INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA

CAMPUS MONTEIRO

Análise e Desenvolvimento de sistemas

nycolas ramon alves da silva

Projeto final – análise de algoritmos

MONTEIRO- PB

2021

1. Introdução

O presente relatório tem como objetivo relatar experimentos práticos de algoritmos de ordenação em diferentes cenários, como pré-requisito para aprovação na disciplina de análise de algoritmos. Os algoritmos escolhidos para os testes foram os algoritmos Bubble Sort, Selection Sort, Insertion Sort, Merge Sort e Quick Sort.

Para comparar os algoritmos, foram utilizados três métricas: o número de comparações de chaves, o número de movimentações de registros e o tempo total gasto para ordenação (tempo de processamento e não o tempo de relógio) em milissegundos. Para realizar essas comparações, cada algoritmo calcula seus 3 valores respectivos a cada execução e os retorna em um array de 3 posições, onde o índice 0 corresponde ao tempo de execução, o índice 1 à quantidade de comparações de chaves feitas e o índice 2 à quantidade de movimentações de registro realizadas.

As implementações dos algoritmos foram feitas na linguagem de programação Java e os testes foram executados em um notebook com Windows 10, 8GB de memória R.A.M. e processador Intel i5.

Os elementos ordenados pelos algoritmos escolhidos foram números dispostos em sequencias ordenadas, inversamente ordenadas, quase ordenadas e aleatórias. Para cada tipo de sequência, foram usados arrays de tamanhos que variam em 10, 100, 1000, 10000, 100000 e 1000000 de números.

Para cada tamanho de array, foram feitos 30 testes de ordenação, onde o primeiro consistiu em usar uma sequência ordenada, o segundo uma sequência inversamente ordenada, os próximos 14 testes usaram as sequencias quase ordenadas e os últimos 14, sequencias aleatórias.

O presente relatório está disposto em partes; sendo esta introdução a primeira delas, a segunda parte contém os detalhamentos dos testes, dividida em sub tópicos para cada algoritmo (cada tópico contendo o código do algoritmo ordenador, os dados dos resultados em tabelas e gráficos para visualização do desempenho do algoritmo) e a terceira e última parte conta com análises mais profundas combinando os resultados dispostos na segunda parte.

1. Análise dos Algoritmos
   1. Bubble Sort
      1. Código fonte

**public** **long**[] **sort**(Long[] array) {

**long** **comparacoesDeChaves** = 0;

**long** **movimentacoesDeRegistros** = 0;

Instant **start** = Instant.*now*();

// Se não houveram mudanças, o array já está ordenado

**boolean** **hasChanged** = **true**;

**while** (hasChanged) {

hasChanged = **false**;

// Contador para que o último elemento não seja analisado

**int** **j** = 0;

**for** (**int** **i** = 0; i < array.length - 1 - j; i++) {

// TODo Comparação de chaves

comparacoesDeChaves++;

**if**(array[i] > array[i+1]) {

// Troca do valor com seu sucessor

**long** **aux** = array[i];

// TODo Movimentação de registro

array[i] = array[i+1];

// TODo Movimentação de registro

array[i+1] = aux;

// TODo Movimentação de registro

movimentacoesDeRegistros += 3;

hasChanged = **true**;

}

}

j++;

}

Instant **end** = Instant.*now*();

Duration **duration** = Duration.*between*(start, end);

**long** **durationInMillisseconds** = duration.toMillis();

**long**[] **resultados** = {

durationInMillisseconds,

comparacoesDeChaves,

movimentacoesDeRegistros

};

**return** resultados;

}

* + 1. Resultados tabelados
       1. Resultados para sequencia ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 | 0 | 9 | 0 |
| 100 | 0 | 99 | 0 |
| 1000 | 0 | 999 | 0 |
| 10000 | 1 | 9999 | 0 |
| 100000 | 1 | 99999 | 0 |
| 1000000 | 14 | 999999 | 0 |

* + - 1. Resultados para sequencia inversamente ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 | 1 | 90 | 135 |
| 100 | 2 | 9900 | 14850 |
| 1000 | 22 | 999000 | 1498500 |
| 10000 | 1149 | 99990000 | 149985000 |
| 100000 | 81846 | 9999900000 | 14999850000 |
| 1000000 | 8320822 | 999999000000 | 1499998500000 |

* + - 1. Resultados para sequencia quase ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 | 0 | 35 | 16 |
| 100 | 1 | 4384 | 1810 |
| 1000 | 3 | 476665 | 186135 |
| 10000 | 427 | 49092947 | 18800822 |
| 100000 | 41958 | 4963807504 | 1874626713 |
| 1000000 | 4.770.026 | 499266929304 | 187501216324 |

* + - 1. Resultados para sequência aleatória

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 | 0 | 65 | 65 |
| 100 | 0 | 8740 | 7251 |
| 1000 | 8 | 959254 | 749809 |
| 10000 | 1183 | 98894395 | 74960706 |
| 100000 | 115059 | 9967914605 | 7499302192 |
| 1000000 | 11260321 | 998347334985 | 749990953181 |

* + 1. Gráficos demonstrativos
  1. Selection Sort
     1. Código fonte

**public** **long**[] **sort**(Long[] array) {

**long** **comparacoesDeChaves** = 0;

**long** **movimentacoesDeRegistros** = 0;

Instant **start** = Instant.*now*();

**for** (**int** **i** = 0; i < array.length; i++) {

// Identificando o índice de menor elemento de um vetor

**int** **indexSmaller** = i; // assume-se que o menor é o i-ésimo

// A iteração em busca do menor valor deve acontecer após

// a posição do i, pois este trecho já está ordenado

**for** (**int** **j** = i+1; j < array.length; j++) {

comparacoesDeChaves++;

// TODo Comparação de chaves

**if**(array[j] < array[indexSmaller]) {

indexSmaller = j;

}

}

// troca do i-ésimo valor com o menor valor encontrado

Long **aux** = array[i];

// TODo Movimentação de registro

array[i] = array[indexSmaller];

// TODo Movimentação de registro

array[indexSmaller] = aux;

// TODo Movimentação de registro

movimentacoesDeRegistros += 3;

}

Instant **end** = Instant.*now*();

Duration **duration** = Duration.*between*(start, end);

**long** **durationInMillisseconds** = duration.toMillis();

**long**[] **resultados** = {

durationInMillisseconds,

comparacoesDeChaves,

movimentacoesDeRegistros

};

**return** resultados;

}

* + 1. Resultados tabelados
       1. Resultados para sequencia ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequencia inversamente ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequencia quase ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequência aleatória

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + 1. Gráficos demonstrativos
  1. Insertion Sort
     1. Código fonte

**public** **long**[] **sort**(Long[] array) {

**long** **comparacoesDeChaves** = 0;

**long** **movimentacoesDeRegistros** = 0;

Instant **start** = Instant.*now*();

**for**(**int** **i** = 0; i < array.length - 1; i++) {

**int** **j** = i + 1;

// TODo Movimentação de registro

**long** **number** = array[j]; // Número a ser posicionado

movimentacoesDeRegistros++;

// Colocando um número de sequência desordenada na sequência ordenada

**while**(j > 0 && number < array[j-1]) {

// TODo Comparação de chaves

comparacoesDeChaves++;

array[j] = array[j-1];

// TODo Movimentação de registro

movimentacoesDeRegistros++;

j--;

}

array[j] = number;

// TODo Movimentação de registro

movimentacoesDeRegistros++;

}

Instant **end** = Instant.*now*();

Duration **duration** = Duration.*between*(start, end);

**long** **durationInMillisseconds** = duration.toMillis();

**long**[] **resultados** = {

durationInMillisseconds,

comparacoesDeChaves,

movimentacoesDeRegistros

};

**return** resultados;

}

* + 1. Resultados tabelados
       1. Resultados para sequencia ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequencia inversamente ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequencia quase ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequência aleatória

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + 1. Gráficos demonstrativos
  1. Merge Sort
     1. Código fonte

**public** **class** MergeSort **extends** AlgoritmoOrdenacao {

**private** **long** comparacoesDeChaves = 0;

**private** **long** movimentacoesDeRegistros = 0;

@Override

**public** **long**[] **sort**(Long[] array) {

Instant **start** = Instant.*now*();

sort(array, 0, array.length-1);

Instant **end** = Instant.*now*();

Duration **duration** = Duration.*between*(start, end);

**long** **durationInMillisseconds** = duration.toMillis();

**long**[] **resultados** = {

durationInMillisseconds,

comparacoesDeChaves,

movimentacoesDeRegistros

};

**return** resultados;

}

**private** **void** **sort**(Long[] array, **int** inicio, **int** fim) {

**if**(inicio < fim) {

**int** **meio** = (inicio + fim) / 2;

sort(array, inicio, meio);

sort(array, meio+1, fim);

merge(array, inicio, meio, fim);

}

}

**private** **void** **merge**(Long array[], **int** esquerda, **int** meio, **int** direita) {

// Obtendo o tamanho dos subarrays a serem intercalados

**int** **n1** = meio - esquerda + 1;

**int** **n2** = direita - meio;

// Criação dos arrays temporários

Long **arrayEsquerdo**[] = **new** Long[n1];

Long **arrayDireito**[] = **new** Long[n2];

// Respectivos valores copiados para os arrays temporários

**for** (**int** **i** = 0; i < n1; ++i) {

arrayEsquerdo[i] = array[esquerda + i]; // TODo Movimentação de registro

**this**.movimentacoesDeRegistros++;

}

**for** (**int** **j** = 0; j < n2; ++j) {

arrayDireito[j] = array[meio + 1 + j]; // TODo Movimentação de registro

**this**.movimentacoesDeRegistros++;

}

/\* Intercalando os arrays temporários \*/

// Indices iniciais do primeiro e segundo sub-arrays

**int** **i** = 0, **j** = 0;

// Indicice inicial do array intercalado

**int** **k** = esquerda;

**while** (i < n1 && j < n2) {

**this**.comparacoesDeChaves++;

**if** (arrayEsquerdo[i] <= arrayDireito[j]) { // TODo Comparação de chaves

array[k] = arrayEsquerdo[i]; // TODo Movimentação de registro

i++;

} **else** {

array[k] = arrayDireito[j]; // TODo Movimentação de registro

j++;

}

**this**.movimentacoesDeRegistros++;

// Entrando no if ou no else, será feita uma movimentação de registro

k++;

}

// Cópia dos elementos restantes de arrayEsquerdo[] se existirem

**while** (i < n1) {

array[k] = arrayEsquerdo[i]; // TODo Movimentação de registro

**this**.movimentacoesDeRegistros++;

i++;

k++;

}

// Cópia dos elementos restantes de arrayDireito[] se existirem

**while** (j < n2) {

array[k] = arrayDireito[j]; // TODo Movimentação de registro

**this**.movimentacoesDeRegistros++;

j++;

k++;

}

}

}

* + 1. Resultados tabelados
       1. Resultados para sequencia ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequencia inversamente ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequencia quase ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequência aleatória

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + 1. Gráficos demonstrativos
  1. Quick Sort
     1. Código fonte

**public** **class** QuickSort **extends** AlgoritmoOrdenacao {

**private** **long** comparacoesDeChaves = 0;

**private** **long** movimentacoesDeRegistros = 0;

@Override

**public** **long**[] **sort**(Long[] array) {

Instant **start** = Instant.*now*();

sort(array, 0, array.length-1);

Instant **end** = Instant.*now*();

Duration **duration** = Duration.*between*(start, end);

**long** **durationInMillisseconds** = duration.toMillis();

**long**[] **resultados** = {

durationInMillisseconds,

comparacoesDeChaves,

movimentacoesDeRegistros

};

**return** resultados;

}

**private** **void** **sort**(Long[] array, **int** inicio, **int** fim) {

**int** **esquerda** = inicio, **direita** = fim;

// Pivô calculado aleatoriamente

Random **random** = **new** Random();

**long** **pivot** = array[esquerda + random.nextInt(direita - esquerda + 1)];

// TODo Movimentação de registro

**this**.movimentacoesDeRegistros++;

**while**(esquerda <= direita) {

**while**(esquerda <= fim && array[esquerda] < pivot) {

// TODo Comparação de chave

**this**.comparacoesDeChaves++;

esquerda++;

}

**while**(direita >= inicio && array[direita] > pivot) {

// TODo Comparação de chaves

**this**.comparacoesDeChaves++;

direita--;

}

**if**(esquerda <= direita) {

**long** **aux** = array[esquerda]; // TODo Movimentação de registro

array[esquerda] = array[direita]; // TODo Movimentação de registro

array[direita] = aux; // TODo Movimentação de registro

**this**.movimentacoesDeRegistros += 3;

esquerda++;

direita--;

}

}

**if**(esquerda < fim)

sort(array, esquerda, fim);

**if**(direita > inicio)

sort(array, inicio, direita);

}

}

* + 1. Resultados tabelados
       1. Resultados para sequencia ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequencia inversamente ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequencia quase ordenada

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + - 1. Resultados para sequência aleatória

|  |  |  |  |
| --- | --- | --- | --- |
| Tamanho das Sequencias | Tempo de Execução | Comparação de Chaves | Movimentação de Registros |
| 10 |  |  |  |
| 100 |  |  |  |
| 1000 |  |  |  |
| 10000 |  |  |  |
| 100000 |  |  |  |
| 1000000 |  |  |  |

* + 1. Gráficos demonstrativos

1. Conclusões Finais