Rafael Guedes Alves

Trabalho “Exercício Programa” de Análise de Algoritmos

Prof ° Dr ° Luiz Carlos da Silva Rozante

Centro de Matemática,

Computação e Cognição - UFABC

(VERSÃO NÃO FINALIZADA)

Santo André

2013

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

Hoje em dia a eficiência em algoritmos é um fator muito importante e que contribui para a melhoria de serviços para os usuários finais. Ao longo da evolução da computação existiu uma necessidade de aumentar a quantidade de dados que deveriam ser salvos de forma organizada. A fim de solucionar estes problemas de organização, a ciência da computação foi estimulada a estudar algoritmos de ordenação cada vez mais eficientes. E através destes estudos várias soluções foram encontradas.

CAPÍTULO 2 – DESCRIÇÃO DOS ALGORITMOS

2.1 – BubbleSort

A ordenação por bolha é a ordenação menos complexa para implementação, porém só é recomendada para pequenos valores de ordenação, uma vez que ela possui a complexidade da ordem de $O(n^2)$ aumentando muito o tempo de ordenação para grandes vetores. Seu funcionamento se baseia em verificar cada valor do vetor afim de verificar se ele é o menor e assim ordená-lo de forma correta.

2.2 – MergeSort

O *MergeSort* se basei no conceito de divisão de problemas, ou seja, a cada interação no momento de ordenação o algoritmo quebra o problema em 2, e assim por diante. Este algoritmo tem complexidade da ordem de $O(nlogn)$ ou seja, extremamente rápido, portanto é um dos algoritmos recomendados para grandes valores de n.

2.3 – InsertionSort

A ordenação por inserção acontece similarmente como a ordenação por bolha, onde o algoritmo verifica se um determinado valor é menor que todos os outros e o alinha mais a esquerda. Este algoritmo possui complexidade da ordem de $O(n^2)$.

2.4 – HeapSort

O *HeapSort* é outra variação de algoritmos que utilizam o conceito de divisão de trabalho para concluir um projeto maior. Para ordenar o heap cria uma estrutura de dados que ao ir inserindo também vai organizando, o tornando extremanente rápido. Ele possui uma complexidade da ordem de $O(nlogn)$.

2.5 – QuickSort

Este algoritmo é um dos mais rápidos e eficientes, e também se baseia no conceito de *divisão para conquistar*. A estratégia básica deste algoritmo é a ordenação por chave, montando um vetor com o tamanho menor. Este algoritmo possui complexidade da ordem de $O(nlogn)$.

2.6 – SelectionSort

O *SelectionSort* faz parte dos algoritmos de ordenação da ordem de $O(n^2)$ e é muito lento para grandes vetores, porém possui uma fácil implementação assim como o *BubbleSort.* Sua ordenação consiste em sempre passar o menor valor do vetor para a primeira posição e assim sucessivamente.

CAPÍTULO 3 – AMBIENTE DE EXECUÇÃO

Para este projeto foi desenvolvido uma aplicação que irá rodar em ambiente Android OS, afim de verificar a eficiência destes algoritmos em dispositivos móveis. Esta aplicação foi chamada de "AlgorithmAnalizer" e estará disponível para download no Google Play Store (https://play.google.com/store/apps/details?id=com.nmobile.ufabc.algorithmanalyzer).

A aplicação possui um menu inicial que pode ser visto na figura 1. Neste menu o usuário tem acesso a lista de algoritmos de ordenação mais utilizados.

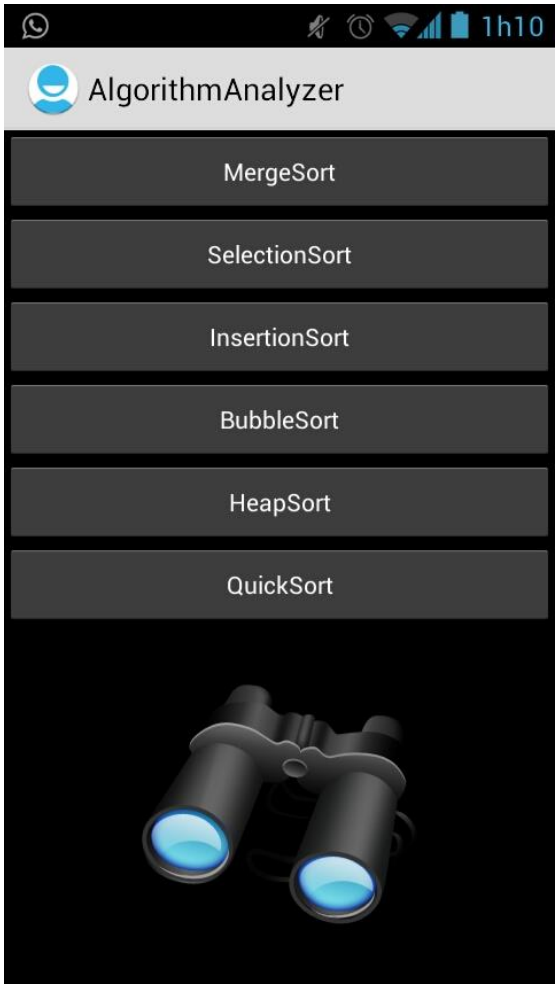
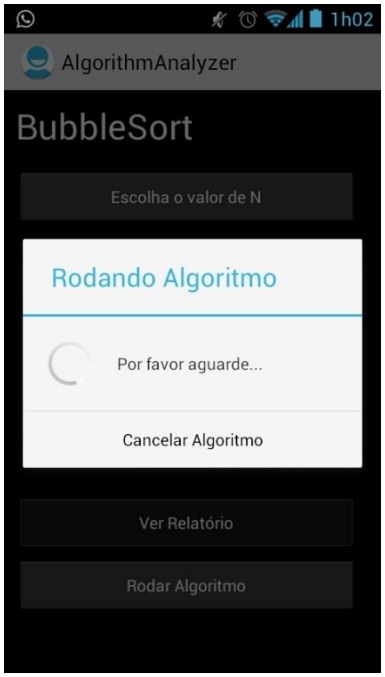
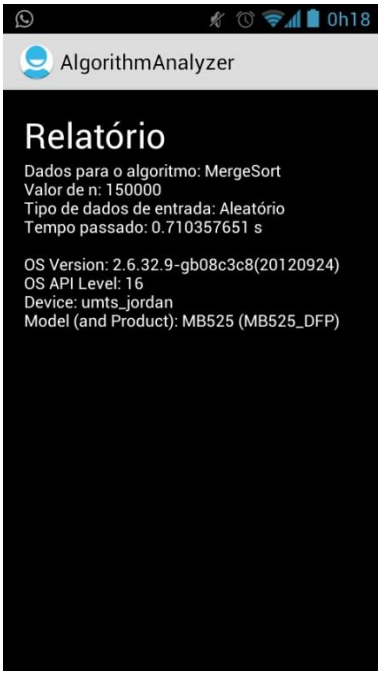


Figura 1 – Opções de Algoritmos presentes no aplicativo

Além da tela inicial o aplicativo ainda disponibiliza para o usuário a possibilidade de alterar configurações iniciais para o métodos de ordenação e há também a possibilidad de visualizar o relatório que contém informações mais detalhadas sobre cada algoritmo. A figura 3 apresenta outras telas da aplicação.


19h58
AlgorithmAnalyzer
BubbleSort
Escolha o valor de N
Dados Aleatórios
Dados Crescentes
Dados Decrescente
Ver Relatório
Rodar Algoritmo


1h02
AlgorithmAnalyzer
BubbleSort
Escolha o valor de N
Rodando Algoritmo
Por favor aguarde...
Cancelar Algoritmo
Ver Relatório
Rodar Algoritmo


0h18
AlgorithmAnalyzer
Relatório
Dados para o algoritmo: MergeSort
Valor de n: 150000
Tipo de dados de entrada: Aleatório
Tempo passado: 0.710357651 s
OS Version: 2.6.32.9-gb08c3c8(20120924)
OS API Level: 16
Device: umts_jordan
Model (and Product): MB525 (MB525_DFP)

CAPÍTULO 4 – ANÁLISE DE ALGORITMOS

Afim de analisar os algoritmos, foi proposto a utilização de vetores de inteiros que poderiam estar desorganizados, ou organizados de forma crescente ou decrescente. Além da organização inicial deste vetores, também foi possível a alteração do tamanho destes vetores e a quantidade de vezes que o algoritmo seria executado.

O dado que foi levado em consideração para mostrar a eficiência do algoritmo, foi o tempo necessário para completar a ordenação, ou seja, pegando o tempo inicial e subtraindo do tempo final, gerando um tempo total para realização da tarefas. Abaixo é apresentado os dados obtidos pelo aplicativo.

Dentro do aplicativo o usuário tem acesso ao relatório, que contém as informações referentes ao algoritmo que esta sendo testado, além de informações sobre o aparelho em que a aplicação está sendo rodada. A figura 2 apresenta o relatório para o algoritmo BubbleSort rodando com n = 10000 números.

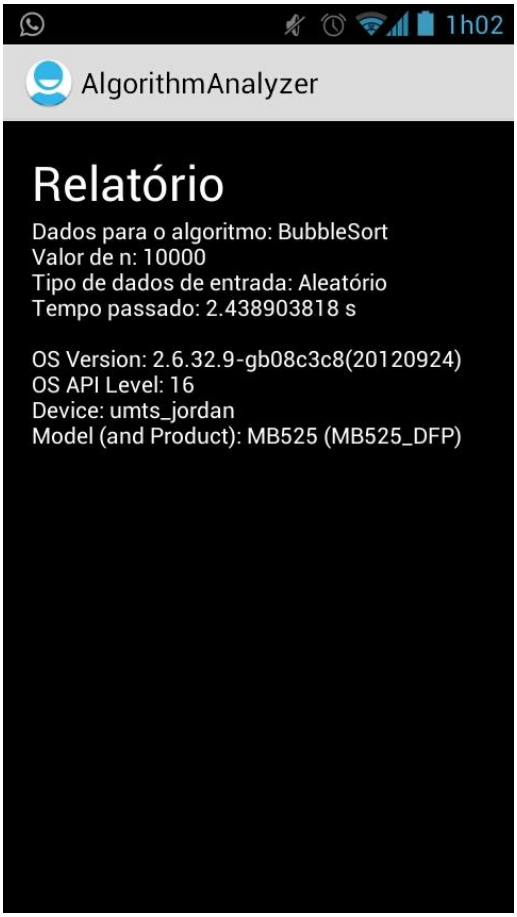


Figura 2 – Relatório gerado pelo aplicativo

CAPÍTULO 5 – DOCUMENTAÇÃO

O aplicativo possui 11 classes, sendo que 6 são responsáveis pela a ordenação dos vetores de inteiros que são passados como parâmetros. Sobre as classes temos uma breve descrição a seguir:

- BubbleSort: Classe responsável pela ordenação pelo método BubbleSort;
    - sort(int[] valores): Ordena o vetor valores pelo método BubbleSort.

- HeapSort: Classe responsável pela ordenação pelo método HeapSort;
    - sort(int[] valores): Ordena o vetor valores pelo método HeapSort.
    - sort(int left, int right): Método que é chamado recursivamente e que divide o Heap;
    - left(int i): Retorna um inteiro acrescido de 1 multiplicado duas vezes;
    - right(int i): Retorna um inteiro acrescido de 1 multiplicado das vezes;
    - swap(int x, int y): Alterna entre as tabelas Heap de ordenação;
    - buildMaxheap(): Monta a tabela Heap maior para aquela divisão;
    - maxHeapify(int i): Divide o Heap para problemas menores;

- Inicio: Apenas uma splashscreen;

- InsertionSort: Classe responsável pela ordenação pelo método InsertionSort;
    - insertionSort(int[] valores): Ordena o vetor valores pelo método InsertionSort.

- MenuActivity: Classe responsável por montar a tela de menu que contém as opções de algoritmos para ordenação;
    - onCreate(Bundle savedInstanceState): Cria a aplicação em ambiente android;
    - abreMerge(View v): Inicia a OpcoesOrdenador para o tipo de ordenação por MergeSort;
    - abreSelection(View v): Inicia a OpcoesOrdenador para o tipo de ordenação por SelectionSort;
    - abreInsertion(View v): Inicia a OpcoesOrdenador para o tipo de ordenação por InsertionSort;

- abreBubble(View v): Inicia a OpcoesOrdenador para o tipo de ordenação por BubbleSort;
- abreHeap(View v): Inicia a OpcoesOrdenador para o tipo de ordenação por HeapSort;
- abreQuick(View v): Inicia a OpcoesOrdenador para o tipo de ordenação por QuickSort;
- abreTodos(View v): Inicia a OpcoesOrdenador para realizar a ordenação para todos os algoritmos;

- MergeSort: Classe responsável pela ordenação pelo método MergeSort;
  - mergeSort(int[] valores): Inicia o método de ordenação MergeSort chamando o método recursivamente mergesort;
  - mergesort(int[] data, int first, int n): Método recursive que divide o vetor que foi passado inicialmente em 2;

- OpcoesOrdenador: Classe responsável pelas configurações de algoritmo, como por exemplo o valor de n e tipo de dado de entrada;
  - setValorN(View v): Método chamado pelo botão na Activity e responsável por iniciar o AlertDialog que solicita ao usuário o valor do tamanho do vetor;
  - rodaAlgoritmo(View v): Método responsável por verificar a consistência dos dados passados pelo usuário bem como iniciar o algoritmo solicitado pelo usuário;
  - createCancelProgressDialog(String title, String message, String buttonText, int tipo): Monta o dialog de espera enquanto o algoritmo é executado;
  - enviaRelatorio(View v): Envia e monta a tela que exibe o relátório final para o usuário;
  - gerarValoresAleatorios(int tamanho): Cria um vetor de inteiros com dados aleatórios criados pelo método Math.random();
  - gerarValoresCrescentes(int tamanho): Cria um vetor de inteiros ordenado de forma crescente;
  - gerarValoresDecrescente(int tamanho): Cria um vetor de inteiros ordenado de forma decrescente, através de uma ordenação invertida;

- mostrarValores(int[] valoresOrdenados): Cria uma *string* que é enviada para a tela de relatórios e posteriormente é apresentada ao usuário;
- rodarParaTodosAlgoritmos(int tVezes): Roda todos os algoritmos pela quantidade de vezes passadas pela variável tVezes.

- QuickSort: Classe responsável pela ordenação pelo método QuickSort;
  - sort(int[] valores): Ordena o vetor valores pelo método QuickSort.
  - sort(int left, int right): Método chamado recursivamente com o objetivo de dividir o vetor inicial em vetores menores;
  - findPivot(int left, int right): Encontra a posição de centro do vetor inicial;
  - partition(int left, int right): Divide em 2 partes o vetor passado inicialmente através do pivot gerado;
  - swap(int x, int y): Alterna entre os subvetores;

- SelectionSort: Classe responsável pela ordenação pelo método SelectionSort;
  - Métodos: sort(int[] valores): Ordena o vetor valores pelo método SelectionSort.

- VerRelatorio:
  - salvarRelatorio(View v): Salva o arquivo relatorio.txt na raiz do SDCARD do usuário;