CENTRO UNIVERSITÁRIO CATÓLICO DE VITÓRIA

Mateus Garcia Lopes
Vitor Knupp Costa

Análise e Projeto de Algoritmos

Vitória - ES
2017

Análise de teste dos algoritmos de ordenação

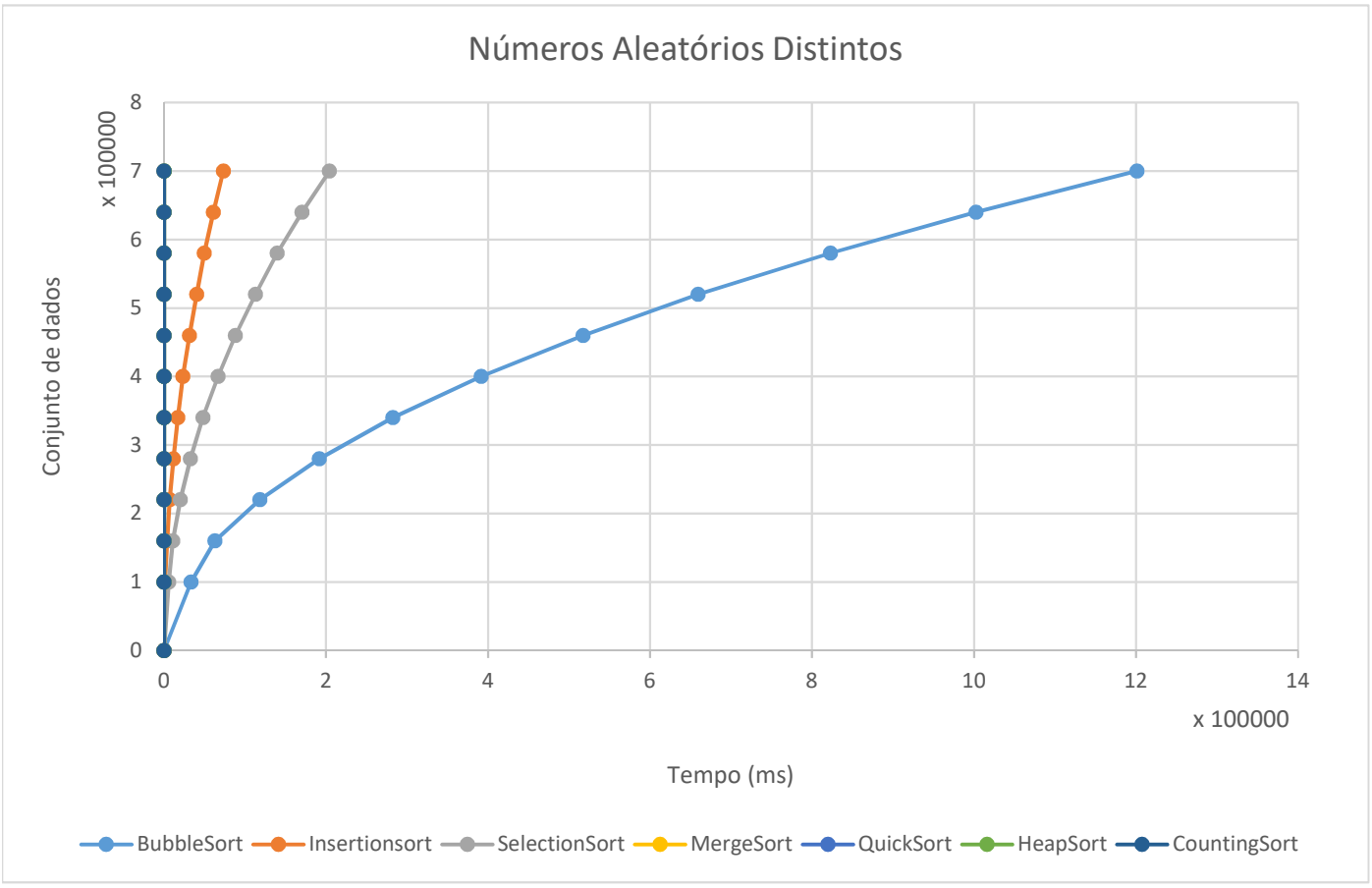
Foi desenvolvido um sistema para ordenação de números. Esse sistema foi implementado na linguagem de programação Java com diversos algoritmos de ordenação conforme apresentados em sala de aula.
Esse documento registra os testes efetuados e como é o comportamento dos algoritmos de ordenação em diferentes cenários.

Será apresentando abaixo a extração das médias do tempo de execução de cada algoritmo implementado com o respectivo conjunto de dados. Em outras palavras, será exibido em tabelas e gráficos qual foi a média de tempo de execução de cada algoritmo para ordenar os conjuntos de dados a partir de 100.000 até 700.000 números inteiros.

Os algoritmos implementados foram: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort, Heap Sort e Counting Sort. Os conjuntos de dados foram: 100.000, 160.000, 220.000, 280.000, 340.000, 400.000, 460.000, 520.000, 640.000 e 700.000 números inteiros, sendo esses conjuntos de números aleatório e repetido.

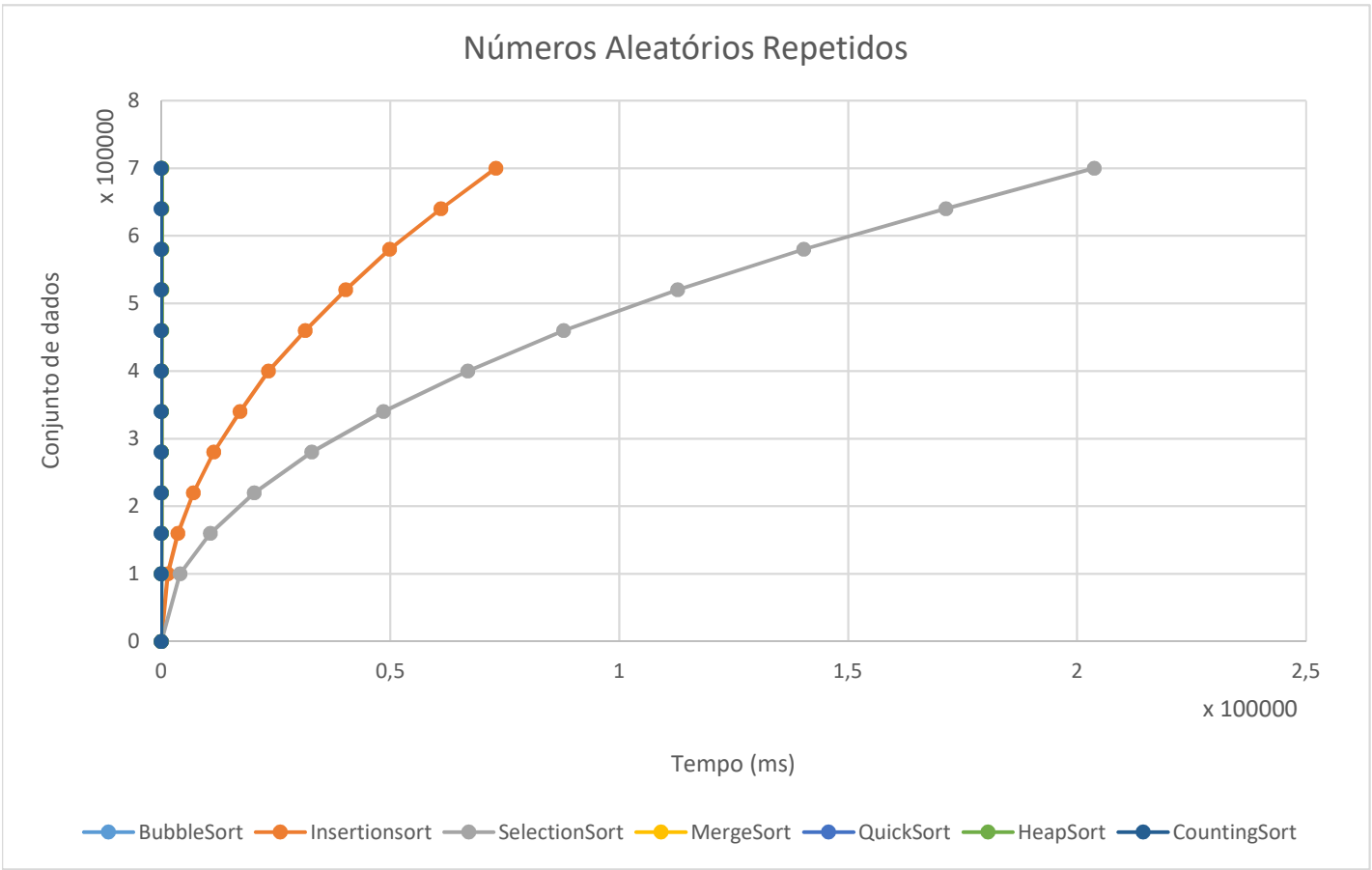
Aleatório Distintos

Dados	BubbleSort	Insertionsort	SelectionSort	MergeSort	QuickSort	HeapSort	CountingSort
0	0	0	0	0	0	0	0
100.000	33201,67	1291,67	5792,33	20,67	18.33	21	5,33
160.000	62715	3656,67	10897	26	26.0	26	0
220.000	118385,33	6985	20010,67	41,67	38.33	47	5,33
280.000	191690,33	11419	32675,67	52,33	45.0	57,33	10,33
340.000	282545,33	17199,67	48145,67	62,33	52.0	67,67	0
400.000	391595,67	23320,33	66663	78	60.33	78	0
460.000	517183,33	31367	87947,67	88,33	66.0	94	0
520.000	659194	40171	112865,33	93,67	78.67	109	10,67
580.000	822865	49656,33	139525	104	95.0	125	15
640.000	1002575	60709	170496,67	119,67	101.33	130	15,33
700.000	1201261	73111,67	204268,33	130,33	109.33	172	15,33



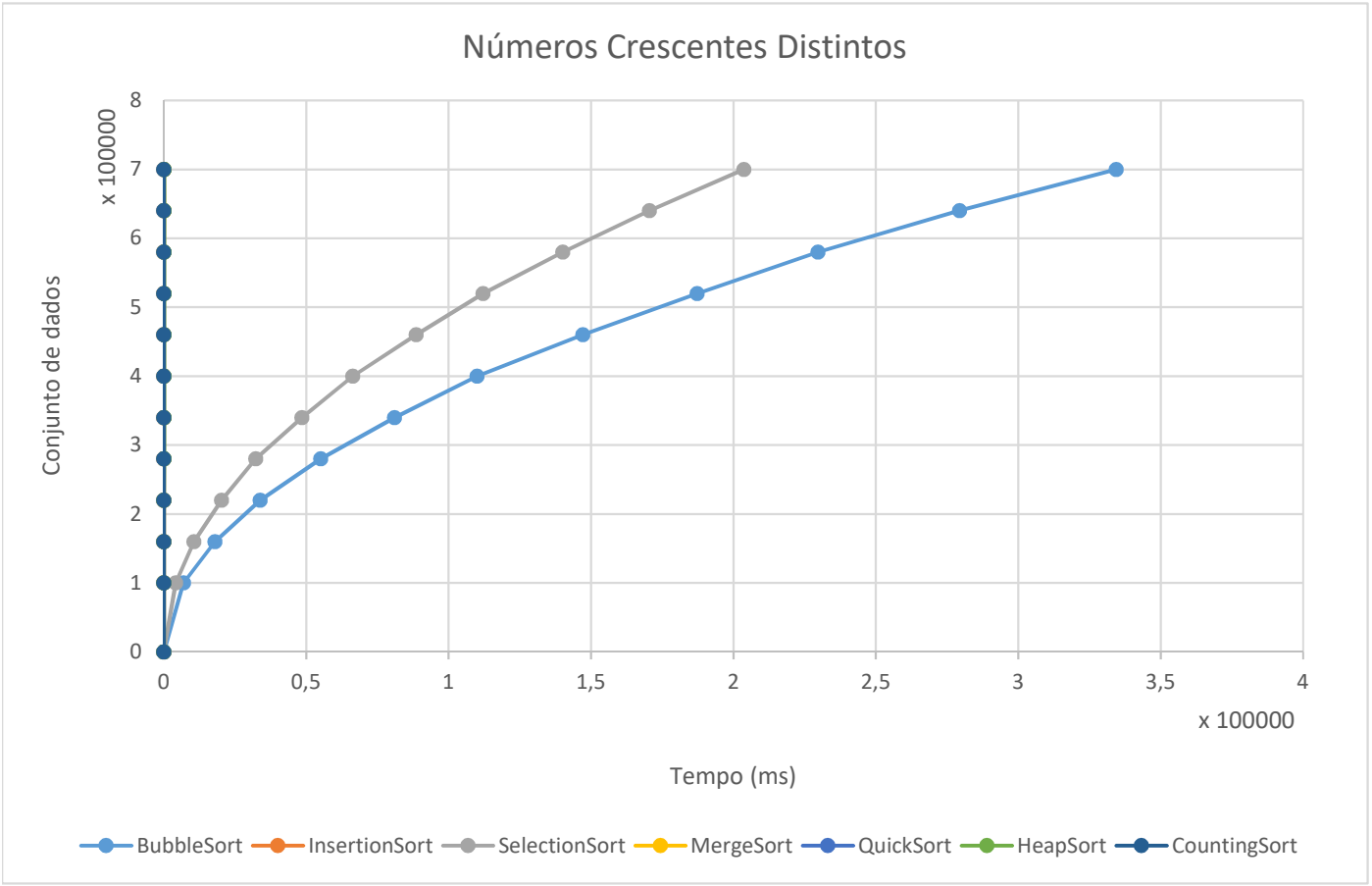
Aleatório Repetidos

Dados	BubbleSort	Insertionsort	SelectionSort	MergeSort	QuickSort	HeapSort	CountingSort
0	0	0	0	0	0	0	0
100.000	24497,67	1406,33	4125,33	15,33	14.67	21,33	0
160.000	62558	3635,67	10742	31	25.33	31,33	0
220.000	118549,33	7006	20325,33	36,67	32.0	47	0
280.000	191932	11454,33	32878,33	51,67	38.33	52,33	5,33
340.000	282719,67	17231,33	48515,67	62	53.33	68,33	0
400.000	391396,67	23410	66970	62,67	59.33	88,67	0
460.000	517556,33	31415,33	87890,33	88,67	69.67	93,67	5
520.000	661733,33	40275,33	112762,33	99	87.33	109	10,33
580.000	821844,33	49890,67	140291,33	109,33	87.0	125	5
640.000	1001722,33	61090	171341	120	94.33	135,33	10,67
700.000	1199776,33	73060,33	203748,67	141	110.67	156,33	15,67



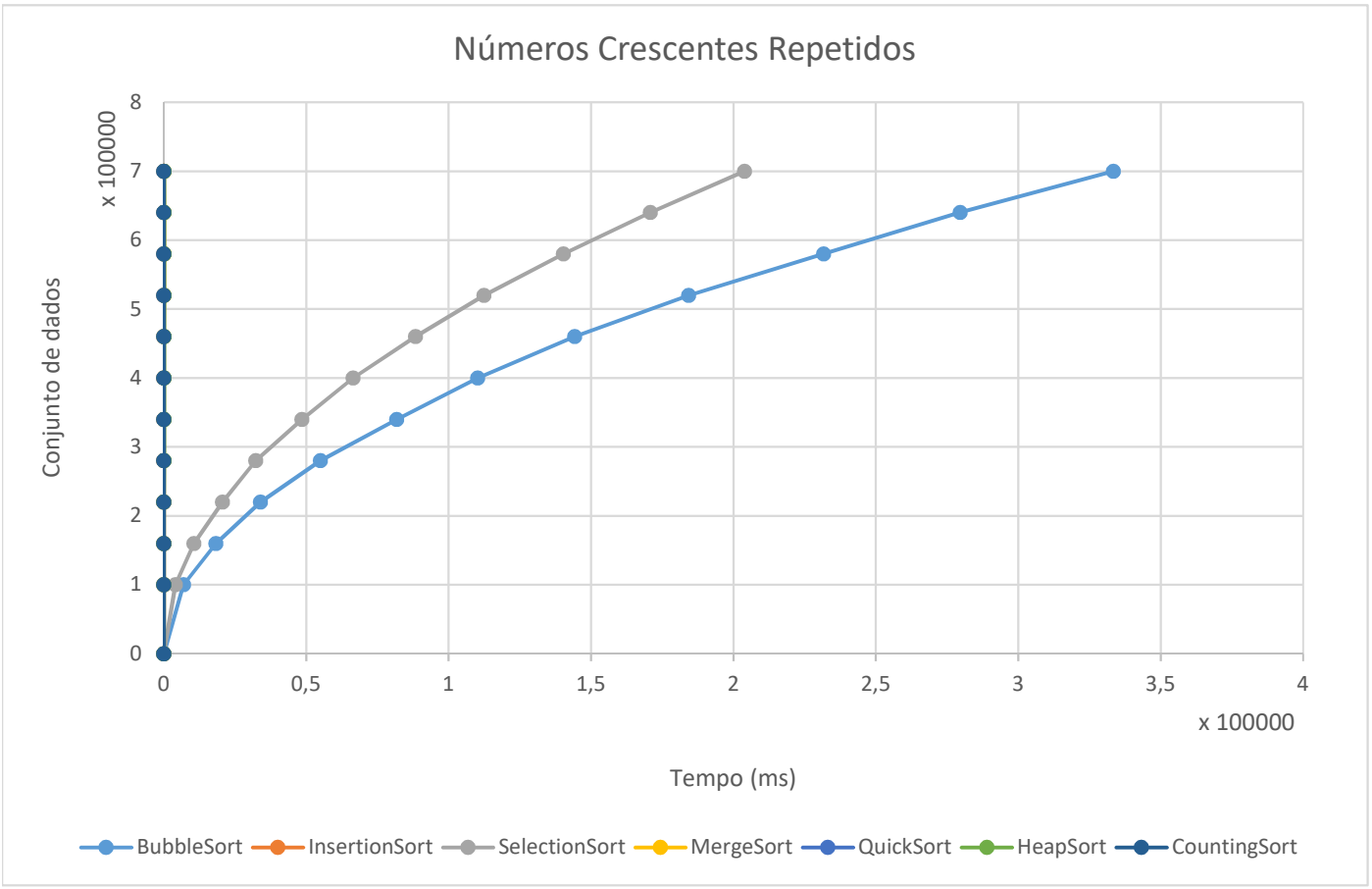
Cresentes Distintos

Dados	BubbleSort	Insertionsort	SelectionSort	MergeSort	QuickSort	HeapSort	CountingSort
0	0	0	0	0	0	0	0
100.000	6954	0	4119,67	5,33	4.33	10,33	0
160.000	17944,67	0	10590	0	7.0	15,33	5,33
220.000	33795,33	0	20236,33	16	16.33	31	5,33
280.000	55115	0	32248	15,33	13.0	41,67	0
340.000	80971,67	0	48428,33	31,33	15.67	41,67	0
400.000	109939,67	0	66315,33	31	19.33	47	0
460.000	147098	0	88567,33	36,33	24.67	57	0
520.000	187221,33	0	112088,67	36,67	27.0	62,33	0
580.000	229716	5,33	140063,33	36,33	28.67	67,67	0
640.000	279374	5,33	170454,67	47	28.33	72,67	10,33
700.000	334427,67	0	203676,67	52	30.67	78	10,33



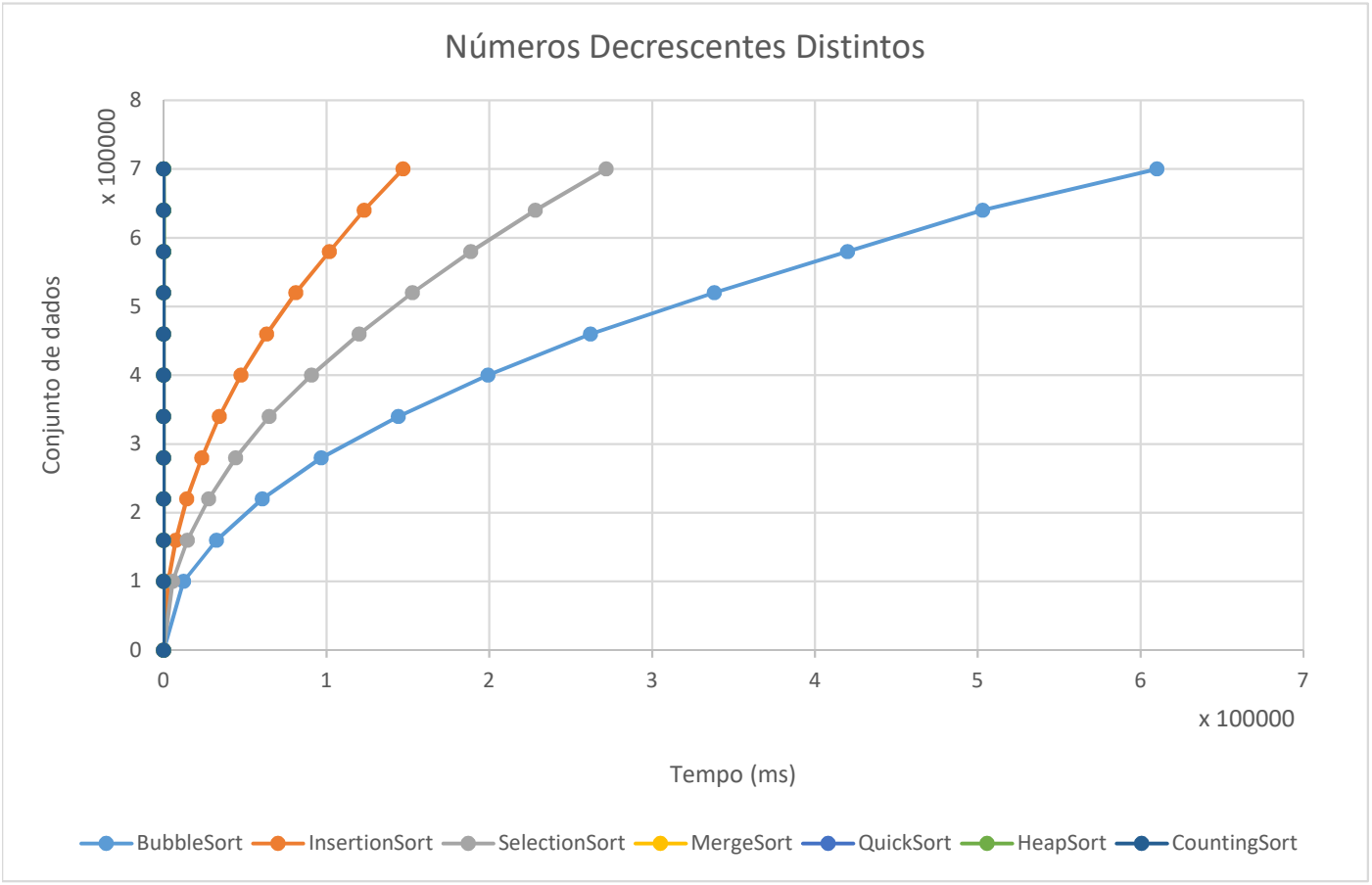
Crescentes Repetidos

Dados	BubbleSort	Insertionsort	SelectionSort	MergeSort	QuickSort	HeapSort	CountingSort
0	0	0	0	0	0	0	0
100.000	6943	5,33	4114,33	0	4.67	10,33	5,33
160.000	18293,67	0	10589,67	10	12.0	20,67	0
220.000	33925,33	0	20554,33	15,67	18.33	25,67	5
280.000	54927	0	32226,33	21	18.0	42	5,33
340.000	81727,33	0	48432,67	20,67	23.33	41,67	5,33
400.000	110230,67	0	66423,33	36,33	25.67	47	5,33
460.000	144234,33	0	88348,67	36,33	32.67	57,67	0
520.000	184253,67	10,33	112455	41,67	32.0	67,33	5,33
580.000	231609	0	140322,67	46,67	41.33	78	0
640.000	279567	0	170804,67	52	41.33	78	5,33
700.000	333419	5,33	203917,67	57,33	39.0	93,67	5



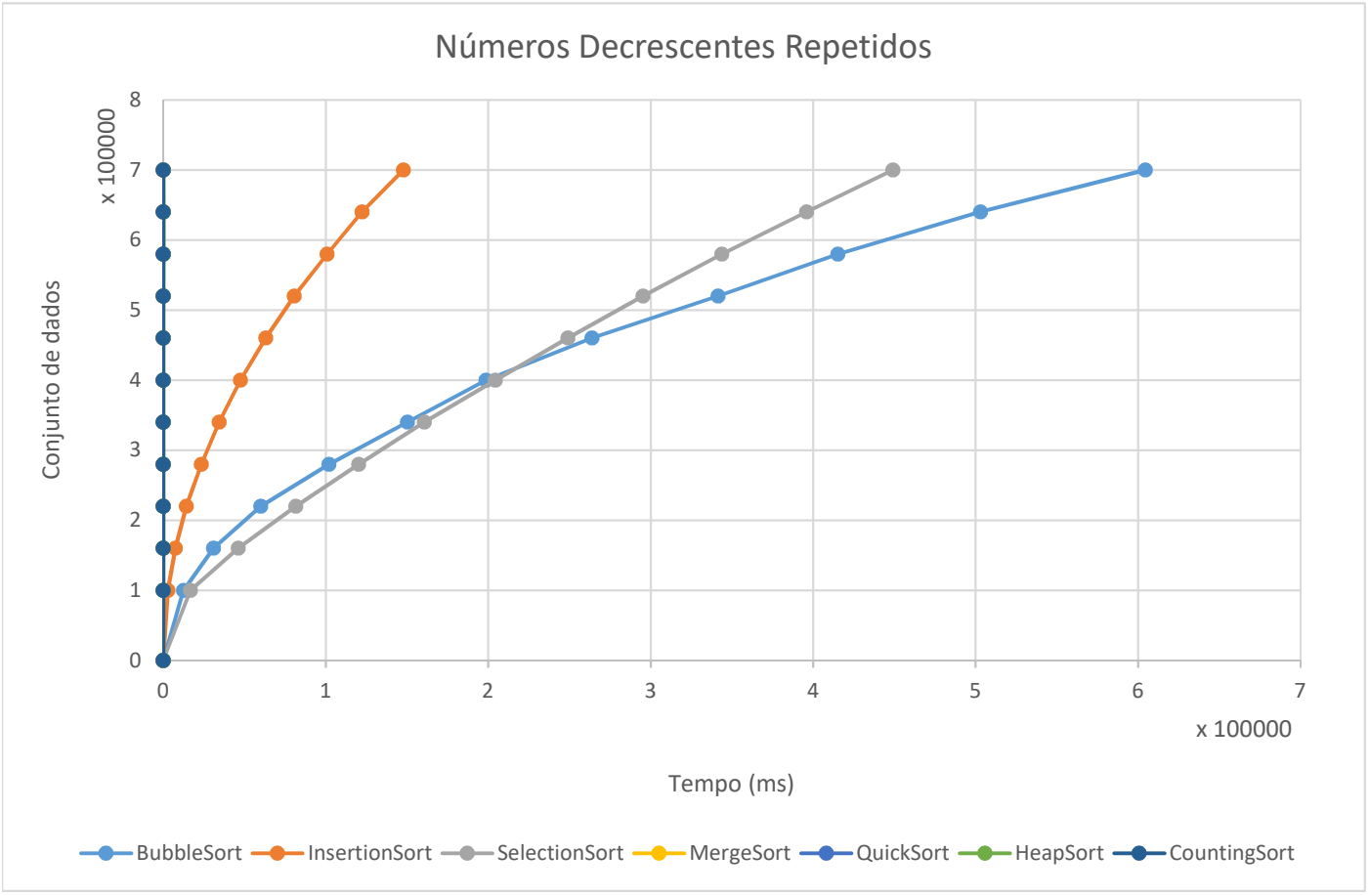
Decrescentes Distintos

Dados	BubbleSort	Insertionsort	SelectionSort	MergeSort	QuickSort	HeapSort	CountingSort
0	0	0	0	0	0	0	0
100.000	12277,33	2844	5594,33	5,33	6.67	10,33	5,33
160.000	32576	7505,67	14574,33	10	7.33	25,67	0
220.000	60585,33	14142	27706	16	12.33	26,33	0
280.000	96863	23476,33	44316,33	21	13.33	31	0
340.000	144229,67	34310,67	64914,67	26	16.33	41,33	0
400.000	199129,33	47526,33	90814,67	15,67	19.0	46,67	5,33
460.000	262138,33	63283	120128,33	36,67	22.33	52	5
520.000	338122	81138,67	152823	36,67	25.33	62	5,33
580.000	419936,67	101760,33	188596,33	36,33	30.33	73	0
640.000	502973,67	123199,67	228362,67	41,67	31.67	78	0
700.000	610071,67	147108,67	271704,67	46,67	33.0	88,67	5,33



Decrescentes Repetidos							

Dados	BubbleSort	Insertionsort	SelectionSort	MergeSort	QuickSort	HeapSort	CountingSort
0	0	0	0	0	0	0	0
100.000	12470,33	2839	16679	5,33	5.33	20,33	0
160.000	31029,67	7501	46230	10,67	13.67	15,33	0
220.000	60023,33	14147,33	81571,33	15,67	15.0	31,67	0
280.000	101923,33	23513	120288,33	21	17.33	36,33	0
340.000	150274	34431,67	160780	26,33	26.33	51,67	0
400.000	198567	47452,33	204434	25,67	32.0	47	0
460.000	263865,33	63198,67	249000,67	31,33	31.0	52,33	10,33
520.000	341493,33	80550	295198	31,33	84.67	57,33	10,33
580.000	415190,33	100838	343810,33	47	73.0	72,67	5,33
640.000	502986,67	122389	395997	41,67	63.67	83,33	5
700.000	604516	147876,33	449098	47	54.0	88,67	5,33



Conclusão

Qual é o melhor algoritmo quando os elementos estão ordenados de forma crescente?
R.: Insertion Sort, complexidade no pior caso $O(n^2)$. Conforme os testes foi ordenado em poucos millisegundos a maior quantidade de elementos, 700.000 números inteiros distintos ou repetidos. Isso ocorre pois o Insertion Sort, no melhor caso que é quando os elementos estão ordenados de forma crescente, passa a ter complexidade O(n) e com isso, o tempo de execução é muito pequeno.

Qual é o melhor algoritmo quando os elementos estão ordenados de forma decrescente?
R.: Counting Sort, complexidade O(n). Esse algoritmo se mostrou o mais eficiente em ordenar o conjunto de dados testados, números inteiros distintos ou repetidos. Sua complexidade é O(n) no melhor e no pior caso e por ser linear o tempo de execução é menor que os demais métodos de ordenação.

Qual é o algoritmo mais estável em relação ao tempo de processamento? Ou seja, qual o que menos varia o tempo de processamento independente da forma como os dados estão organizados no vetor?
R.: O Counting Sort se mostrou bem estável independente da forma como os dados estão organizados e também foi o que executou em menor tempo. Vale lembrar que mesmo usando a lógica diferente do Counting Sort, os algoritmos Heap Sort O(n log n) e o Merge Sort O(n log n) também se mostraram estáveis, porém com o tempo de execução superior ao Counting Sort O(n).