Análise de Algoritmos

Aline Lima, Amanda Leticia, Eduardo Azevedo, Lucas Almeida

Curso de Computação – Universidade do Estado do Amazonas (UEA) – Escola Superior de Tecnologia  
69050-020 – Manaus – AM– Brasil

lima.alinee@gmail.com, a.amandachaves@outlook.com, eduardoareboucas@gmail.com,lucas.alsilva01@gmail.com

**Abstract.** This paper describes the BubbleSort ordering methods and QuickSort and sequential search methods, Sequential with sentry and Binary, an analysis of the algorithms mentioned above in order to verify the performance of such algorithms through its runtime is performed, which is used calculating the mean time, variance and standard deviation for better analysis.

**Resumo.** Este artigo descreve os métodos de ordenação BubbleSort e QuickSort e os métodos de busca Sequencial, Sequencial com sentinela e Binária, é realizada uma análise dos algoritmos citados acima visando verificar o desempenho de tais algoritmos através do seu tempo de execução, onde é utilizado o cálculo do tempo médio, variância e desvio padrão para melhor analise.

# 1. Introdução

Nota-se que a analise de algoritmos é fundamental pata medir o desempenho dos algoritmos por meio do tempo de execução e requisito de memoria, onde os dados obtidos por meio desses métodos comprovam sua eficiência no desempenho.

Utilizando a função time serão analisados os métodos de ordenação bubble sort e quicksort e o métodos de busca sequencial, busca binária e busca sequencial com sentinela.

**2. Objetivo**

Entender estruturas de dados e suas soluções de busca é fundamental no atual mundo computacional. Cada vez mais a otimização de programas é requisito obrigatório de todo programador quando usuário estão cada vez mais exigentes quanto a fluidez e rapidez das aplicações.

**3. Metodologia**

Para de alcançar o objetivo e aprender as estruturas de dados necessárias, foi necessário usar o sistema Linux (Ubuntu 15.1), numa maquina da marca Acer com processador Intel inside CORE i5, 6gb de memoria ram.

Os códigos foram feitos na maquina em questão através da IDE NetBeans 8.1 para a linguagem C, que é a linguagem utilizada na disciplina e foi utilizada nos códigos deste projeto. E com o sistema Linux podemos utilizar o terminal com alguns comandos específicos para a linguagem C como por exemplo o comando para ler e/ou gerar um arquivo ‘.txt’ que foi utilizado na maioria das questões, entre outros.

Após os códigos serem feitos e gerados os arquivos foi feito as analises solicitadas através do comando ‘time’ do terminal do Linux para medir o tempo médio de execução dos algoritmos e assim gerar as tabelas e gráficos através do software LibreOffice.

# 4. Análise de Algoritmos de Ordenação e Busca

São algoritmos que colocam elementos de uma sequência em uma certa ordem, normalmente numéricas sendo crescente ou decrescente. No trabalho em questão foi utilizado os algoritmos bubblesort e quicksort.

Além dos algoritmos de ordenação utilizamos em algumas questões os algoritmos de busca. Que em termos gerais é um algoritmo que pega um problema como entrada e retorna a possível solução do problema, geralmente após resolver um número possível de soluções. Os algoritmos utilizados foram a busca sequencial, binária e sequencial com sentinela que tem como objetivo buscar um determinado número num conjunto de números.

## 4.1 BubbleSort

É um dos mais simples algoritmos de ordenação, a ideia do algoritmo é percorrer um vetor diversas vezes e a cada passagem pelo vetor encontrar o maior elemento da sequencia e “flutuar” ele para o topo do vetor. Esse tipo de algoritmo não é recomendado para programas que usem um numero elevado de dados ou que precisem de velocidade na operação pois sua complexidade é de ordem quadrática, ou seja, o tempo é O(nˆ2).

Exemplo:

1 int bubble (int tamanho\_vetor)

2 {

3 int i, j, aux;

4

5 for(i=0 ; i < tamanho\_vetor ; i++)

6 {

7 for(j=i+1; j < tamanho\_vetor ; j++)

8 {

9 if(vetor[i] > vetor[j])

10 {

11 aux=vetor[i];

12 vetor[i]=vetor[j];

13 vetor[j]=aux;

14 }

15 }

16 }

17 return 0;

18 }

## 4.2 QuickSort

É um algoritmo de ordenação rápido e eficiente comparado a outros como o BubbleSort por exemplo. Ele implementa a estratégia de divisão e conquista que consiste em rearranjar as chaves de modo q as chaves menores precedem as chaves maiores, em seguida ele ordena as duas sub listas de chaves menores em seguida maiores por meio da recursividade até que a lista de números esteja ordenada. Sua complexidade de tempo é logarítmica no melhor caso, ou seja, tempo é O(n log2 n) e no pior caso é quadrático, ou seja, O(nˆ2).

Exemplo:

1 void troca (int \*n1, int \*n2)

2 {

3 int aux;

5 aux = \*n1;

6 \*n1 = \*n2;

7 \*n2 = aux;

8 }

10 int partição (int vetor[], int p, int r)

11 {

12 int i, j, x, temp;

14 x = vetor[p];

16 i = p - 1;

17 j = r + 1;

19 while (1)

20 {

21 do

22 {

23 i++;

24 } while (vetor[i] < x);

26 do

27 {

28 j--;

29 } while (vetor[j] > x);

31 if (i >= j)

32 {

33 return j;

34 }

36 troca (&vetor[i], &vetor[j]);

37 }

38 }

40 void quicksort (int vetor[], int p, int r)

41 {

42 int q;

43 if (p < r)

44 {

45 q = particao (vetor, p, r);

46 quicksort (vetor, p, q);

47 quicksort(vetor, q + 1, r);

48 }

49 }

**4.3 Busca Sequencial e Sequencial com Sentinela**

É um método de busca de forma sequencial, ou seja, uma busca em um conjunto de elementos de um por um, de modo crescente e proporcional, no pior caso de busca o elemento a ser encontrado estará na ultima posição do conjunto levando um tempo um pouco extreme para encontrar o elemento em um programa que use um numero elevado de dados, sua complexidade é linear, ou seja, O(n).

Exemplo de algoritmo de busca sequencial:

1 int busca\_seq (int x)

2 {

3 int i;

4

5 for (i=0 ; i<10 ; i++)

6 {

7 if (vetor[i] == x)

8 return 1;

9 }

10 return 0;

11 }

A busca sequencial com sentinela adiciona o valor buscado ao final do vetor garantindo que o valor sempre será encontrado e então o programa ira percorrer o laço com a condição de parada quando o valor que esta passando no laço for igual ao valor pesquisado. No final basta fazer o teste para garantir que o índice do laço é igual ao final do vetor, ou seja, não encontrou o valor, ou se é diferente, nesse caso significa que encontrou o valor buscado.

Exemplo de algoritmo de busca sequencial com sentinela:

1 int busca\_sentinela(int x, int n, int v[])

2 {

3 int i = 0;

4

5 v[n] = x; // sentinela

6

7 while (v[i] != x)

8 i++;

9

10 return (i == n) ? -1 : i;

11 }

## 4.4 Busca Binária

É um algoritmo de busca que assim como o QuickSort segue a estratégia de divisão e conquista, ela parte do pressuposto de que o conjunto de números esteja ordenado pois somente com o conjunto ordenado se obtém sucesso nesse método de busca, assim ele realiza sucessivas divisões de espaço de busca comparando o elemento buscado com o elemento do meio do vetor. Se o elemento do meio for o buscado a busca finaliza com sucesso, se não, se o elemento do meio for menor que o elemento buscado então a busca continua da metade posterior do conjunto, e finalmente se o elemento do meio for maior que o elemento procurado a busca continua na metade anterior do conjunto. Como é utilizada a estratégia de divisão sua complexidade é logarítima, ou seja, O(n log2 n).

Exemplo:

1 int BuscaBinaria(int vector[], int inicio, int end, int numBusc)

2 {

3 if (end >= inicio)

4 {

5 int meio = inicio + (end - inicio)/2;

6

7 if (vector[meio] == numBusc)

8 {

9 return meio;

10 }

11

12 if (vector[meio] > numBusc)

13 {

14 return BuscaBinaria(vector, inicio, meio-1,numBusc);

15 }

16

17 return BuscaBinaria(vector, meio+1, end, numBusc);

18 }

19

20 return -1;

21 }

**5. Análise**

Para realizar este artigo foram criados diferentes programas com o objetivo de analisar comparações entre os algoritmos citados anteriormente. Após a criação dos algoritmos requisitados, pegou-se um número aleatório para realizar as devidas comparações. Em seguida foram feitos testes com números aleatórios, crescentes e decrescentes, de 10000 a 1000000. Cada uma das possibilidades solicitadas foram realizadas e relatadas em tabelas e gráficos, como veremos a seguir. As análises do algoritmo de bubblesort para 1000000 de valores, não puderam ser realizadas devido a falta de equipamento para a realização das mesmas.

# 6. Tabelas e Gráficos

**Tabela 1. Análise do algoritmo BubbleSort com os diferentes tipos de vetores (Dez Mil)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 10000 C | 10000 D | 10000 A |
| Teste 1 | 0,171 | 0,235 | 0,348 |
| Teste 2 | 0,166 | 0,2 | 0,385 |
| Teste 3 | 0,217 | 0,22 | 0,346 |
| Teste 4 | 0,192 | 0,231 | 0,381 |
| Teste 5 | 0,194 | 0,256 | 0,358 |
| Teste 6 | 0,187 | 0,213 | 0,404 |
| Teste 7 | 0,211 | 0,217 | 0,379 |
| Teste 8 | 0,187 | 0,249 | 0,364 |
| Teste 9 | 0,175 | 0,226 | 0,336 |
| Teste 10 | 0,175 | 0,212 | 0,4 |
| Média | 0,1875 | 0,2259 | 0,3701 |
| Variância | 0,000281389 | 0,000299211 | 0,0005421 |
| Desvio Padrão | 0,01677465 | 0,01729772 | 0,023283041 |

**Tabela 2. Análise do algoritmo BubbleSort com os diferentes tipos de vetores (Cem Mil)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 100000 C | 100000 D | 100000 A |
| Teste 1 | 15,625 | 17,974 | 35,306 |
| Teste 2 | 15,886 | 17,86 | 34,497 |
| Teste 3 | 15,684 | 17,922 | 34,682 |
| Teste 4 | 15,602 | 17,908 | 34,868 |
| Teste 5 | 15,657 | 17,863 | 34,144 |
| Teste 6 | 15,644 | 17,902 | 33,773 |
| Teste 7 | 15,619 | 17,938 | 34,4 |
| Teste 8 | 15,631 | 18,321 | 37,369 |
| Teste 9 | 15,635 | 17,845 | 34,109 |
| Teste 10 | 15,589 | 18,17 | 34,439 |
| Média | 15,6572 | 17,9703 | 34,7587 |
| Variância | 0,007179511 | 0,023798456 | 1,022440456 |
| Desvio Padrão | 0,084731996 | 0,154267481 | 1,011157978 |

**Gráfico 1. Análise do tempo médio dos valores crescente, decrescente e aleatório das tabelas 1 e 2**

**Tabela 3. Análise do algoritmo QuickSort com os diferentes tipos de vetores (Dez Mil)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 10000 C | 10000 D | 10000 A |
| Teste 1 | 0,462 | 0,33 | 0,001 |
| Teste 2 | 0,458 | 0,334 | 0,002 |
| Teste 3 | 0,452 | 0,324 | 0,001 |
| Teste 4 | 0,468 | 0,324 | 0,002 |
| Teste 5 | 0,536 | 0,325 | 0,001 |
| Teste 6 | 0,473 | 0,323 | 0,002 |
| Teste 7 | 0,463 | 0,32 | 0,001 |
| Teste 8 | 0,446 | 0.327000 | 0,003 |
| Teste 9 | 0,443 | 0,325 | 0,001 |
| Teste 10 | 0,536 | 0,325 | 0,001 |
| Média | 0,4737 | 0,325625 | 0,0015 |
| Variância | 0,001163789 | 0,011798028 | 0,0000005 |
| Desvio Padrão | 0,03411435 | 0,108618727 | 0,000707107 |

**Tabela 4. Análise do algoritmo QuickSort com os diferentes tipos de vetores (Cem Mil)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 100000 C | 100000 D | 100000 A |
| Teste 1 | 40,3170 | 35,1420 | 0,02 |
| Teste 2 | 41,1350 | 35,2340 | 0,03 |
| Teste 3 | 39,1930 | 35,1100 | 0,025 |
| Teste 4 | 38,9250 | 34,6840 | 0,04 |
| Teste 5 | 39,3610 | 34,9120 | 0,032 |
| Teste 6 | 42,4300 | 34,5650 | 0,037 |
| Teste 7 | 40,6270 | 35,4060 | 0,027 |
| Teste 8 | 40,8590 | 34,8810 | 0,029 |
| Teste 9 | 40,0530 | 35,5200 | 0,03 |
| Teste 10 | 40,6270 | 35,4060 | 0,027 |
| Média | 40,3527 | 35,086 | 0,0297 |
| Variância | 1,087976011 | 0,102679778 | 3,29E-05 |
| Desvio Padrão | 1,043060886 | 0,320436855 | 0,005735852 |

**Tabela 5. Análise do algoritmo QuickSort com os diferentes tipos de vetores (Um Milhão)**

|  |  |  |
| --- | --- | --- |
|  | 100000 C | 100000 D |
| Teste 1 | 410,5300 | 371,4100 |
| Teste 2 | 422,5090 | 372,5290 |
| Teste 3 | 436,4070 | 394,3210 |
| Teste 4 | 414,5000 | 381,3000 |
| Teste 5 | 414,6010 | 378,2010 |
| Teste 6 | 392,2050 | 374,1550 |
| Teste 7 | 393,0030 | 392,4330 |
| Teste 8 | 411,2350 | 394,2570 |
| Teste 9 | 420,0170 | 395,0000 |
| Teste 10 | 422,5090 | 372,5290 |
| Média | 413,7516 | 382,6135 |
| Variância | 179,8511594 | 104,9043294 |
| Desvio Padrão | 13,41085976 | 10,24228145 |

**Gráfico 2. Análise do tempo médio dos valores crescente, decrescente e aleatório.**

# Referências

Análise de Algoritmos Quick. Disponível em <<http:www.ime.usp.br>> Acesso em: 15 de setembro de 2016.

Entendendo o Algoritmo Bubble Sort em Java. Disponível em <<http://www.devmedia.com.br/>> Acesso em: 15 de setembro de 2016.