

Avaliação de OAD

Nome:Rafael Nogueira Rodrigues **Matrícula:** 2210100445

Introdução

Neste trabalho, falarei das diversas funções de ordenação trabalhadas em aula, no qual mostrarei e explicarei o funcionamento de cada uma, mostrando e comparando em gráfico o comportamento delas nos casos em que um vetor é randômico, e com tamanhos variantes, sendo assim, as funções a serem trabalhadas foram separadas em grupo 1 e grupo 2, além da Radix Sort(que será falada no final da avaliação).

No grupo 1, temos o bubble sort, selection sort e insertion sort. Esses algoritmos são considerados mais simples e representados pela fórmula ($O(n^2)$). Enquanto o grupo 2 será composto por quick sort, merge sort e heap sort, sendo representados pela fórmula ($O(n \log(x))$).

Grupo 1

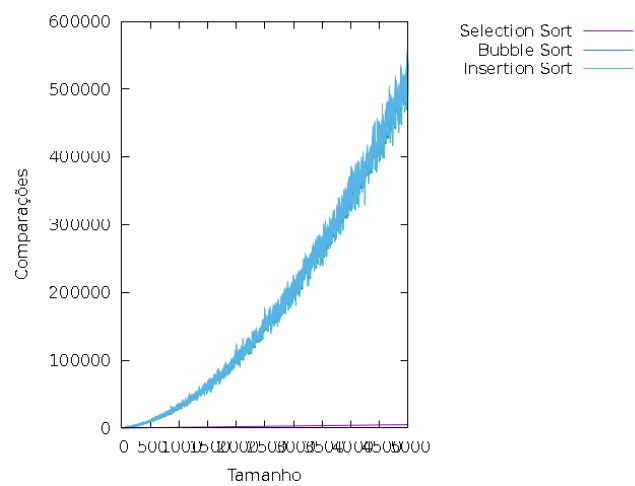
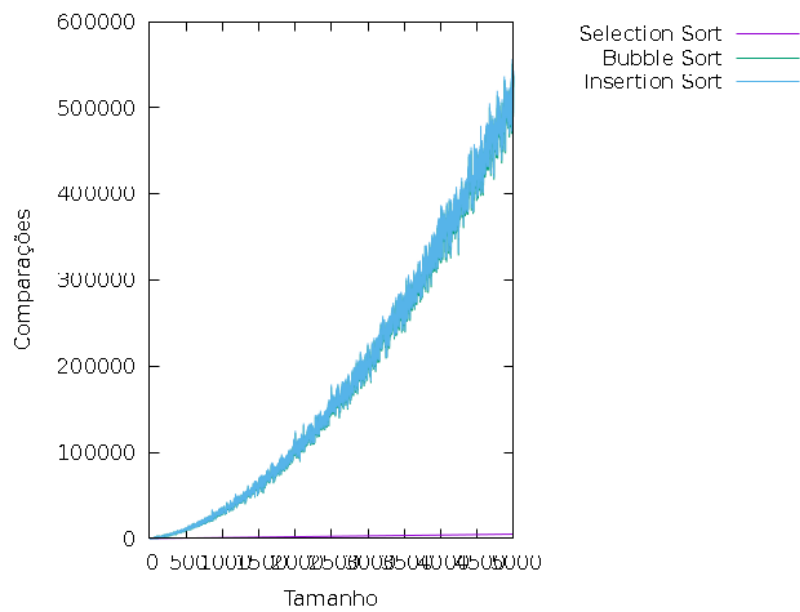
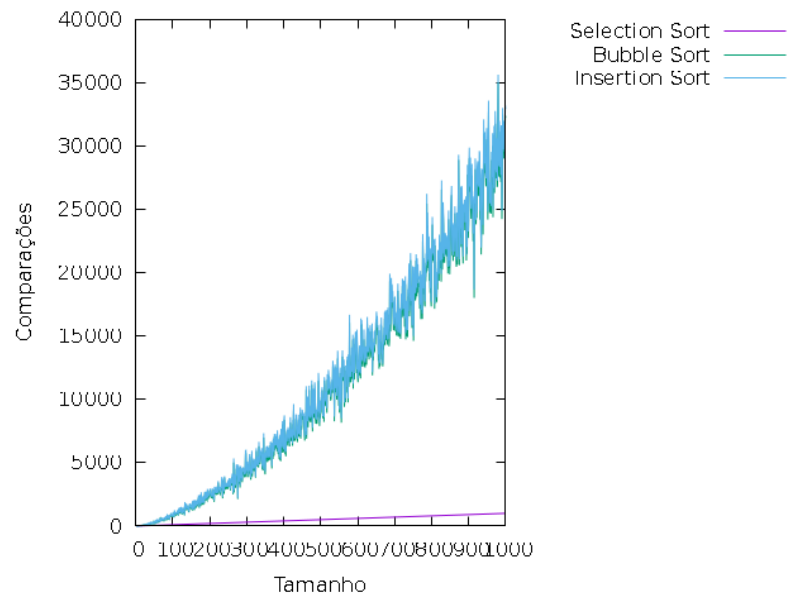
O grupo 1 é composto pelos algoritmos de ordenação bubble sort, selection sort e insertion sort. por mais que esses algoritmos sejam considerados mais simples em comparação aos outros do grupo 2, eles ainda desempenham um papel importante em determinados contextos e podem ser úteis em certas situações.

O bubble sort é considerado o algoritmo de ordenação mais simples, pois, sua função é unicamente comparar os elementos com os seus adjacentes, por exemplo, ele irá comparar o primeiro elemento com o segundo, caso ele seja menor e a ordenação esta sendo feita de forma crescente, o elemento será trocado, no caso do código desenvolvido, ele entrará na função swap para que isso ocorra, como ja foi dito, ele é simples, logo fará diversas comparações consideradas ineficientes e isso o torna inviável em algumas situações.

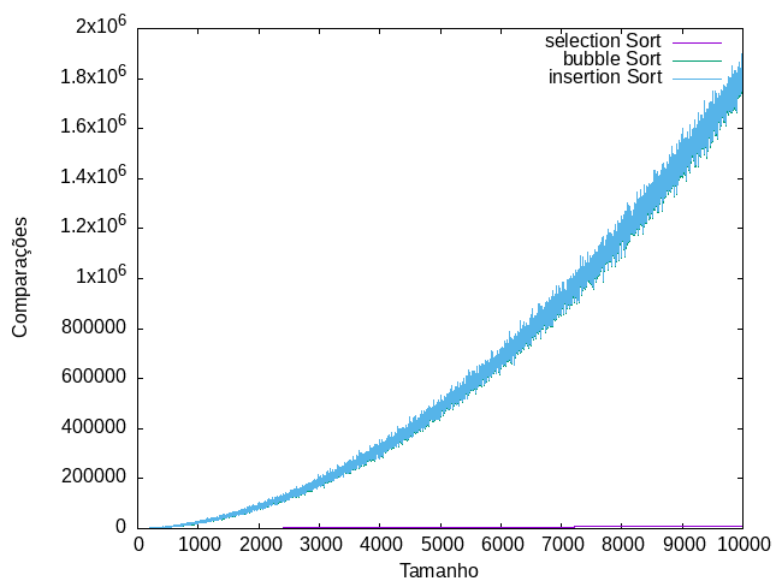
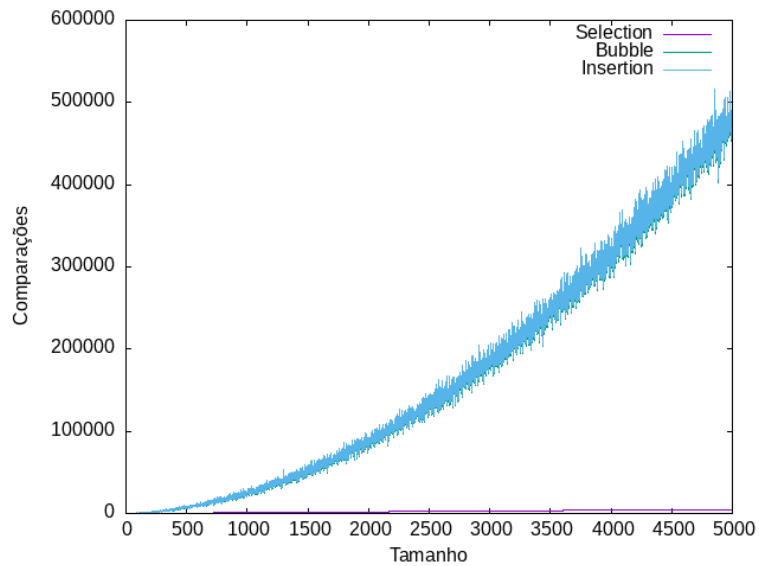
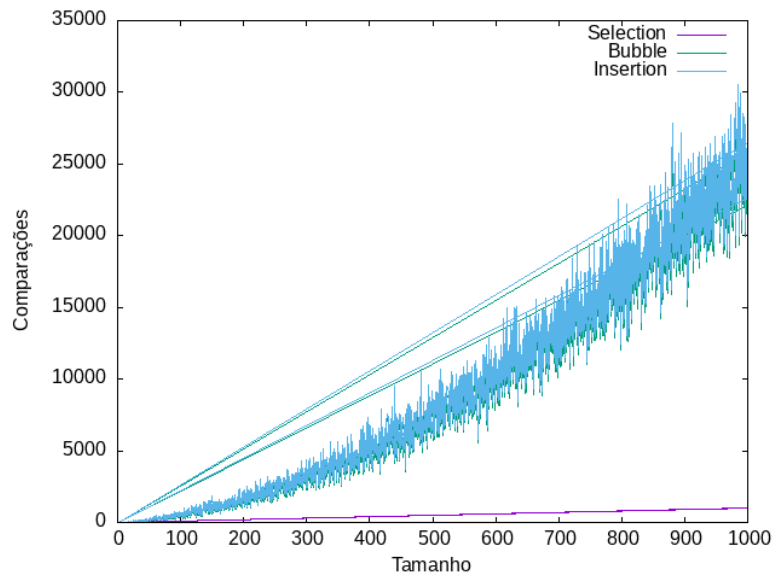
O selection sort também é um algoritmo simples de ordenação e sua característica é a que ele se baseia na posição(Index) de cada elemento, sendo assim, cada passo, o algoritmo encontra o menor elemento não classificado e o coloca na posição correta na sub lista ordenada. Isso se dará até que o vetor esteja ordenado, mais para frente do trabalho esse sort será descrito com mais detalhes.

O insertion sort é outro algoritmo de ordenação básico. uma de suas características é a condição dentro do 2 loop " $j > 0 \ \&\& \text{vetor}[j - 1] > \text{vetor} [j]$ ", isso faz com que ele compare o elemento com seus antecessores, e para que não haja problema o loop inicial começa pela segunda posição. Aqui está os gráficos de comparação do Grupo 1, eles foram gerados tanto no linux quanto no windows, então fiz nos 2 Sistemas operacionais

Imagens geradas no Windows



Imagens geradas nos Linux:



Primeira observação são os dados, cada imagem representa a variação no tamanho, primeiro no windows foi feito com o tamanho que chamaremos de P, ou seja, o vetor iniciará com

tamanho 1, irá até ter o tamanho 1000, cada loop, com uma incrementação de 1.0 segundo tamanho chamaremos de M, começará com tamanho 1, irá até tamanho 5000, e com incrementação de 1.0 terceiro tamanho chamaremos de G, começará com tamanho 1, irá até tamanho 10000 e com incrementação de 1. Um dos motivos para eu ter feito nos 2 SO foi a qualidade da imagem e para caso haja uma diferença entre.

Segunda observação que podemos notar nos gráfico, é possível ver que o Selection tem menos comparações isso se dá porque o selection sort seleciona o menor elemento não classificado e o coloca na posição correta na sub lista ordenada. Nesse processo, apenas uma comparação é necessária em cada iteração para encontrar o menor elemento. Essa estratégia permite que o selection sort tenha um número menor de comparações em relação ao bubble sort e ao insertion sort.

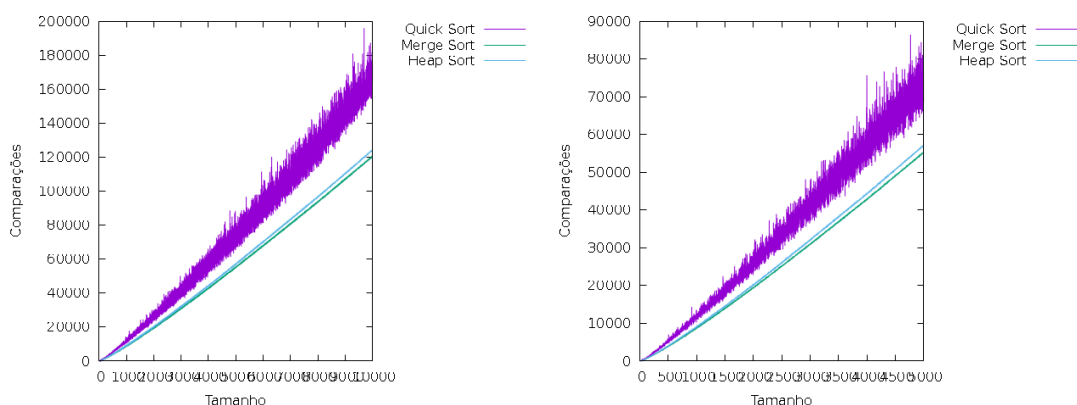
Grupo 2

O grupo 2 possui sim, uma complexidade a mais que o grupo 1, eles são formados pelo quick sort, merge sort e heap sort.

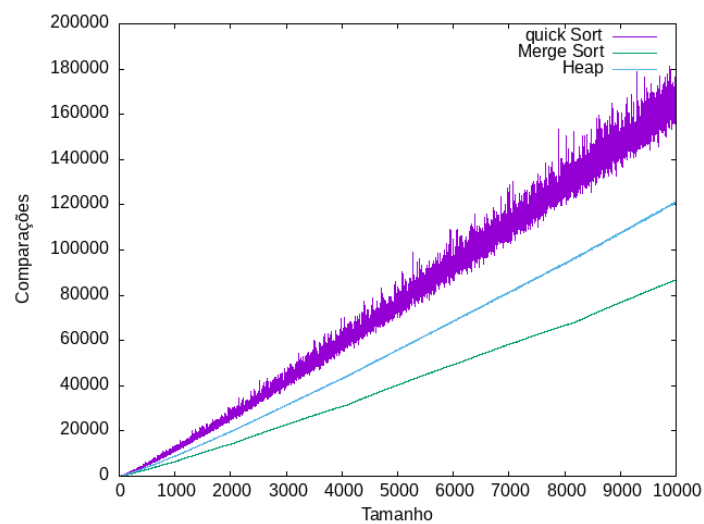
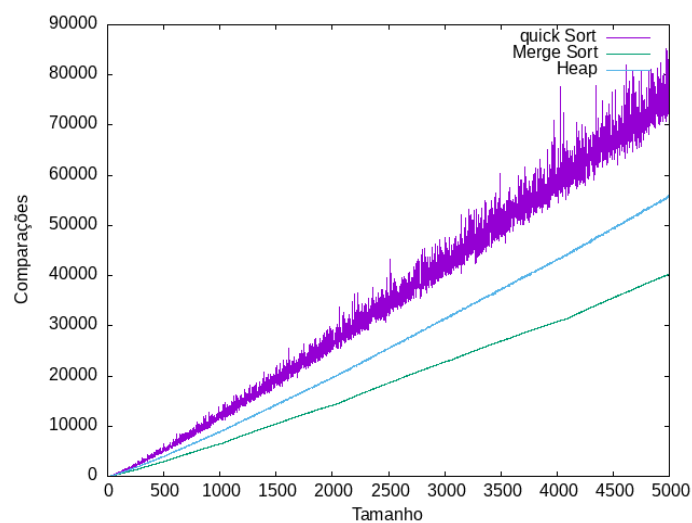
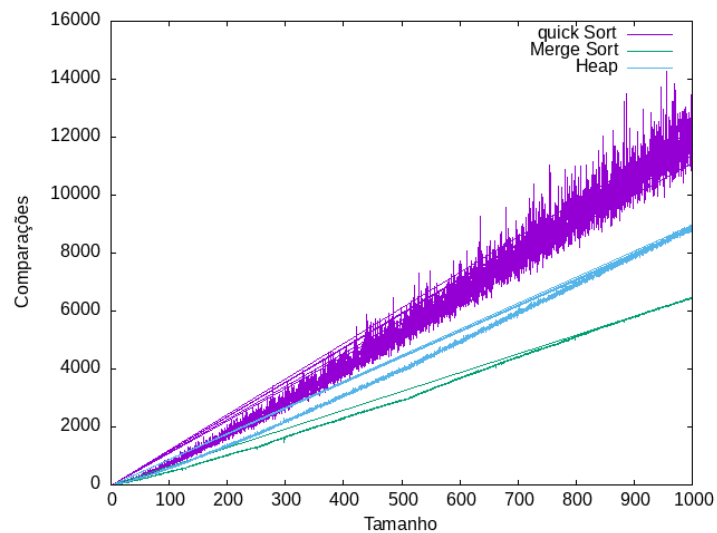
Basicamente tanto o quick sort , quanto o merge sort, possuem a prática de “dividir para conquistar”. Quick sort ele tem como grande característica a escolha de um pivô, no qual ele verificará se esse pivô é maior ou menor, então começar a ordenação de forma recursiva. Quanto ao Merge sort, sua maior característica é fato de que ele divide um vetor várias e várias vezes, até que ele possa começar a fazer trocas, isso permite com que ele desempenhe bem em grande numero de dados.

Heap Sort possui uma dinâmica diferente, e é comumente utilizado para descoberta de maior e menor valor. Isso se dá pois, ao pesquisarmos por ele, observamos que é julgado como uma arvore binária especial, no qual ele sempre extrai o valor da raiz, descobrindo assim o maior ou menor termo.

Imagens geradas no Windows:



Imagens Geradas no Linux:

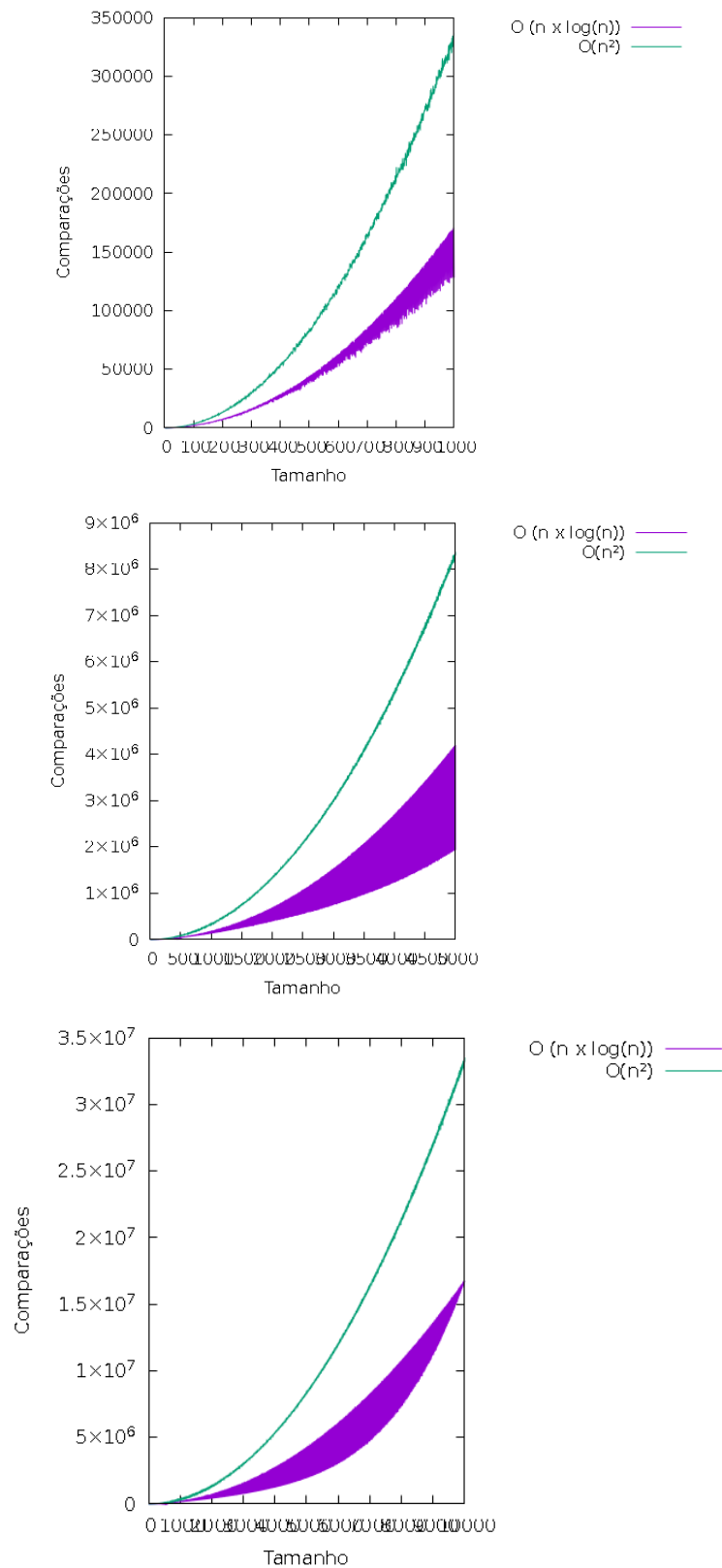


Esses gráficos também foram gerados naquele padrão visto anteriormente(P,M,G)
 Ao observarmos esses gráficos é perceptível que o quick Sort possui uma grande variância comparado aos outros sorts e o Merge Sort possui uma grande eficiência por causa da sua estratégia de divisão, e a tendência de diferença entre ele e os demais sort tende a aumentar conforme o tamanho do vetor.

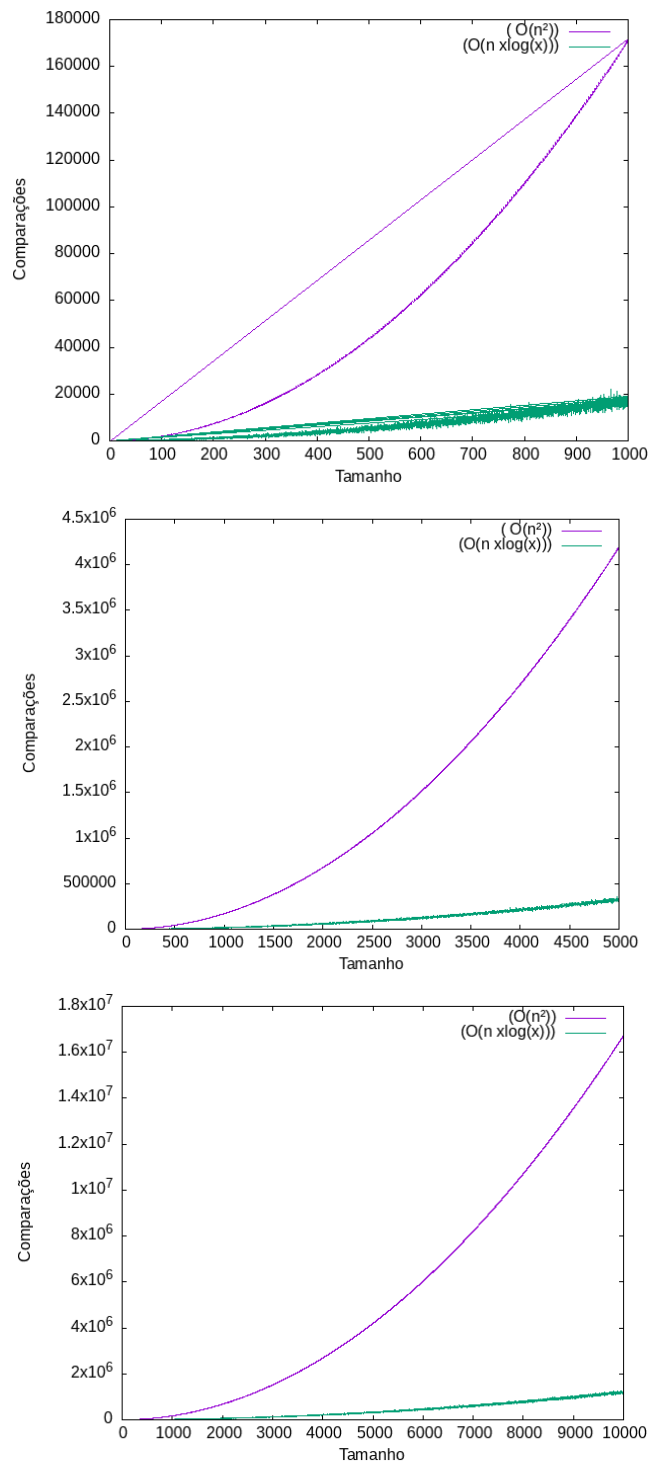
Grupo 1 x Grupo 2

Com base nos ultimos dados foi feita uma média de comparações dentro de cada grupo e os resultados obtidos foram:

Imagens do Windows:



Imagens do linux:

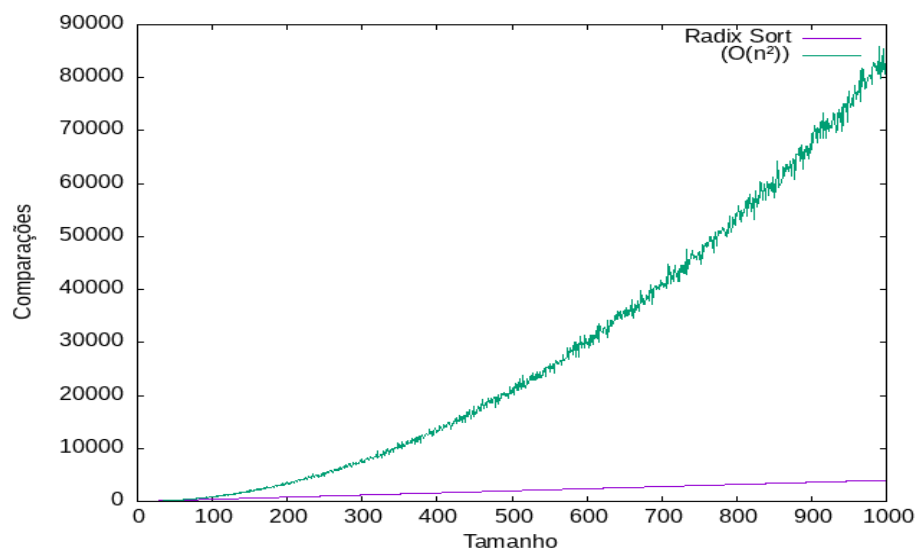
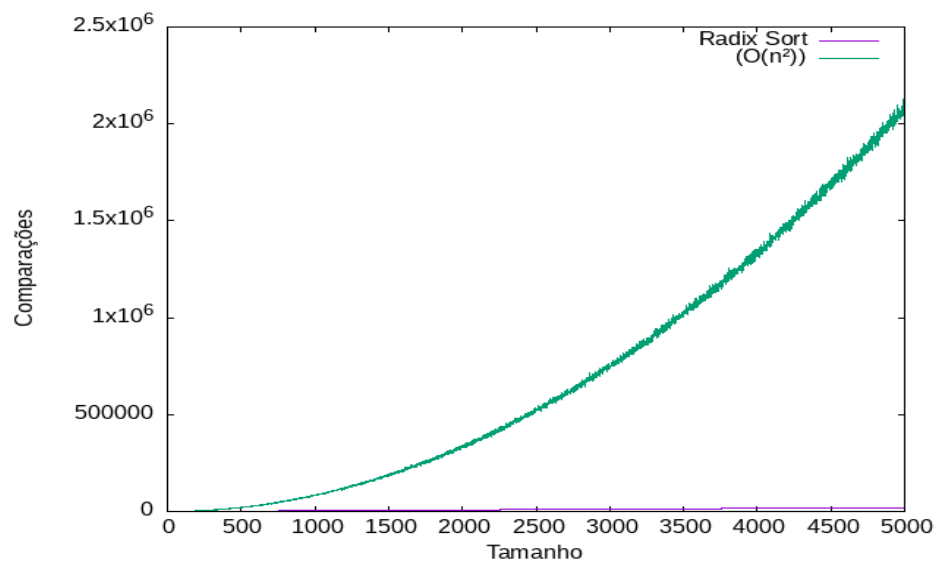
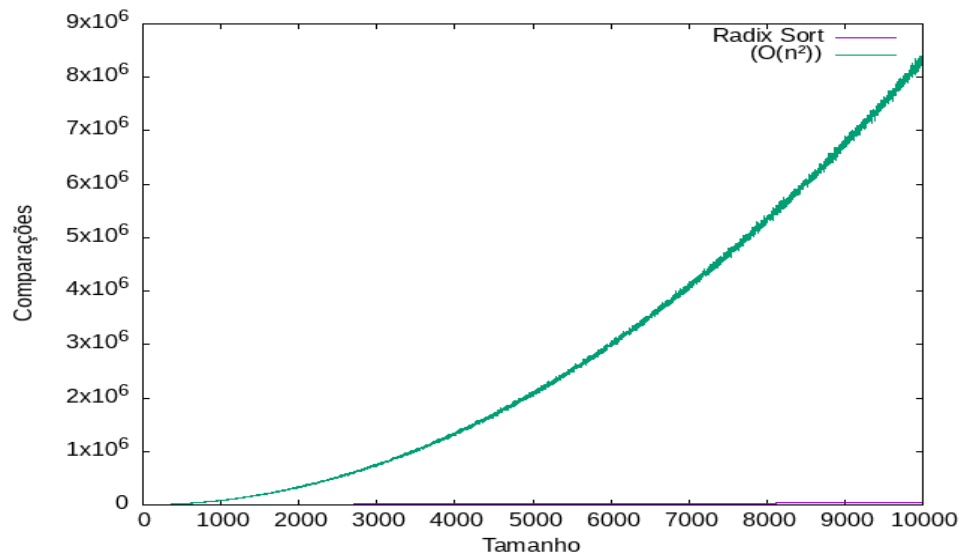


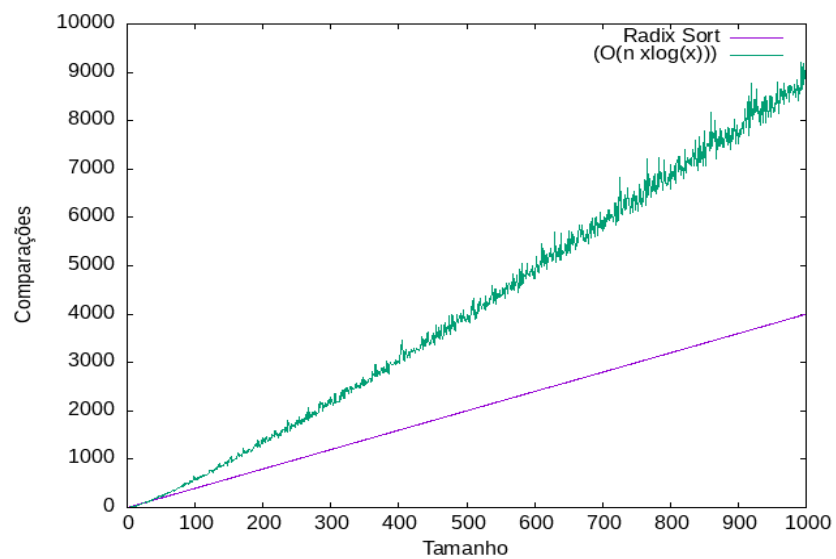
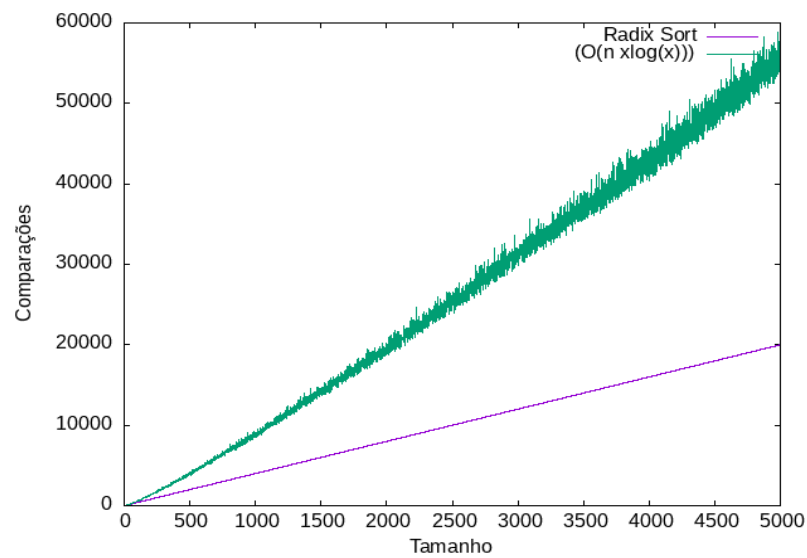
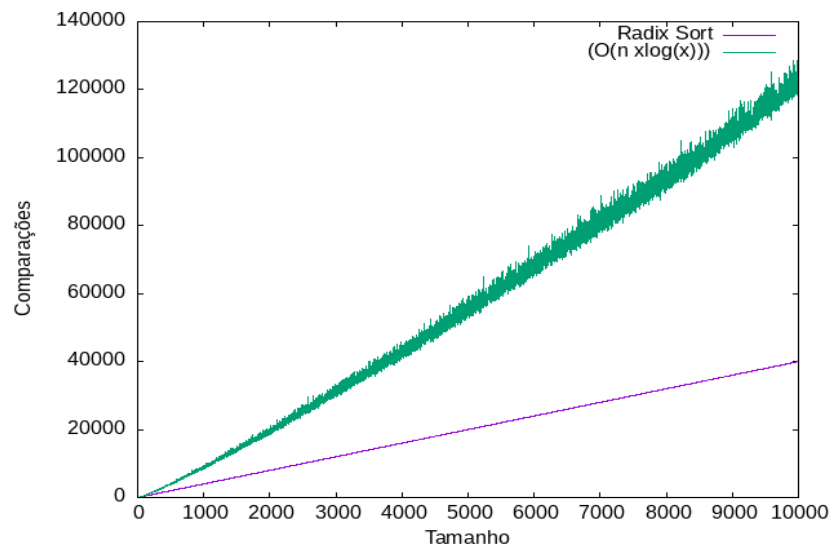
Observamos que há uma diferença nos gráficos gerados do windows e do Linux, porém ambos ainda mostram a grande eficiência das funções do Grupo 2, tanto para pequenos vetores quanto para grandes vetores, é importante ressaltar que o impacto que a quick sort faz na média do grupo 2 é muito mais aparente no windows.

Podemos concluir também que programas podem se tornar muito mais lentos se utilizarem grande números de dados juntamente as funções de $O(n^2)$, saber atribuir a função de ordenação certa melhorará a aplicação concerteza.

Radix Sort

Essa função de ordenação é representada pela fórmula $O(w + n)$, no qual ele ordena com base nos números de dígitos de cada um.





Uma característica interessante do Radix Sort é que ele é aparentemente uma função linear, separei entre Radix **X** $O(n \times \log(x))$ e Radix **X** ($O(n^2)$). independente de seu tamanho ele continua linear.

Conclusão

Podemos concluir que o uso das funções mais simples com complexidade $O(n^2)$ é útil apenas para baixos volumes de dados. Conforme o volume de dados aumenta, é recomendado utilizar funções com complexidade $O(n \log n)$, como por exemplo, as funções de ordenação baseadas em comparação. Por outro lado, em casos específicos em que a ordem dos números depende apenas da quantidade de dígitos, pode ser mais eficiente utilizar uma função de complexidade $O(w + n)$. Essa abordagem mantém os números organizados de forma eficiente com base em sua quantidade de dígitos.