Introdução a Ciencia da Computação 2
Relatório: Trabalho Ordenação e Busca

Alunos:

Ian Castilho Caldeira Brant Nº 10133967
Alex Galhardo Nº 10408344

Introdução

Esse relatório apresenta os resultados obtidos em testes de algoritmos de ordenação e busca. Para esses estudo, foram desenvolvidos dois códigos em C, um para ordenação e outro para busca. O código de ordenação conta com os algoritmos Quicksort, Heapsort e Counting Sort, que fazem ordenação em diferentes chaves com dados de países do mundo. O código de busca conta com os algoritmos de Busca Sequencial, Busca Binária, Busca por Interpolação e busca por árvore binária de busca, e faz buscas por diferentes chaves em um banco de dados de municípios brasileiros.

Para cada algoritmo, foi variado o número de elementos a serem ordenados ou na base de busca. Essa variação foi feita duplicando os bancos de dados manualmente. Ou seja, n corresponde ao número de itens contidos originalmente nos bancos de dados. 2n é duas vezes esse valor, etc. Em alguns casos também foram feitos testes com diferentes chaves, para se entender se há diferença no tempo.

As bases de dados foram tratadas para que cada campo fosse separado por espaços. Espaços nos nomes foram substituídos por _.

As curvas comparativas nos gráficos foram devidamente adaptadas, através de seus coeficientes, para se adequarem visualmente aos gráficos.

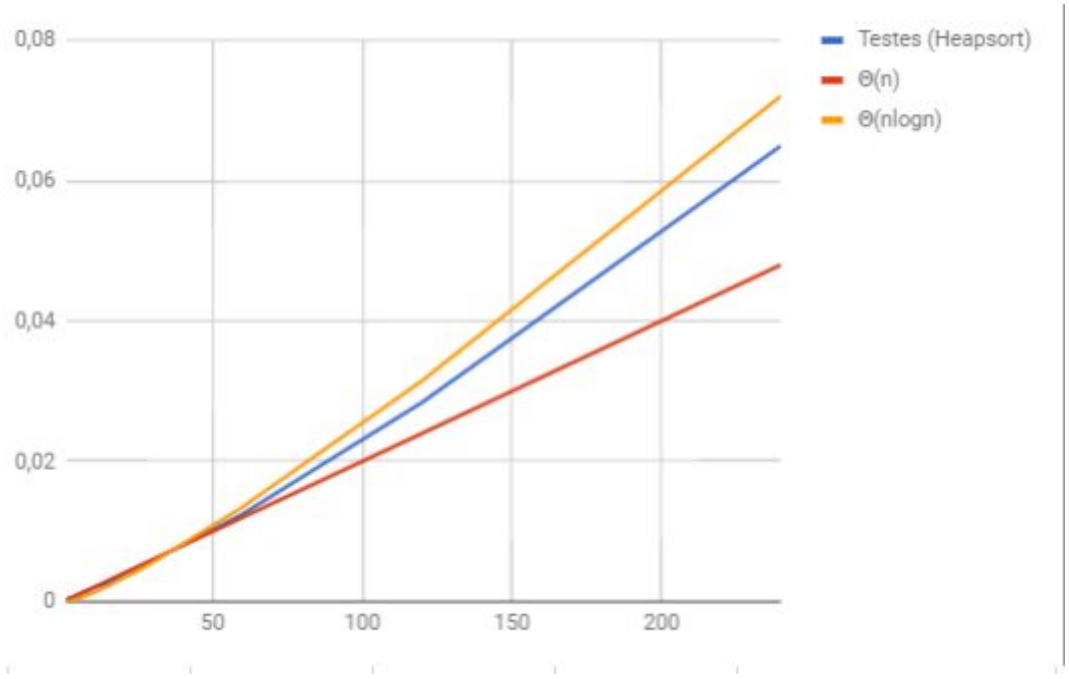
O arquivo com os valores dos testes foi enviado em anexo.

Algoritmos de Ordenação

Heapsort

O Heapsort foi testado com três diferentes tipos de chave de ordenação: População, Nome (ja ordenado) e Area. Não foram observadas diferenças no tempo de ordenação para diferentes tipos de chaves.

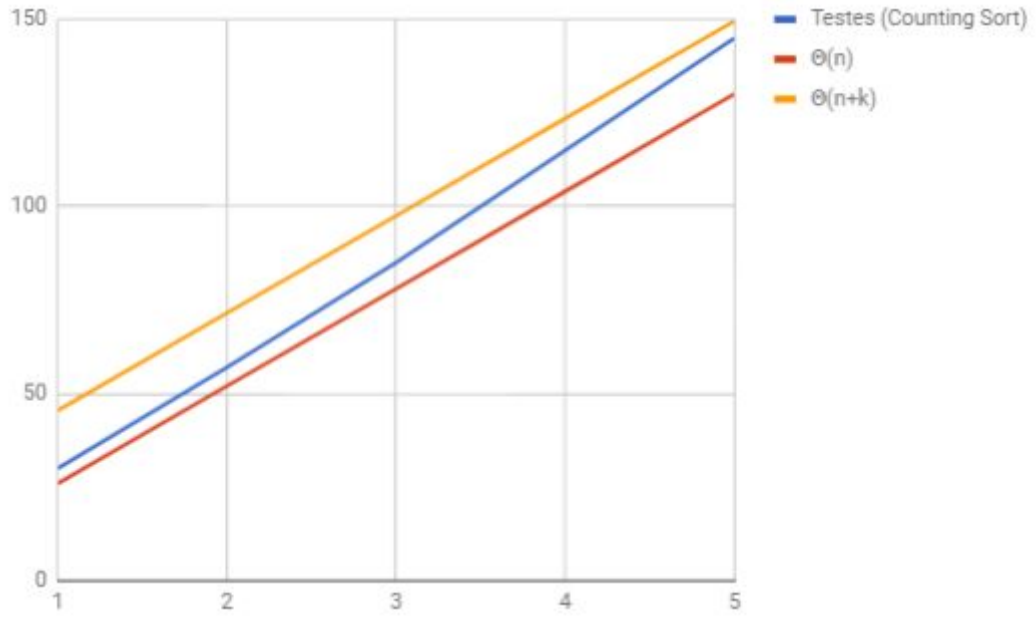
No caso da população, variou-se o tamanho da base de dados a serem ordenados e para cada tamanho foi medido um tempo, obtendo os resultados apresentados no gráfico abaixo. O tempo foi medido em nanosegundos:



Dessa forma, podemos ver que os resultados obtidos são condizentes com o esperado, já que sabemos que o Heapsort possui complexidade Θ(nlogn) no caso médio.

Counting Sort

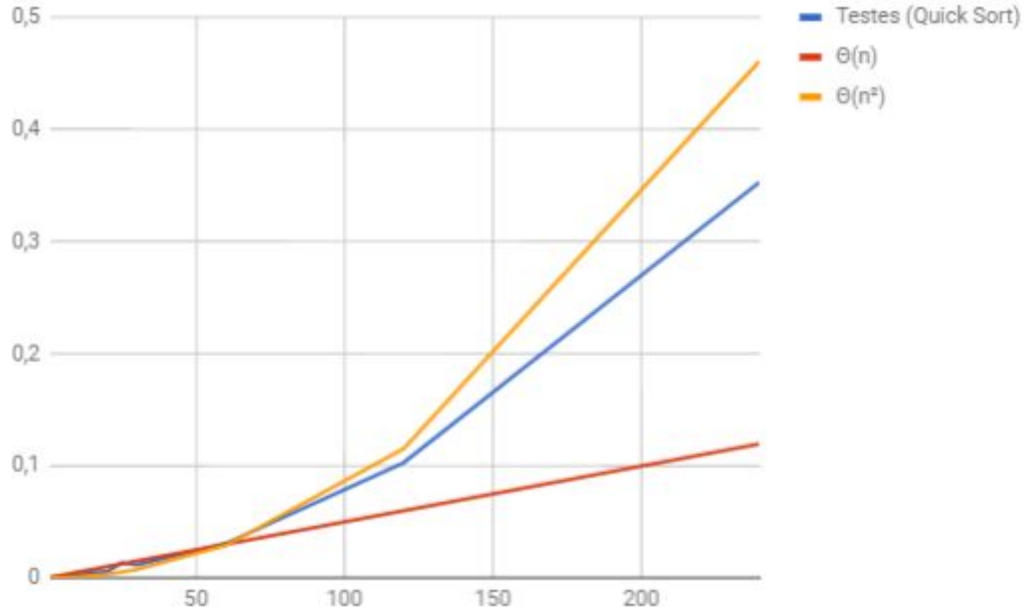
O counting sort precisaria ser feito com um numero inteiro, e dessa foram escolhidos os valores de área de cada país. Como esse algoritmo tem complexidade proporcional também ao maior valor na base a ser ordenada, e os valores de área são grandes, ele teve tempo de execução bem maior que os demais. Por isso seu tempo de execução foi medido em segundos utilizando um cronômetro, e os resultados são apresentados abaixo:



Podemos ver que, apesar de algumas flutuações, o algoritmo se comporta da forma esperada, tendo complexidade proporcional a Θ(n+k).

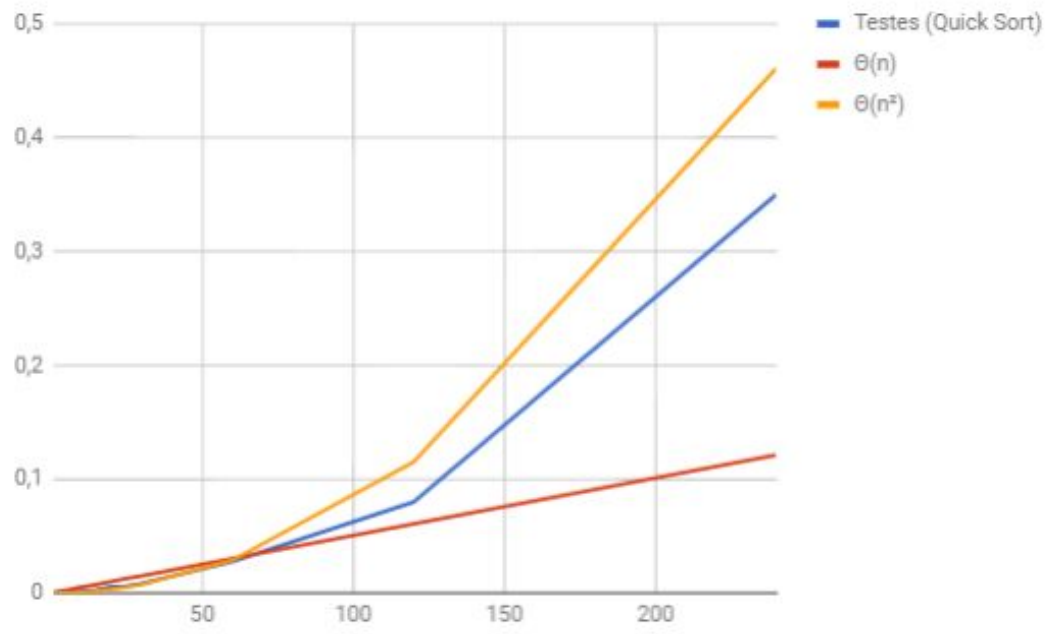
Quicksort

O Quicksort foi estudado em duas diferentes situações. Em uma foi escolhido o nome como chave de ordenação e o primeiro elemento como pivo. Dessa forma seria o pior caso do algoritmo, pois o pivo seria o menor valor. Os resultados são apresentados abaixo:



Podemos ver que nesse caso o tempo cresce proporcionalmente a uma função $\Theta(n^2)$, como era esperado do Quicksort em seu pior caso.

Na outra situação, também foi escolhido o Nome como chave de ordenação, mas como pivo foi escolhido o elemento do meio. Nas variações do valor de n, a lista de paises sempre era ordenada pelo nome, para que o pivo fosse sempre a mediana dos valores, obtendo o melhor caso do Quicksort, obtendo o seguinte resultado:

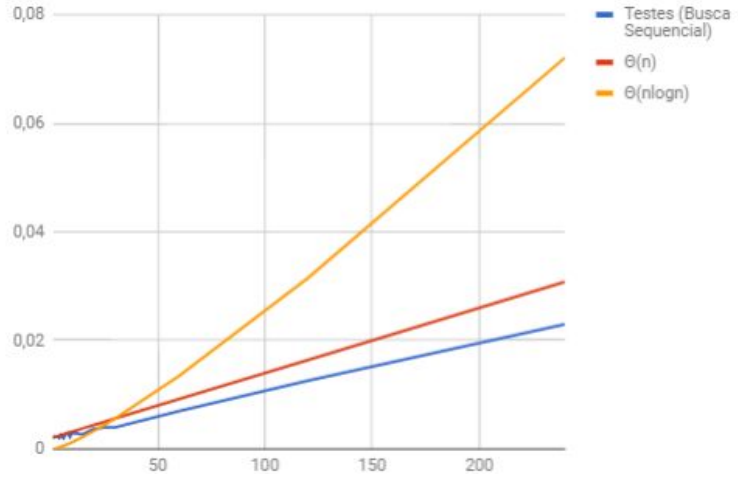


Também nesse caso foi obtida uma complexidade proporcional a uma função Θ(n²). No entanto, para esse caso era esperada uma complexidade Θ(nlogn). Essa discrepância pode ter sido obtida devido a alguma má execução do algoritmo, que não conseguimos identificar.

Algoritmos de Busca

Busca Sequencial

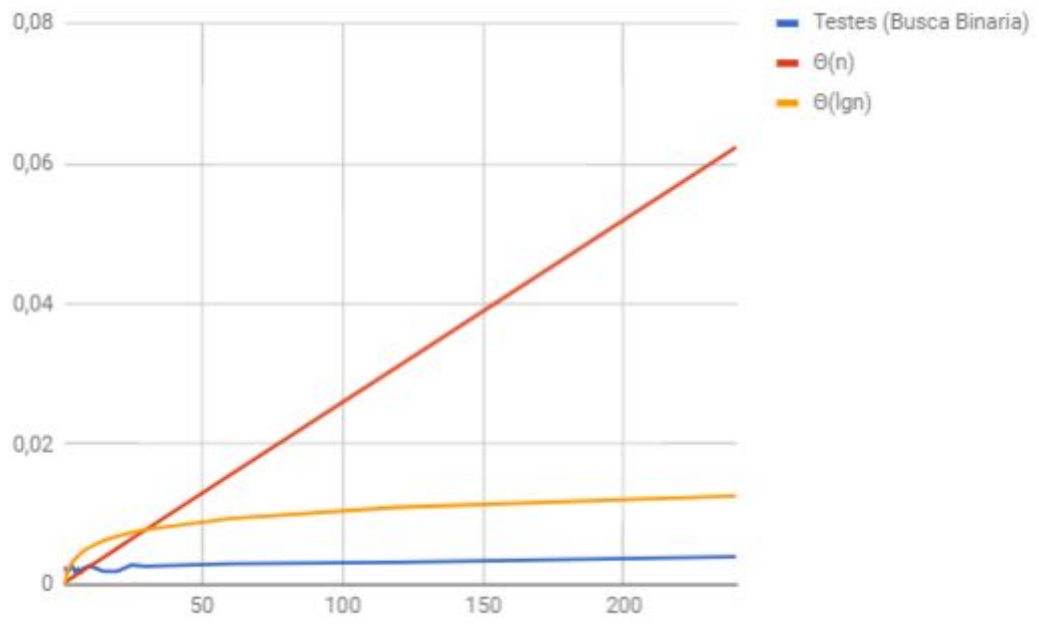
A busca sequencial foi testada utilizando o nome como chave. Em cada teste, foi inserido manualmente um municipio ficticio ao fim da lista, de forma que era buscado sempre o ultimo municipio da lista, obtendo assim o pior caso. Os resultados sao apresentados abaixo:



Podemos ver que a busca sequencial no pior caso tem complexidade propocional a uma função Θ(n), como já era esperado.

Busca Binaria

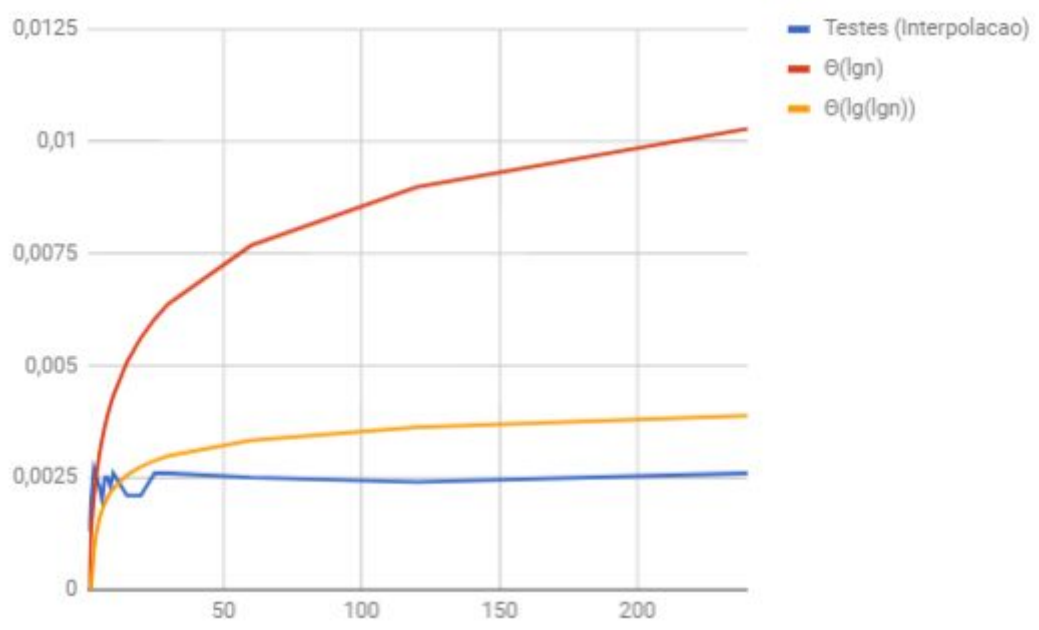
A busca Binaria também foi testada utilizando o nome como chave de busca, utilizando sempre um municipio posicionado aleatoriamente, de forma a se obter o caso médio:



Os resultados encontrados são condizentes com o esperado, de que a busca binária tenha no caso médio complexidade Θ(lgn)

Busca por Interpolação

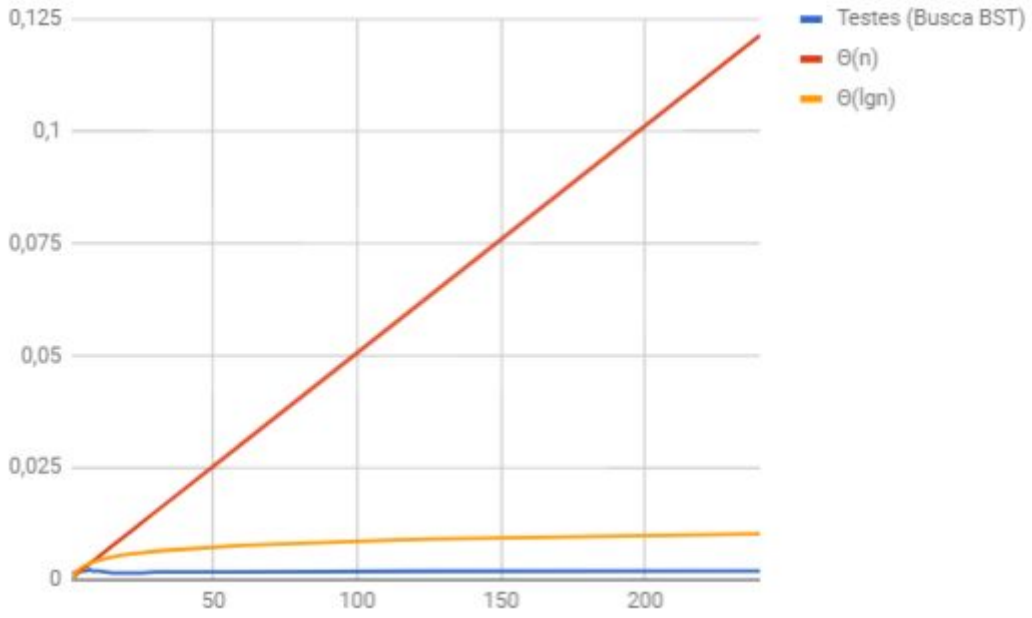
A busca por interpolação foi testada utilizando os códigos dos municípios como chave de busca. Além disso, foi utilizado um municipio aleatorio em cada teste, obtendo os resultados abaixo:



Como os dados estavam distribuídos uniformemente, era esperada uma complexidade Θ(lg(lgn)) da busca por interpolação. Apesar da imprecisão na medição não permitir ver esse comportamente tão claramente, é possível ver que os resultados obtidos não são inconsistentes com essa hipótese.

Busca por Árvore Binária de Busca

Para os testes com esse tipo de de busca, após a leitura dos dados eles foram inseridos em um árvore binária de busca. Feito isso, foram feitos testes buscando os códigos dos municípios, também sempre um município distribuído aleatoriamente na base de dados. Os resultados são mostrados abaixo:



Apesar de também não ser possível observar tão claramente devido a pouca variação nos valores, o gráfico sugere que a complexidade do algoritmo é proporcional a Θ(lgn), que é o esperado da árvore binária de busca no caso médio.

Um ponto notado nesse estudo, foi de que o tempo gasto para se organizar os dados em uma árvore binária de busca, é muito maior que o tempo gasto por qualquer outro algoritmo de ordenação. Dessa forma, organizar a estrutura para depois fazer a busca não é uma boa escolha. Esse algoritmo será útil apenas quando os dados já estiverem organizados em um BST