MergeSortOnlognUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“Onlogn”	MERGE	retorno de sort == 0
271	MergeSortOnlognUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“Onlogn”	MERGE	a == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
272	MergeSortOnlognInvalidLowerBound	[8]	1	“Onlogn”	MERGE	retorno de sort == 1
273	MergeSortOnlognInvalidUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“Onlogn”	MERGE	retorno de sort == 1
274	MergeSortOnlognInvalidUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“Onlogn”	MERGE	Mesmo vetor de entrada, sem modificação

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
275	MergeSortOnlognLengthZero	[ ]	0	“Onlogn”	MERGE	retorno de sort == 1
276	MergeSortOnlognNullArray	NULL	0	“Onlogn”	MERGE	retorno de sort == 1
277	MergeSortOnlognMuchBiggerUpperBound	Vetor com 100 elementos fora de ordem	100	“Onlogn”	MERGE	retorno de sort == 1
278	MergeSortOnlognMuchBiggerUpperBoundArrayCheck	Vetor com 100 elementos fora de ordem	100	“Onlogn”	100	Mesmo vetor de entrada, sem modificação
279	MergeSortInvalidAlgorithm	[4, 5, 3, 1, 2]	5	“Onlogn”	100	retorno de sort == 1
280	MergeSortInvalidAlgorithmArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	100	*a* == [4, 5, 3, 1, 2]
281	MergeSortInvalidAlgorithm2	[4, 5, 3, 1, 2]	5	“Onlogn”	-1	retorno de sort == 1
282	MergeSortInvalidAlgorithm2ArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	-1	*a* == [4, 5, 3, 1, 2]
283	MergeSortOnlognUpperBoundContentsArrayCheck	[2147483447, 2044483647, 56237, 21, 0, 2147483647, 13131313, 1947483647, 13, 21474647]	10	“Onlogn”	MERGE	a == [0, 13, 21, 56237, 13131313, 21474647, 1947483647, 2044483647, 2147483447, 2147483647]
284	MergeSortOnlognBiggerThanUpperBoundContentsArrayCheck	[2147483648, 2147483646, 2147483647]	3	“Onlogn”	MERGE	a == [-2147483648, 2147483646, 2147483647]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	***length***	***type***	***algorithm***	
285	MergeSortOnlognNegAndPosValuesArrayCheck	[5, -2, 3, 0, -1, 2, 1, 4, -3]	9	“Onlogn”	MERGE	a == [-3, -2, -1, 0, 1, 2, 3, 4, 5]
286	MergeSortOnlognNegativeValuesArrayCheck	[-8, -18, -1, -5, -2, -19, -10, -4, -3, -15, -11, -6, -14, -7, -9, -17, -12, -13, -16]	19	“Onlogn”	MERGE	a == [-19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1]

MERGE SORT
TESTES COM ERRO NA EXECUÇÃO

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
287	MergeSortOnlognNullString	[4, 5, 3, 1, 2]	5	NULL	MERGE	retorno de sort == 1
288	MergeSortOnlognNullStringArrayCheck	[4, 5, 3, 1, 2]	5	NULL	MERGE	a == [4, 5, 3, 1, 2]
289	MergeSortOnlognNegativeLength	[4, 5, 3, 1, 2]	-1	“Onlogn”	MERGE	retorno de sort == 1
290	MergeSortOnlognNegativeLengthArrayCheck	[4, 5, 3, 1, 2]	-1	“Onlogn”	MERGE	a == [4, 5, 3, 1, 2]
291	MergeSortOnlognNullArrayWithInvalidLength	NULL	5	“Onlogn”	MERGE	retorno de sort == 1
292	MergeSortOnlognNullArrayWithInvalidLengthArrayCheck	NULL	5	“Onlogn”	MERGE	a == NULL
293	MergeSortOnlognFloatArrayCheck	[8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]	18	“Onlogn”	MERGE	a == [8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]

QUICK SORT — TESTES

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
294	QuickSortOnlognArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	QUICK	*a* == [1, 2, 3, 4, 5]
295	QuickSortOnArrayCheck	[4, 5, 3, 1, 2]	5	“On”	QUICK	*a* == [4, 5, 3, 1, 2]
296	QuickSortOn2ArrayCheck	[4, 5, 3, 1, 2]	5	“On2”	QUICK	*a* == [4, 5, 3, 1, 2]
297	QuickSortOnlognFuncReturn	[4, 5, 3, 1, 2]	5	“Onlogn”	QUICK	retorno de sort == 0
298	QuickSortOnFuncReturn	[4, 5, 3, 1, 2]	5	“On”	QUICK	retorno de sort == 1
299	QuickSortOn2FuncReturn	[4, 5, 3, 1, 2]	5	“On2”	QUICK	retorno de sort == 1
300	QuickSortOnlognLowercased	[4, 5, 3, 1, 2]	5	“onlogn”	QUICK	retorno de sort == 1
301	QuickSortOnlognLowercasedArrayCheck	[4, 5, 3, 1, 2]	5	“onlogn”	QUICK	*a* == [4, 5, 3, 1, 2]
302	QuickSortOnlognUppercased	[4, 5, 3, 1, 2]	5	“ONLOGN”	QUICK	retorno de sort == 1
303	QuickSortOnlognUppercasedArrayCheck	[4, 5, 3, 1, 2]	5	“ONLOGN”	QUICK	*a* == [4, 5, 3, 1, 2]
304	QuickSortOnlognToggled	[4, 5, 3, 1, 2]	5	“oNlOgN”	QUICK	retorno de sort == 1
305	QuickSortOnlognToggledArrayCheck	[4, 5, 3, 1, 2]	5	“oNlOgN”	QUICK	*a* == [4, 5, 3, 1, 2]
306	QuickSortOnlognWithSpace	[4, 5, 3, 1, 2]	5	“Onlogn “	QUICK	retorno de sort == 1
307	QuickSortOnlognWithSpaceArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn “	QUICK	*a* == [4, 5, 3, 1, 2]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
308	QuickSortOnlognEmpty	[4, 5, 3, 1, 2]	5	“"	QUICK	retorno de sort == 1
309	QuickSortOnlognEmptyArrayCheck	[4, 5, 3, 1, 2]	5	“"	QUICK	*a* == [4, 5, 3, 1, 2]
310	QuickSortOnlognLowerBound	[8, 2]	2	“Onlogn”	QUICK	retorno de sort == 0
311	QuickSortOnlognLowerBoundArrayCheck	[8, 2]	2	“Onlogn”	QUICK	*a* == [2, 8]
312	QuickSortOnlognUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“Onlogn”	QUICK	retorno de sort == 0
313	QuickSortOnlognUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16]	20	“Onlogn”	QUICK	a == [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
314	QuickSortOnlognInvalidLowerBound	[8]	1	“Onlogn”	QUICK	retorno de sort == 1
315	QuickSortOnlognInvalidUpperBound	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“Onlogn”	QUICK	retorno de sort == 1
316	QuickSortOnlognInvalidUpperBoundArrayCheck	[8, 18, 1, 20, 5, 2, 19, 10, 4, 3, 15, 11, 6, 14, 7, 9, 17, 12, 13, 16, 21]	21	“Onlogn”	QUICK	Mesmo vetor de entrada, sem modificação

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
317	QuickSortOnlognLengthZero	[ ]	0	“Onlogn”	QUICK	retorno de sort == 1
318	QuickSortOnlognNullArray	NULL	0	“Onlogn”	QUICK	retorno de sort == 1
319	QuickSortOnlognMuchBiggerUpperBound	Vetor com 100 elementos fora de ordem	100	“Onlogn”	QUICK	retorno de sort == 1
320	QuickSortOnlognMuchBiggerUpperBoundArrayCheck	Vetor com 100 elementos fora de ordem	100	“Onlogn”	100	Mesmo vetor de entrada, sem modificação
321	QuickSortInvalidAlgorithm	[4, 5, 3, 1, 2]	5	“Onlogn”	100	retorno de sort == 1
322	QuickSortInvalidAlgorithmArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	100	*a* == [4, 5, 3, 1, 2]
323	QuickSortInvalidAlgorithm2	[4, 5, 3, 1, 2]	5	“Onlogn”	-1	retorno de sort == 1
324	QuickSortInvalidAlgorithm2ArrayCheck	[4, 5, 3, 1, 2]	5	“Onlogn”	-1	*a* == [4, 5, 3, 1, 2]
325	QuickSortOnlognUpperBoundContentsArrayCheck	[2147483447, 2044483647, 56237, 21, 0, 2147483647, 13131313, 1947483647, 13, 21474647]	10	“Onlogn”	QUICK	a == [0, 13, 21, 56237, 13131313, 21474647, 1947483647, 2044483647, 2147483447, 2147483647]
326	QuickSortOnlognBiggerThanUpperBoundContentsArrayCheck	[2147483648, 2147483646, 2147483647]	3	“Onlogn”	QUICK	a == [-2147483648, 2147483646, 2147483647]

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
327	QuickSortOnlognNegAndPosValuesArrayCheck	[5, -2, 3, 0, -1, 2, 1, 4, -3]	9	“Onlogn”	QUICK	a == [-3, -2, -1, 0, 1, 2, 3, 4, 5]
328	QuickSortOnlognNegativeValuesArrayCheck	[-8, -18, -1, -5, -2, -19, -10, -4, -3, -15, -11, -6, -14, -7, -9, -17, -12, -13, -16]	19	“Onlogn”	QUICK	a == [-19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1]

QUICK SORT
TESTES COM ERRO NA EXECUÇÃO

Número do Teste	Nome do Teste	Casos de Teste				
		Entrada				Saída Esperada
		a	*length*	*type*	*algorithm*	
329	QuickSortOnlognNullString	[4, 5, 3, 1, 2]	5	NULL	QUICK	retorno de sort == 1
330	QuickSortOnlognNullStringArrayCheck	[4, 5, 3, 1, 2]	5	NULL	QUICK	a == [4, 5, 3, 1, 2]
331	QuickSortOnlognNegativeLength	[4, 5, 3, 1, 2]	-1	“Onlogn”	QUICK	retorno de sort == 1
332	QuickSortOnlognNegativeLengthArrayCheck	[4, 5, 3, 1, 2]	-1	“Onlogn”	QUICK	a == [4, 5, 3, 1, 2]
333	QuickSortOnlognNullArrayWithInvalidLength	NULL	5	“Onlogn”	QUICK	retorno de sort == 1
334	QuickSortOnlognNullArrayWithInvalidLengthArrayCheck	NULL	5	“Onlogn”	QUICK	a == NULL
335	QuickSortOnlognFloatArrayCheck	[8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]	18	“Onlogn”	QUICK	a == [8.0, 1.0, 20.0, 5.0, 2.0, 10.0, 4.0, 3.0, 15.0, 11.0, 6.0, 14.0, 7.0, 9.0, 17.0, 12.0, 13.0, 16.0]

COBERTURA DE CÓDIGO (*Code Coverage*)

Nome do Arquivo	Linhas executadas (%)	*Branches* executados (%)	Linhas não executadas (se aplicável)
counting_sort.c	100%	100%	N/A
radix_sort.c	92,73%	100%	Linhas 54, 55, 76 e 77
heap_sort.c	93,35%	100%	Linha 48, 49
insertion_sort.c	100%	100%	N/A
merge_sort.c	100%	100%	N/A
quick_sort.c	100%	100%	N/A
selection_sort.c	100%	100%	N/A
bubble_sort.c	100%	100%	N/A
sort.c	100%	100%	N/A