UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Ciência da Computação

Rafael Igor Athaydes Rios

Algoritmos de Ordenação

Vitória
2015

Resumo: Este trabalho tem como finalidade apresentar uma disversidade de Algoritmos de Ordenação e comparar seus tempos de execução a partir de três tipos diferentes de entrada, são elas: ordenada crescente, decrescente e aleatório.
As entradas usadas possuem apenas números inteiros entre 0 e 1.000.000 e tem tamanhos: mil, cinquenta mil, cem mil, quinhentos mil e um milhão.
Foram analisados neste trabalho quatorze algoritmos diferentes: bubblesort, shakesort, insertionsort, shellsort, selectionsort, ranksort, quicksort primeiro, quicksort central, quicksort random, quicksort mediana3, mergesort, heapsort, radixsort e radixbinsort.

Palavras-chave: algortimos, ordenação.

Introdução

Os Algortimos de Ordenação possuem uma grande importância não só na área da computação, mas também no dia a dia. Imagine você ter que consultar seus contatos no telefone sem estarem em ordem alfabética.

Existe uma variedade enorme de algortimos de ordenação conhecidos, cada um com seu tempo de execução. Muitas vezes os algoritmos mais complexos, logo mais difíceis de implementar, são os que possuem os melhores tempos. Mas mesmo assim um algoritmo simples ainda possui utilidade por serem facilmente implementados.

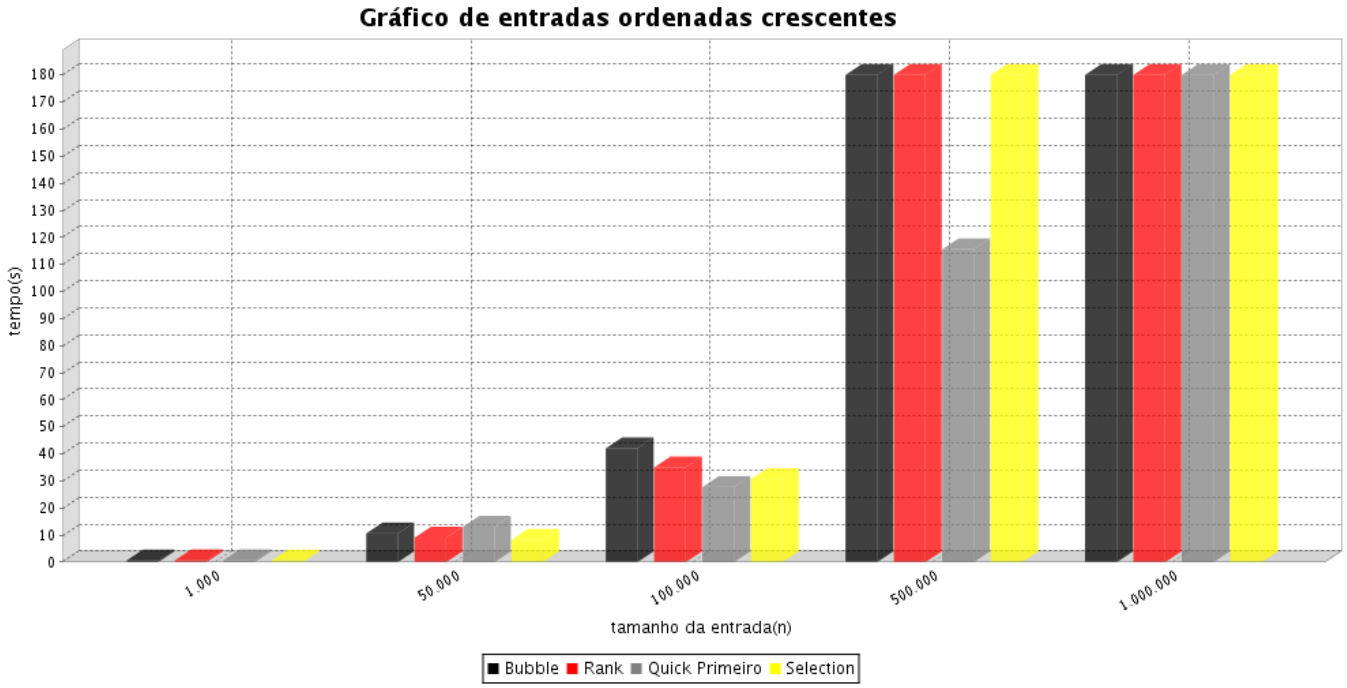
Implementação

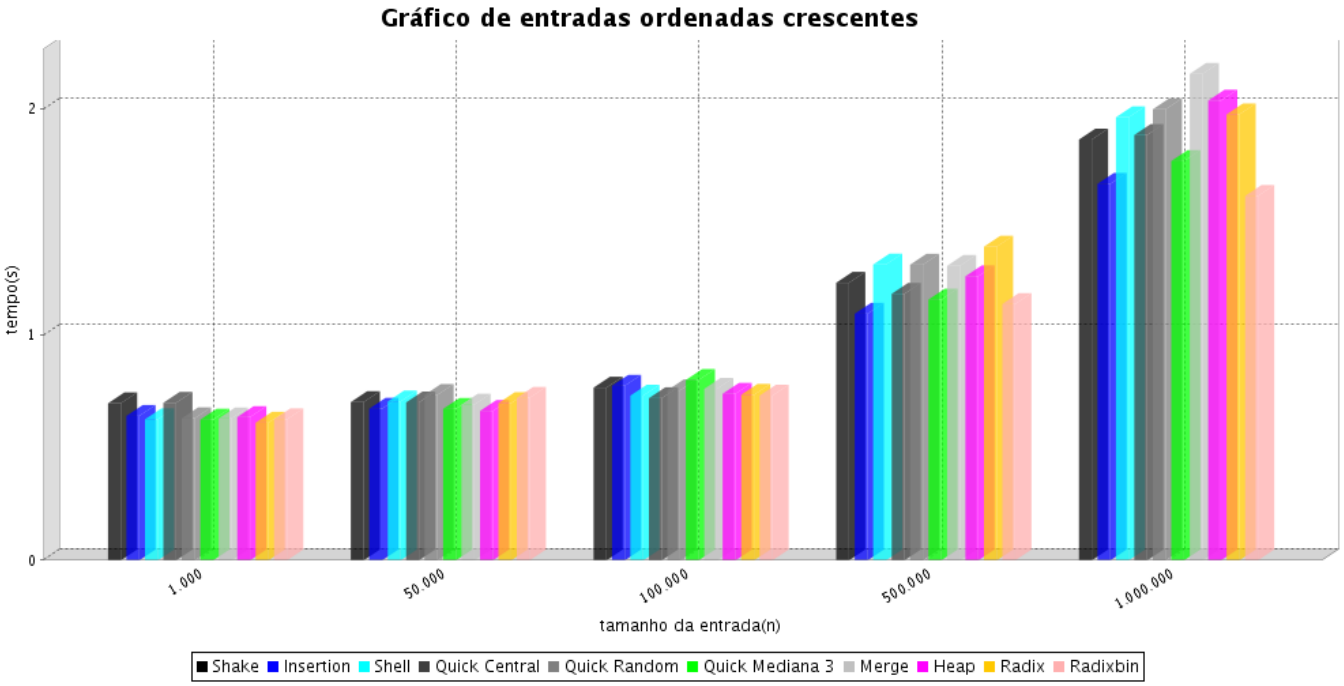
Para cada tipo de entrada existe uma tabela e seu respectivo gráfico. O tempo máximo pra cada algoritmo será de 3 minutos (180 s).

- **Entradas ordenadas crescentes**

Tabela 1: Entradas Ordenadas Crescentes

	Tamanhos				
Algoritmo	**1.000**	**50.000**	**100.000**	**500.000**	**1.000.000**
Bubblesort	0.607	10.67	42.087	>180	>180
Shakesort	0.695	0.700	0.764	1.228	1.862
Insertionsort	0.639	0.671	0.773	1.091	1.666
Shellsort	0.624	0.703	0.729	1.309	1.960
Selectionsort	0.631	8.226	30.679	>180	>180
Ranksort	0.676	9.091	35.020	>180	>180
Quicksort Primeiro	0.662	13.130	27.763	115.584	>180
Quicksort Central	0.697	0.699	0.718	1.179	1.882
Quicksort Random	0.629	0.732	0.752	1.308	1.996
Quick Mediana 3	0.623	0.670	0.798	1.154	1.767
Mergesort	0.628	0.689	0.759	1.303	2.152
Heapsort	0.634	0.661	0.737	1.257	2.034
Radixsort	0.609	0.696	0.733	1.388	1.973
Radixbinsort	0.624	0.719	0.729	1.134	1.612





Analizando os resultados percebe-se que os algoritmos bubblesort, selectionsort, ranksort e quickprimeiro possuem o tempo O(n^2), sendo assim, não possuem boa performance com entradas ordenadas crescentes muito grandes, acima de 500 mil números. Com este tipo de entrada o algortimo quick primeiro cai em seu pior caso, onde o pivô é o menor número da entrada, isso faz com que as partições sejam completamente desbalanceadas.

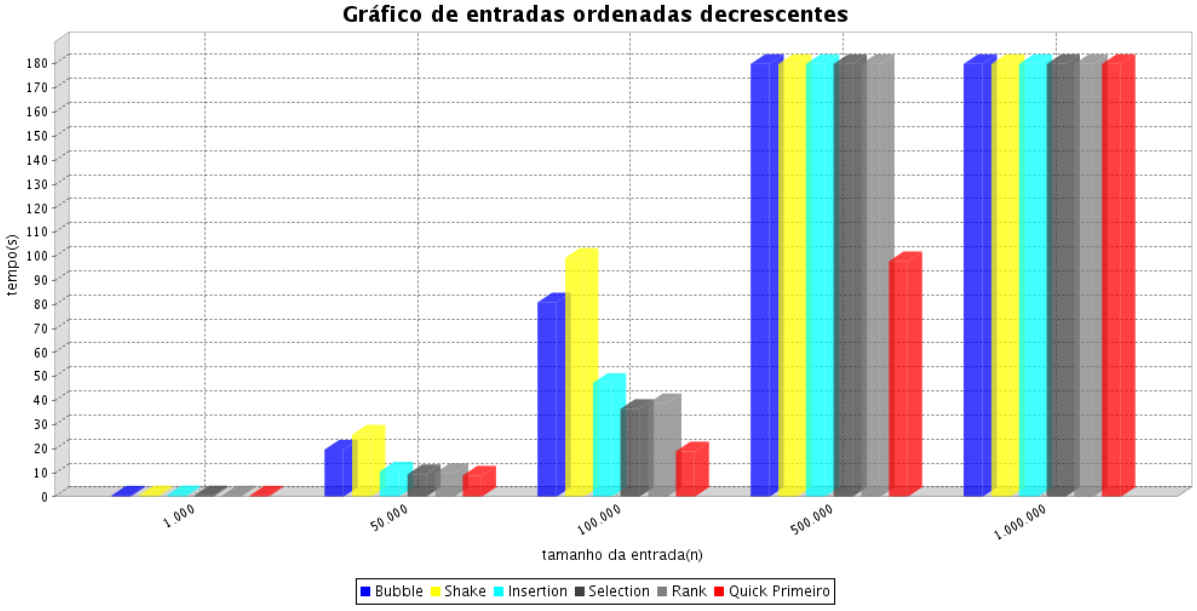
Os algortimos shakesort e insertionsort para este tipo de entrada caem no melhor caso, sendo assim eles possuem tempo O(n).

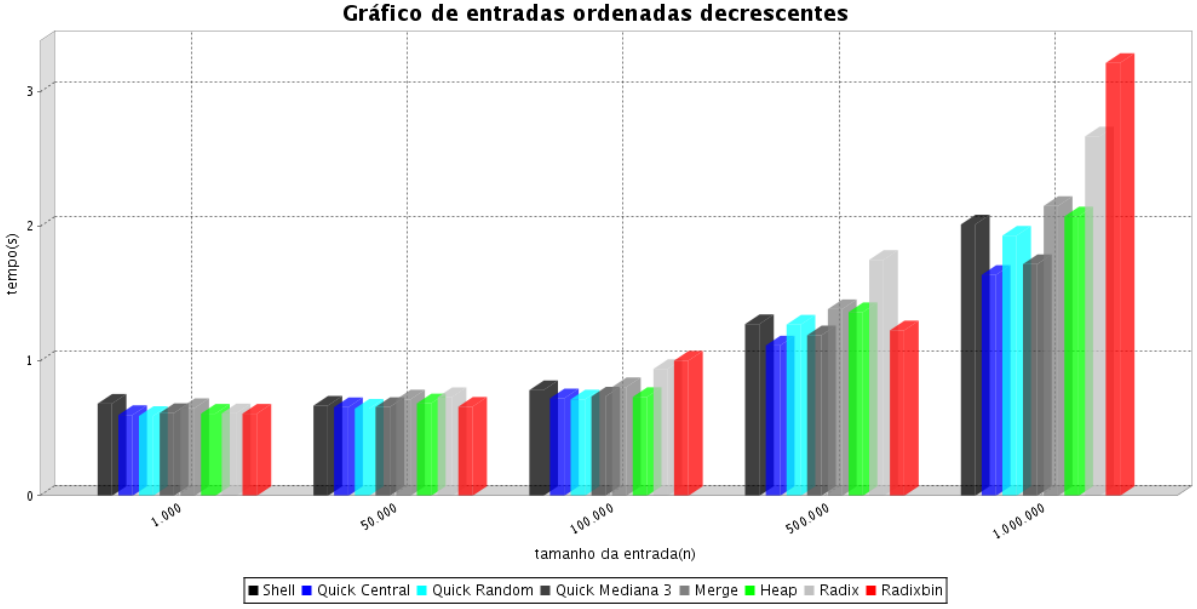
Os demais algoritmos conseguem ordenar as entradas com tempos pequenos, o que se já era esperado.

- **Entradas ordenadas decrescentes**

Tabela 2: Entradas Ordenadas Decrescentes

	Tamanhos				
Algoritmo	**1.000**	**50.000**	**100.000**	**500.000**	**1.000.000**
Bubblesort	0.578	19.644	80.902	>180	>180
Shakesort	0.662	25.934	99.455	>180	>180
Insertionsort	0.661	10.612	47.461	>180	>180
Shellsort	0.684	0.670	0.784	1.274	2.013
Selectionsort	0.634	9.431	36.568	>180	>180
Ranksort	0.657	9.936	38.896	>180	>180
Quicksort Primeiro	0.594	8.877	18.946	97.984	>180
Quicksort Central	0.600	0.658	0.723	1.116	1.639
Quicksort Random	0.593	0.647	0.716	1.270	1.930
Quick Mediana 3	0.614	0.659	0.737	1.191	1.719
Mergesort	0.652	0.715	0.807	1.385	2.150
Heapsort	0.610	0.686	0.733	1.363	2.073
Radixsort	0.612	0.732	0.941	1.750	2.666
Radixbinsort	0.610	0.662	1.003	1.228	3.210





Analizando os resultados percebe-se que os algoritmos bubblesort, shakesort, insertionsort, selectionsort, ranksort e quickprimeiro possuem o tempo O(n^2), sendo assim, não possuem boa performance com entradas ordenadas decrescentes muito grandes, acima de 500 mil números. O algoritmo quick primeiro mais uma vez cai em seu pior caso, assim como na entrada crescente, a única diferença é que agora o pivô vai ter o maior número da entrada.

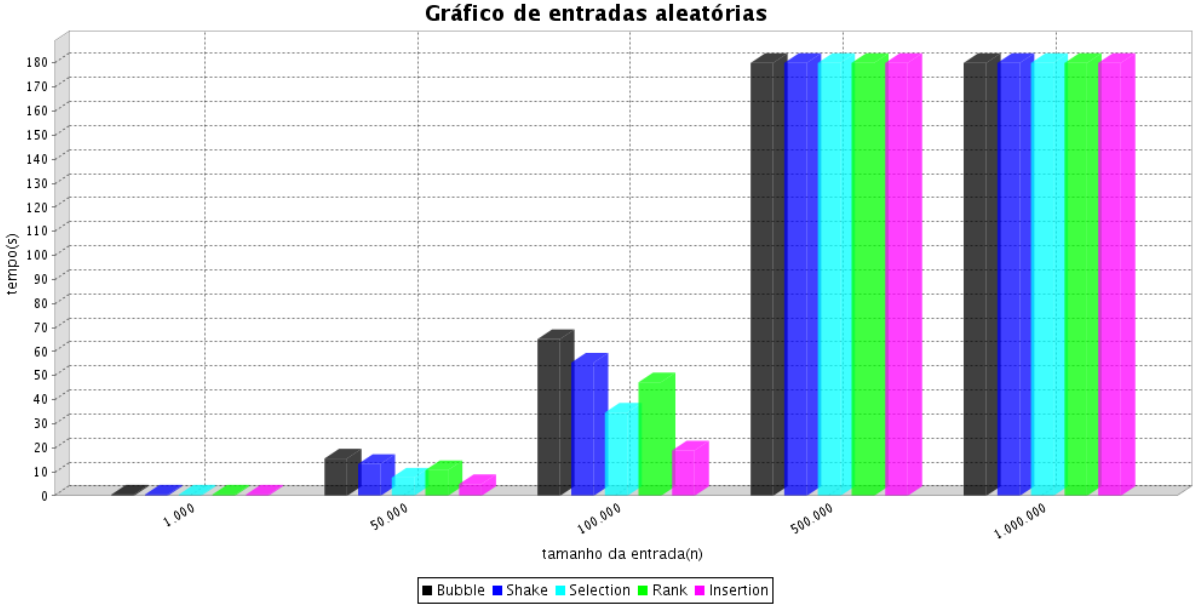
Os algoritmos insertionsort e shakesort, que tinham caído no melhor caso para a entrada anterior, neste caso caem no pior caso e passam a ter o tempo O(n^2).

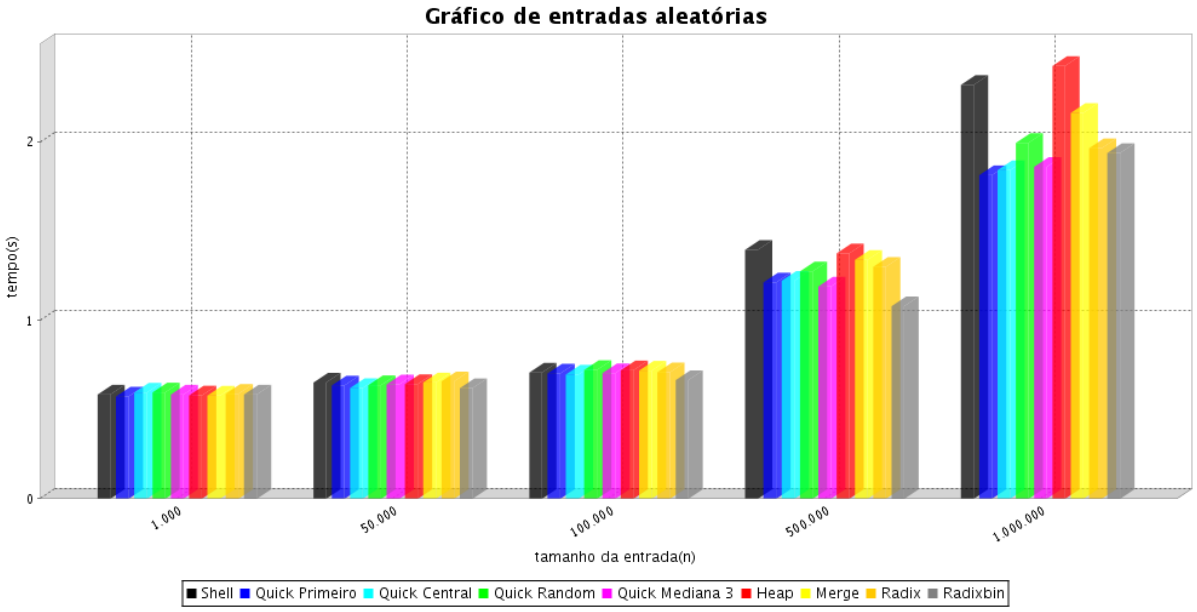
Os demais algoritmos continuam com suas performances muito boas.

- **Entradas aleatórias**

Tabela 3: Entradas Ordenadas Decrescentes

	Tamanhos				
Algoritmo	**1.000**	**50.000**	**100.000**	**500.000**	**1.000.000**
Bubblesort	0.590	15.548	65.186	>180	>180
Shakesort	0.593	13.276	55.433	>180	>180
Insertionsort	0.588	4.920	18.872	>180	>180
Shellsort	0.585	0.652	0.708	1.396	2.321
Selectionsort	0.578	7.608	34.625	>180	>180
Ranksort	0.593	10.683	47.232	>180	>180
Quicksort Primeiro	0.574	0.636	0.703	1.214	1.815
Quicksort Central	0.596	0.622	0.697	1.223	1.843
Quicksort Random	0.597	0.638	0.723	1.276	1.996
Quick Mediana 3	0.585	0.642	0.705	1.191	1.862
Mergesort	0.579	0.653	0.719	1.340	2.164
Heapsort	0.580	0.644	0.721	1.377	2.427
Radixsort	0.589	0.659	0.710	1.301	1.966
Radixbinsort	0.585	0.621	0.666	1.080	1.939





Analizando os resultados percebe-se que os algoritmos bubblesort, shakesort, selction, ranksort e insertionsort possuem as piores performances para entradas acima de 500 mill números, por serem algoritmos O(n^2). O quick primeiro neste tipo de entrada tem tempo O(n log n).

Os demais algoritmos possuem performances muito boas, mesmo para entradas muito grandes.

Conclusão

Neste trabalho foi possível verificar a complexidade de 14 algoritmos de ordenação. Percebeu-se que para as entradas ordenada crescente e descrescente o algoritmo Quick Sort Primeiro obteve tempos péssimos, pois caiu em seu pior caso, onde o pivô possui o pior valor possível (maior ou menor da entrada). Para estas mesmas entradas o algoritmo Quick Sort Central possui tempo O(n), pois cai no melhor caso, onde o pivô consegue ser o elemento central.

Em relação aos algoritmos Shake Sort e Insertion Sort, eles obtiveram os melhores resultados na entrada ordenada crescente, isso porque cairam no melhor caso, O(n). No restante eles voltam a ter tempo O(n^2).

Nos algoritmos que conseguiram fazer os testes para todos os tamanhos houve uma vantagem dos algoritmos Quick Sort, mas o Shell Sort merece um grande destaque, pois é simples de entender e facil de implementar.