Análise de comportamento de Algoritmos de Busca e Ordenação em um Sistema Bancário

Gustavo Antônio C. Alcântara[1], Pedro H. Chaves[1]

[1] Ciência da computação – Instituto Federal do Sul de Minas
Caixa Postal 37903-358 – Passos – MG – Brasil

gustavo.alcantara@outlook.com, phedroxaves@hotmail.com

Abstract. *Provide performance information for search algorithms to search for specific records in a banking institution. In this work, search techniques and ordination techniques were used to generate experiments and results. The main objective is to verify which method of searching and ordering is more efficient for a given situation. The results obtained showed that some ordering algorithms may be more efficient than sequential search, but also the sequential search was more efficient than some ordering methods.*

Resumo. *Prover informações de desempenho de algoritmos de busca para a procura de registros específicos em uma instituição bancaria. Neste Trabalho, técnicas de busca e técnicas de ordenações foram utilizadas para geração de experimentos e resultados. O objetivo principal é verificar qual método de busca e ordenação e mais eficiente para determinada situação criada. Os resultados obtidos demostraram que alguns algoritmos de ordenação podem ser mais eficientes que a busca sequencial, mas também a busca sequencial se mostrou mais eficiente que alguns métodos de ordenação.*

1.Introdução

Atualmente corporações bancarias comportam uma grande quantidade de informações referentes aos clientes, nas quais estão contidas uma extensa base de dados. Com isso as instituições buscam um método de verificação de dados específicos, utilizam-se algoritmos de busca tradicionais como algoritmo sequencial e binária para realizar a busca de informações em particular.

Nesse artigo criamos uma instituição financeira fictícia, para realização dos testes com os algoritmos. Foi criado também valores fictícios para implementação, foram utilizados cem mil cadastros com valores distintos.

O uso de algoritmos de busca tem sido de grande ajuda para promover uma melhor organização dos dados.

Este trabalho tem como objetivo verificar o rendimento dos algoritmos de busca e de ordenação, e observar sua aplicação, utilizando um arquivo com cadastros contendo cem mil registros com valores distintos. E mostrar qual algoritmo e mais eficiente para aplicação.

2. Referencial Teórico

Para realização do trabalho, foi utilizado alguns algoritmos de busca para obter a chave indicada, sendo eles busca sequencial e busca binaria. A busca binária necessita

que os dados de cadastro estejam em ordem do menor para o maior. Com isso empregou-se os seguintes algoritmos ordenação, como Merge Sort, Heap Sort, Shell Sort, Selection sort, Bubble Sort, e Incertion Sort.

2.1 Algoritmos de busca

-Busca sequencial: Sendo o método de busca de chave mais simples, sendo que sua concepção é ir verificando cada elemento do conjunto de forma sequencial até que o elemento seja encontrado. Sua complexidade em melhor caso pode ser O(1), médio O(n/2) e seu pior caso O(n).

-Busca binaria: É o método de busca mais eficiente comparando-se com o método sequencial. Para ocorrer a verificação de busca binaria é necessário que o vetor esteja ordenado, com isso ele é dividido, ele verifica o primeiro elemento, se não for ele, ele verifica o número comparado se for maior ele verifica a outra parte com valores maiores que o comparado. Sua complexidade em pior caso é O (log n), médio O(log n) e melhor caso é O(1).

2.2 Algoritmo de ordenação

-Merge Sort: tem como base o fundamento dividir para conquistar. Sua ideia é dividir o problema em vários subproblemas, utilizando a recursividade, com isso o merge tem um alto gasto de memória. Possui complexidade O (n log n), algoritmo estável, e não e adaptável.

- Heap Sort: o heap pode ser representado por uma árvore binaria, os elementos são ordenados quando são inseridos na estrutura. No final das interações os elementos maiores podem ser removidos da raiz heap. O algoritmo possui complexidade O (n log n), sendo não estável e pode ser adaptável.

-Bubble Sort: sendo um dos algoritmos mais simples, sua concepção é percorrer o vetor diversas vezes, sendo que cada passagem o maior valor flutua para o topo onde fica maior elemento da sequência. Sua complexidade é O (n^2), não é adaptável e também é instável.

-Insertion Sort: a ordenação por inserção, constrói uma matriz final com um elemento de cada vez, realizando uma inserção de cada vez. O modo de ordenação é parecido com uma organização de cartas de baralho, onde as cartas são recebidas é organizada de forma aleatória, durante a organização cada carta é colocada na sua posição correta e movendo as demais. Sua complexidade é O(n^2), sendo estável e adaptável.

-Selection Sort: funciona colocando sempre o menor valor do vetor na primeira posição, o segundo maior na segunda posição, e assim sucessivamente para todo os n números.

-Shell Sort: sendo o mais eficiente algoritmo, de complexidade quadrática, o shell é uma versão atualizada do insertion sort. Foi criado por Donald Shell em 1959, seu funcionamento considera vários segmentos sendo utilizado um método de inserção direta em cada um dos segmentos. O algoritmo passa algumas vezes pela lista dividindo em grupos menores, nesses grupos menores são utilizados a ordenação de inserção. Possui complexidade desconhecida, não há estabilidade, e não é adaptável.

3. Materiais e Métodos

As análises dos algoritmos foram realizadas em uma instituição bancaria fictícia onde fica armazenado cem mil registros de clientes fictícios. Para garantir que dados fossem salvos foi utilizado um arquivo, neste arquivo ficam armazenados os dados referentes ao cadastro, código do cliente, nome do cliente, número da conta e o saldo.

Para criação dos dados de cadastro foi utilizado uma função, onde fica responsável pela criação valores distintos e aleatórios, podendo os valores ser de 1 a 1 milhão.

Para execução do algoritmo foi considerado que a quantidade da busca inicial é igual a cem, e ir aumentando de 10 em 10 o número de comparações até que se encontre um valor específico. O total de chaves buscadas foi de 5000 chaves.

Foi implementado uma função onde chama todos os métodos de ordenação e busca, com essa chamada se realiza a medição de tempo de cada chamada para a realização da comparação de cada método.

Para saída do algoritmo foi criado outro arquivo com nome “saída.txt”, onde contém todas as informações obtidas por meio da análise do algoritmo de busca sequencial, busca binaria e os algoritmos de ordenação. Mostrando a quantidades de chaves buscadas e o tempo de cada algoritmo.

4. Experimentos computacionais

Após algumas execuções foi observado que cada execução do algoritmo apresenta valores diferentes, mas sendo valores bem próximos. Para não ocorrer interferência nos resultados foi utilizado somente um arquivo, sendo assim foram empregados os mesmos valores para cada execução.

Os resultados foram gerados em um computador com a seguinte configurações, processado I3 7100, 8 Gb de memória ram ddr4 2400mhz, placa gráfica Rx 240 2gb, disco rígido de 1 tb, sistema operacional Windows 10 64 bits Education. A ferramenta de compilação foi realizada no DEV- C ++ versão 5.11, os arquivos de entrada e saída foi feitos pelo bloco de notas.

Para oferecer dados mais precisos será considerado três execuções do algoritmo. Nas 3 execuções ira ser feita com os mesmos valores. Cada execução ira ter seu gráfico mostrando tempo de cada método.

Figura 1 resultado da primeira execução do algoritmo.

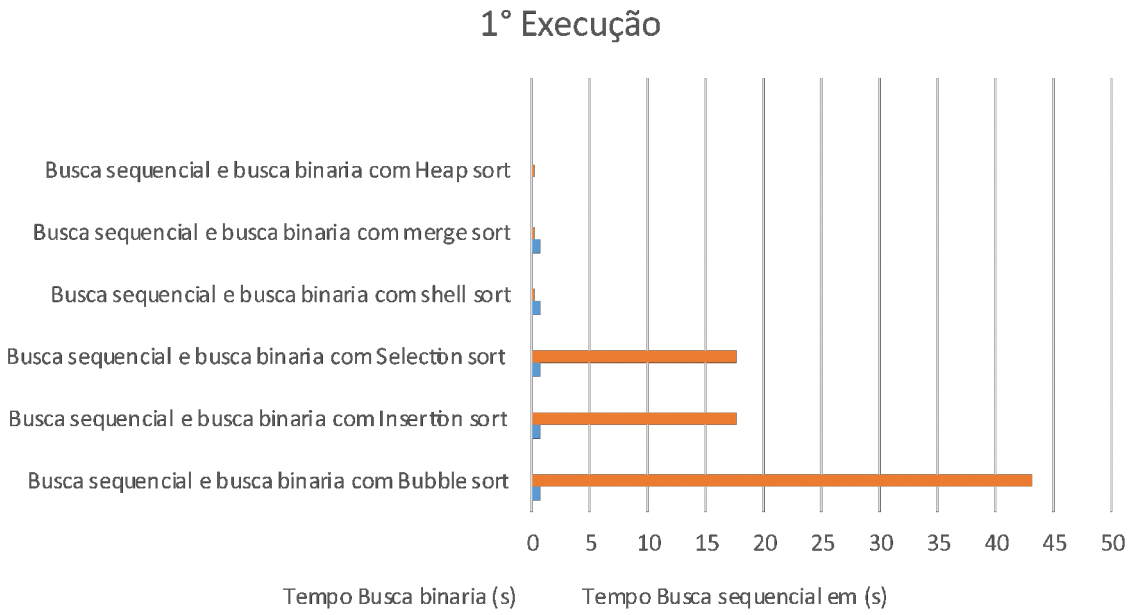


Figura 2 mostras resultados da segunda execução.

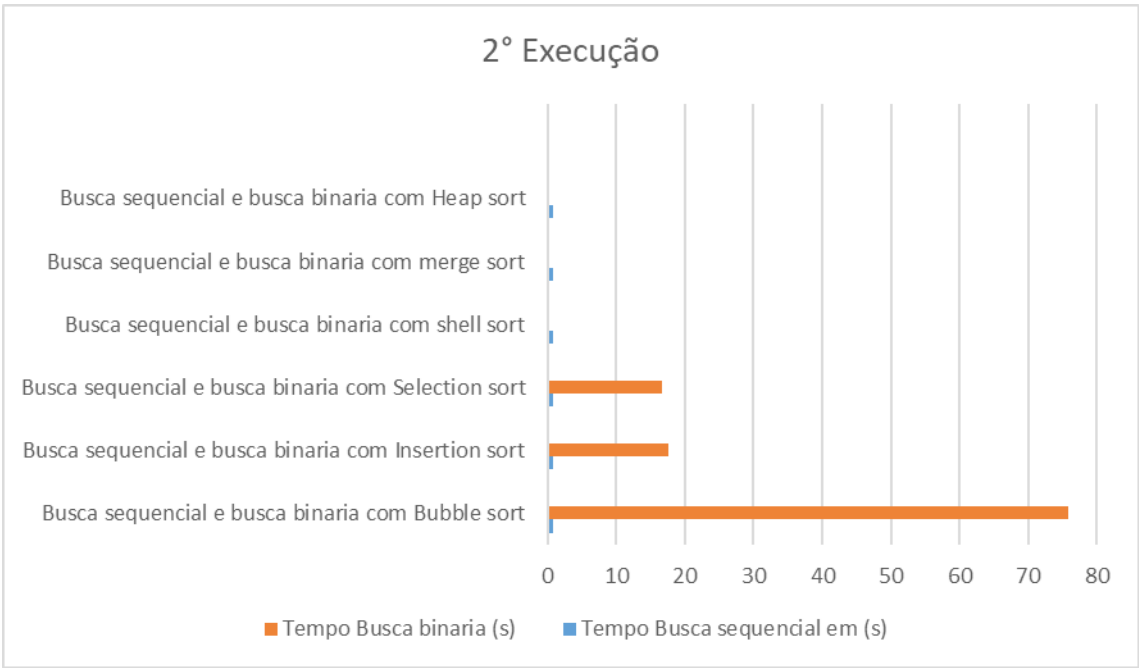
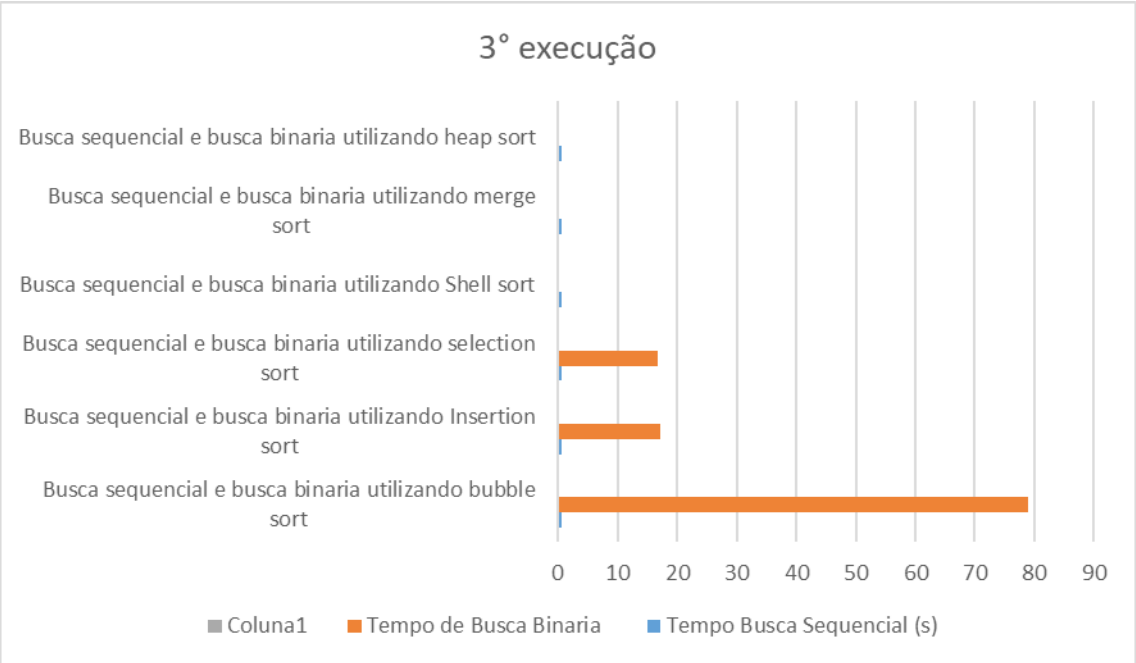


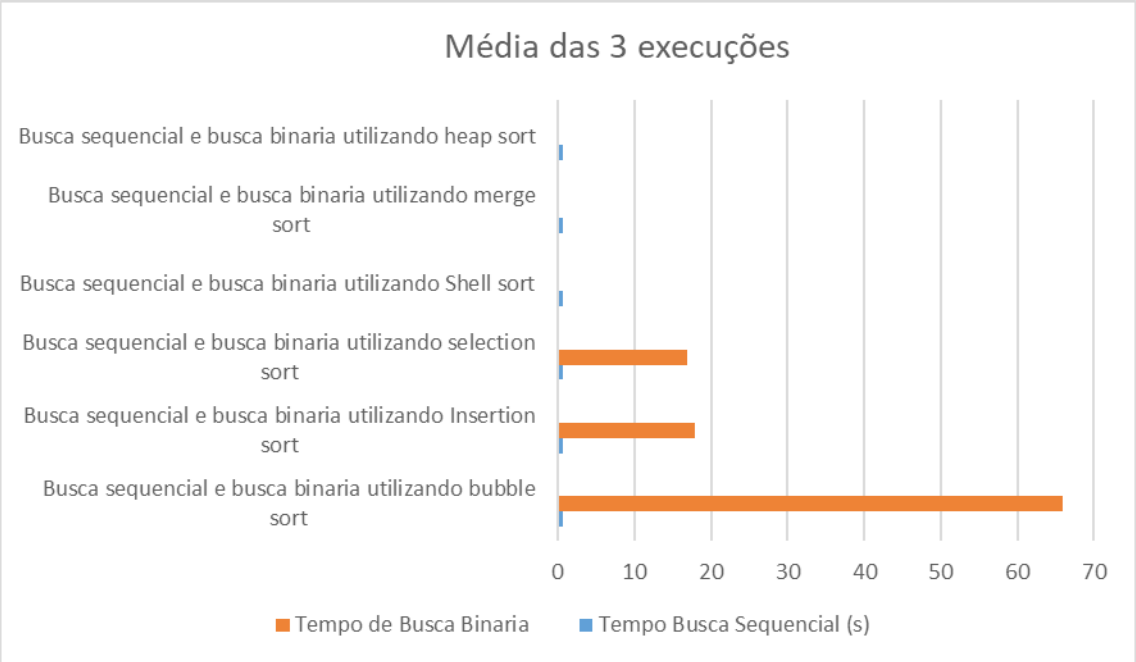
Figura 3 mostra o resultado da 3 execução.



Nas figuras 1,2 e 3 foi verificado que o algoritmo de ordenação Bubble sort foi o que mostrou maior variação de tempo, já os outros algoritmos foram mais precisos mostrando taxa de variação de tempo menor.

Com a variação de tempo, foi feito um gráfico com resultado médio das 3 execuções.

Figura 4 mostra o resultado médio das 3 operações.



Com o gráfico da figura 4 é possível tirar resultados mais consistentes. Nos resultados foi visto que a busca sequencial é mais eficiente que a busca binaria, como no caso do bubble sort, insertion sort, e selection sort, os 3 algoritmos mostraram um tempo bem elevado superior a 15 segundos. Os algoritmos mais eficientes foram shell

sort com tempo de 0,035 s, o merge sort ficou com tempo de 0,074 s e o heap sort se mostrou o mais eficiente com tempo de 0,03 s.

5. Conclusão

Acompanhar os resultados desta análise se mostra um fator importante, permitindo mostrar com qual algoritmo é melhor para determinado objetivo.

A partir desta análise é possível verificar qual algoritmo seria mais rápido, e qual seria melhor para determinada implementação.

O algoritmo realizado em questão teve alguns em imprevistos, como ele o Quick sort, onde não foi possível sua implementação**.** Outra dificuldade foi na parte da criação dos cadastros, como a função rand gera valores aleatórios, seu limite máximo dificultou a implementação. Outro ponto crítico foi a parte onde realiza chamada de todas as funções de busca. Como o bubble sort deu um resultado muito alto, e os outros algoritmos sendo mais eficientes o gráfico gerou valores muito baixo, como no caso do heap sort que seu tempo foi de 0.3s não ficando legível no gráfico.

Com resultados foi visto que o melhor método de busca, seria a busca binaria com ordenação heap sort e o pior foi o bubble sort que mostrou um tempo elevado, e possuindo variação de tempo.

6. Referencias

-TANENBAUM, A.; AUGENSTEIN M.; LANGSAM Y. Estrutura de Dados Usando C. 1. ed. São Paulo: Pearson, 1995. P.408-501

- CORMEN, T. H; RIVEST, R. L.; LEISERSON, C. E; STEIN, C. Algoritmos: Teoria e Prática. Tradução da 3ª edição Americana. São Paulo: Editora Campus. 2012. P.103-125

- Bubble sort. Disponível em https://pt.wikipedia.org/wiki/Bubble_sort. Acesso em abril de 2018.

- Heap sost. Disponível em https://pt.wikipedia.org/wiki/Heapsort. Acesso em abril de 2018.

- Merge sort . Disponível em https://pt.wikipedia.org/wiki/Merge_sort. Acesso em abril de 2018.

- Shell sort. Disponível em https://pt.wikipedia.org/wiki/Shell_sort. Acesso em abril de 2018.

- Selection sort. Disponível em https://pt.wikipedia.org/wiki/Selection_sort. Acesso em abril de 2018.

- Insertion sort. Disponível em https://pt.wikipedia.org/wiki/Insertion_sort. Acesso em abril de 2018.