Avaliação empírica de algoritmos de ordenação

Robson Novato - 20.1.4018
Pedro Lucas - 20.1.4003
Pedro Henrique - 20.1.4005

Assuntos tratados

Apresentação dos métodos	Aqui será apresentado os métodos de ordenação e como funcionam
Análise assintótica	Análise assintótica dos métodos escolhidos, em big-o
Ambiente de teste	Quais especificações da máquina que foi realizado os testes e geração dos vetores
Resultados obtidos	Resultados obtidos com os testes realizados
Análise estatística	Análise do resultado obtido com o teste estatístico t com 95% de confiança
Conclusões	Conclusões com a análise estatística

Participantes:
Pedro Henrique | Robson | Pedro Lucas

Disciplina:
Projeto e Análise de Algoritmos | BCC241

Métodos de ordenação

01

Merge Sort

O merge sort é um exemplo de algoritmo de ordenação por comparação do tipo dividir-para-conquistar. Sua ideia básica consiste em Dividir (o problema em vários subproblemas e resolver esses subproblemas através da recursividade) e Conquistar (após todos os subproblemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos subproblemas).

02

Selection Sort

A ordenação por seleção é um algoritmo de ordenação baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição, e assim é feito sucessivamente com os elementos restantes.

03

Radix Sort

O Radix sort é um algoritmo de ordenação rápido e estável que pode ser usado para ordenar itens que estão identificados por chaves únicas. Cada chave é uma cadeia de caracteres ou número, e o radix sort ordena estas chaves em qualquer ordem relacionada com a lexicografia. Radix sort é um algoritmo de ordenação que ordena inteiros processando dígitos individuais.

Métodos de ordenação

01

Merge Sort

```c
void mergeSort(int *v, int l, int r) {
  int m;
  if (l < r) {
    m = (l+r)/2;
    mergeSort(v, l, m);
    mergeSort(v, m+1, r);
    merge(v, l, m, r);
  }
}
```

```c
void merge(int *v, int l, int m, int r) {
  int sizeL = (m-l+1), sizeR = (r-m), i, j;
  int *vetorL = NULL, *vetorR = NULL;
  vetorL = alocaVetor(vetorL, sizeL);
  vetorR = alocaVetor(vetorR, sizeR);

  for (i = 0; i < sizeL; i++) {
    vetorL[i] = v[i+l];
  }
  for (j = 0; j < sizeR; j++) {
    vetorR[j] = v[m+j+1];
  }

  i = 0; j = 0;
  for (int k = l; k <= r; k++) {
    if (i == sizeL) {
      v[k] = vetorR[j++];
    } else if (j == sizeR) {
      v[k] = vetorL[i++];
    } else if (vetorL[i] <= vetorR[j]) {
      v[k] = vetorL[i++];
    } else {
      v[k] = vetorR[j++];
    }
  }

  vetorL = desalocaVetor(vetorL);
  vetorR = desalocaVetor(vetorR);
}
```

Métodos de ordenação

02

Selection Sort

```c
void selectionSort (int *vetor, int n) {
    int aux;
    int menorAtual;
    for(int i = 0; i < n; i++) {
        menorAtual = i;
        for (int j = i + 1; j < n; j++) {
            if (vetor[j] < vetor[menorAtual])
                menorAtual = j;
        }
        if(menorAtual != i) {
            aux = vetor[i];
            vetor[i] = vetor[menorAtual];
            vetor[menorAtual] = aux;
        }
    }
}
```

Métodos de ordenação

03

Radix Sort

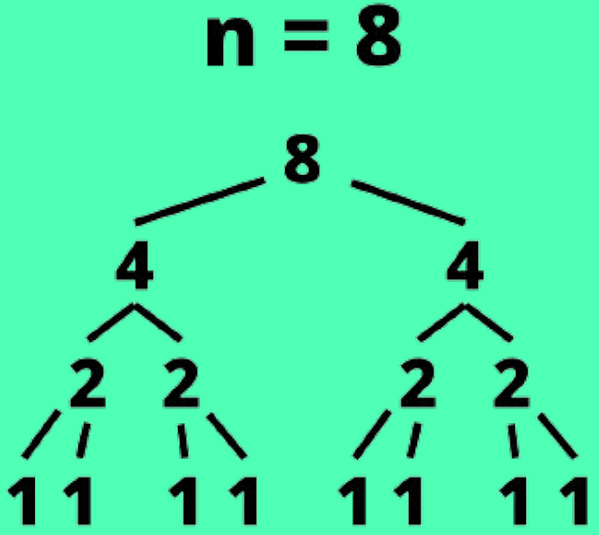
```c
void radixSort(int array[], int size) {
  int max = getMax(array, size);

  for (int place = 1; max / place > 0; place *= 10)
    countingSort(array, size, place);
}
```

```c
int getMax(int array[], int n) {
  int max = array[0];
  for (int i = 1; i < n; i++)
    if (array[i] > max)
      max = array[i];
  return max;
}

void countingSort(int array[], int size, int place) {
  int output[size + 1];
  int max = (array[0] / place) % 10;

  for (int i = 1; i < size; i++) {
    if (((array[i] / place) % 10) > max)
      max = array[i];
  }
  int count[max + 1];

  for (int i = 0; i < max; ++i)
    count[i] = 0;

  for (int i = 0; i < size; i++)
    count[(array[i] / place) % 10]++;

  for (int i = 1; i < 10; i++)
    count[i] += count[i - 1];

  for (int i = size - 1; i >= 0; i--) {
    output[count[(array[i] / place) % 10] - 1] = array[i];
    count[(array[i] / place) % 10]--;
  }

  for (int i = 0; i < size; i++)
    array[i] = output[i];
}
```

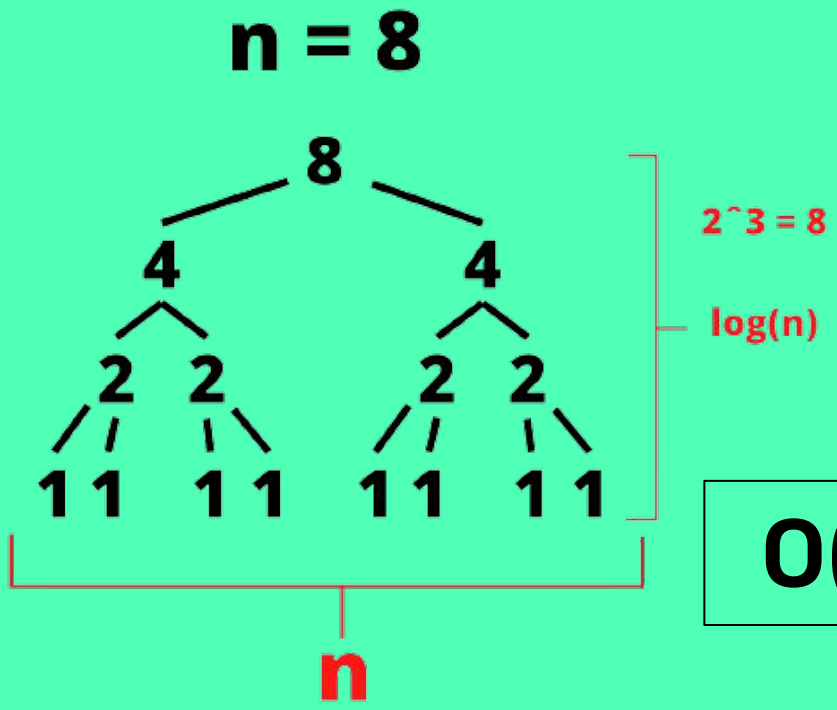
Análise assintótica

Na análise assintótica dos métodos desejamos capturar a tendência real do desempenho de um algoritmo, quando o tamanho da instância de entrada cresce.

01 Merge Sort



01 Merge Sort



O(n * log(n))

02 Selection Sort

```
void selectionSort (int *vetor, int n) {
  int aux;
  int menorAtual;
  for(int i = 0; i < n; i++) {
  menorAtual = i;
  for (int j = i + 1; j < n; j++) {
   if (vetor[j] < vetor[menorAtual])
    menorAtual = j;
   }
   if(menorAtual != i) {
     aux = vetor[i];
     vetor[i] = vetor[menorAtual];
     vetor[menorAtual] = aux;
    }
  }
}
```

O(n^2)

03 Radix Sort

- Ele cria b buckets. O custo disso é constante em cada caso.
- Ele itera sobre todos os n elementos para classificá-los nos buckets. O custo de calcular um número de bucket e inserir um elemento em um bucket é constante.
- Ele itera em b buckets e copia um total de n elementos deles. O custo para cada uma dessas etapas é novamente constante.

O(n+k)

Ambiente de teste

Processador — Intel Core i5-4690 4x3,5 GHz

Ram — 8GB RAM DDR3 1666MHz

S.O — Ubuntu 20.04 LTS (WSL), GCC 9.4

```
srand(time(NULL));
clock_t start, end;
double time[3];
int n = atoi(argv[1]);
int **arr = criarMatriz(3, n);

for (int i = 0 ; i < n ; i++)
    arr[0][i] = arr[1][i] = arr[2][i] = rand () % n;
```

Resultados obtidos

Devem ser geradas 20 instâncias, cujos valores (chaves) devem ser preenchidos
aleatoriamente com valores entre 1 e n.
Para uma mesma instância (vetor), deve-se executar cada algoritmo de ordenação e medir o tempo de execução.
Comece com n=100 e vá aumentando-o em potência de 10. Ou seja, use n=100, 1.000, 10.000, 100.000, 1.000.000, ...

100 instâncias

Metodo	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
MergeSort	0,000016	0,000015	0,000015	0,000015	0,000014	0,000015	0,000015	0,000015	0,000015	0,000015
RadixSort	0,000009	0,000007	0,000006	0,000006	0,000006	0,000006	0,000007	0,000006	0,000007	0,000006
SelectionSort	0,000016	0,000017	0,000016	0,000017	0,000015	0,000017	0,000016	0,000016	0,000016	0,000016
Metodo	**T11**	**T12**	**T13**	**T14**	**T15**	**T16**	**T17**	**T18**	**T19**	**T20**
MergeSort	0,000015	0,000015	0,000013	0,000015	0,000015	0,000014	0,000016	0,000015	0,000017	0,000015
RadixSort	0,000009	0,000006	0,000006	0,000007	0,000010	0,000009	0,000060	0,000007	0,000007	0,000006
SelectionSort	0,000016	0,000017	0,000015	0,000017	0,000016	0,000016	0,000015	0,000017	0,000025	0,000016

1.000 instâncias

Metodo	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
MergeSort	0,000150	0,000152	0,000158	0,000152	0,000153	0,000153	0,000152	0,000152	0,000144	0,000136
RadixSort	0,000099	0,000082	0,000139	0,000085	0,000081	0,000096	0,000149	0,000135	0,000080	0,000073
SelectionSort	0,001265	0,001260	0,001351	0,001280	0,001286	0,001265	0,001267	0,001452	0,001236	0,001350
Metodo	**T11**	**T12**	**T13**	**T14**	**T15**	**T16**	**T17**	**T18**	**T19**	**T20**
MergeSort	0,000140	0,000153	0,000152	0,000155	0,000144	0,000151	0,000170	0,000145	0,000144	0,000142
RadixSort	0,000077	0,000148	0,000082	0,000140	0,000119	0,000084	0,000077	0,000078	0,000141	0,000081
SelectionSort	0,001246	0,001277	0,001276	0,001270	0,001193	0,001280	0,001189	0,001244	0,001199	0,001288

10.000 instâncias

Metodo	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
MergeSort	0,001807	0,002496	0,002358	0,002311	0,002418	0,001722	0,002404	0,001745	0,001816	0,002688
RadixSort	0,001304	0,001498	0,001734	0,001388	0,001431	0,010799	0,001039	0,001281	0,001048	0,001221
SelectionSort	0,122236	0,148519	0,119812	0,127645	0,143068	0,171349	0,125107	0,125299	0,124941	0,123237

Metodo	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
MergeSort	0,001794	0,002243	0,002339	0,003064	0,002307	0,002292	0,001793	0,001760	0,003289	0,002380
RadixSort	0,001170	0,001350	0,001925	0,001699	0,001698	0,001239	0,001006	0,001154	0,001241	0,001403
SelectionSort	0,125243	0,122455	0,126403	0,130843	0,122105	0,123591	0,131919	0,133530	0,126073	0,135456

100.000 instâncias

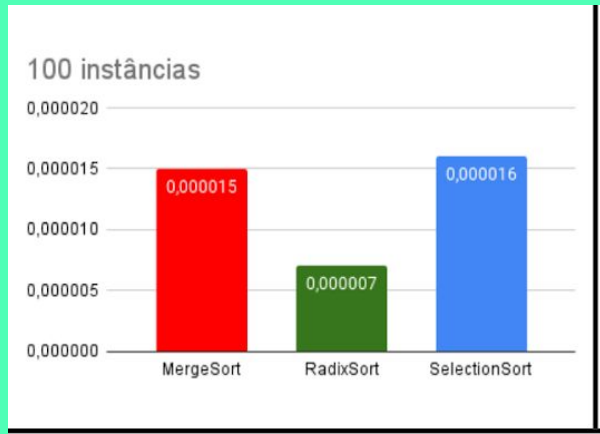
Metodo	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
MergeSort	0,028384	0,024560	0,031437	0,026097	0,027525	0,040001	0,025232	0,028719	0,024361	0,025788
RadixSort	0,016059	0,019050	0,021878	0,016647	0,020532	0,022086	0,017884	0,021650	0,042730	0,022738
SelectionSort	13,316941	11,999216	12,294515	12,827020	12,321020	12,183306	13,899768	12,585869	12,769027	12,189654

Metodo	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
MergeSort	0,023478	0,022682	0,030190	0,021610	0,027563	0,030429	0,030931	0,023500	0,025256	0,030096
RadixSort	0,018728	0,020951	0,027016	0,013608	0,018546	0,017197	0,023970	0,018033	0,022669	0,021616
SelectionSort	12,294147	12,432721	12,753176	12,412293	12,600752	11,987734	12,254060	12,762909	12,204842	12,911878

1.000.000 instâncias

Metodo	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
MergeSort	0,264256	0,241417	0,272875	0,238329	0,239961	0,274317	0,255418	0,264872	0,244880	0,264875
RadixSort	0,180955	0,203967	0,190613	0,203553	0,163242	0,163771	0,178749	0,197875	0,181521	0,185314
SelectionSort	1.150,263643	1.160,596512	1.157,628853	1.148,036706	1.147,924104	1.140,502756	1.154,216540	1.153,248790	1.159,487124	1.149,781976

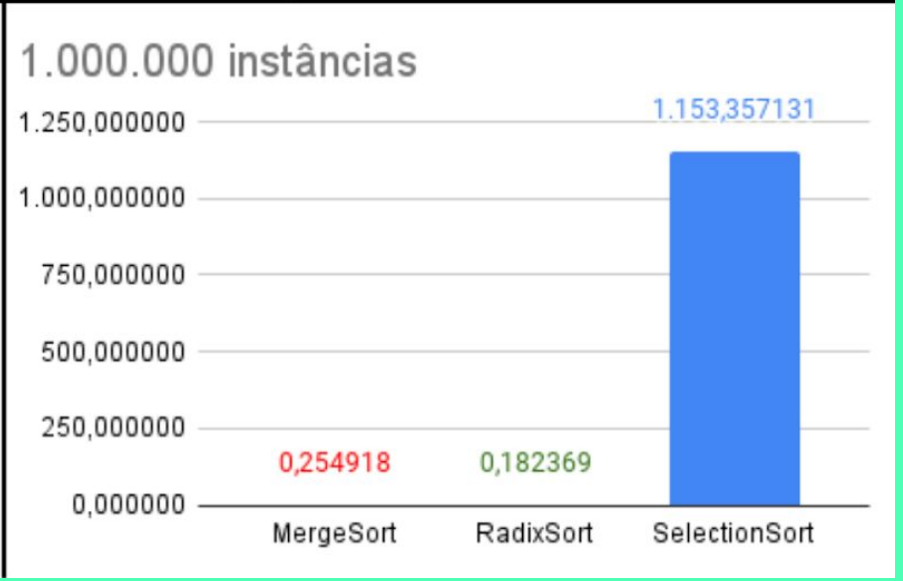
Metodo	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
MergeSort	0,261549	0,239187	0,247875	0,257844	0,254874	0,238114	0,241487	0,256784	0,251781	0,254962
RadixSort	0,171249	0,203165	0,171555	0,194579	0,183218	0,199486	0,164641	0,181143	0,234124	0,165469
SelectionSort	1.157,167814	1.160,781452	1.150,473148	1.143,487154	1.153,465471	1.166,615478	1.154,457620	1.148,484621	1.149,341560	1.156,669312


100 instâncias
0,000020
0,000015
0,000010
0,000005
0,000000
0,000015
0,000007
0,000016
MergeSort
RadixSort
SelectionSort


1.000 instâncias
0,001500
0,001000
0,000500
0,000000
0,000152
0,000085
0,001269
MergeSort
RadixSort
SelectionSort

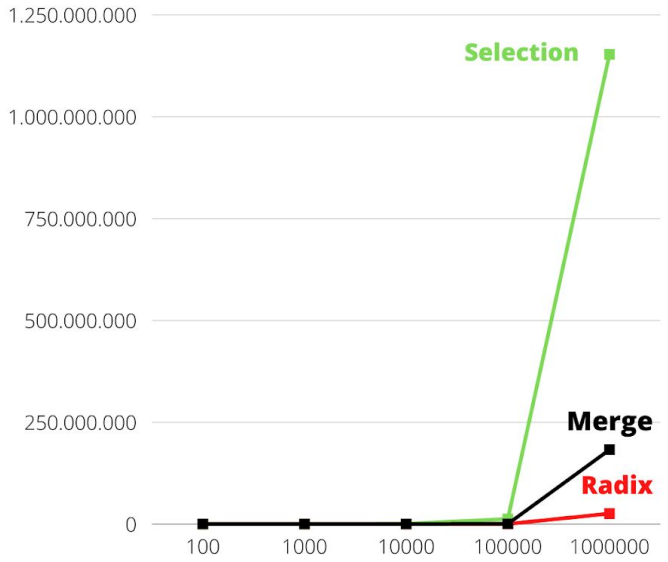

10.000 instâncias
0,150000
0,100000
0,050000
0,000000
0,002309
0,001327
0,125686
MergeSort
RadixSort
SelectionSort


100.000 instâncias
15,000000
10,000000
5,000000
0,000000
0,026811
0,020742
12,422507
MergeSort
RadixSort
SelectionSort


1.000.000 instâncias
1.250,000000
1.000,000000
750,000000
500,000000
250,000000
0,000000
0,254918
0,182369
1.153,357131
MergeSort
RadixSort
SelectionSort

Algumas conclusões

- Com instâncias pequenas averigua-se que a diferença de desempenho, embora exista, é bem pouca como com 100 e até 1.000 instâncias
- à medida que a entrada aumenta a diferença de desempenho fica nítida, com 1.000.000, 100.000 e até 10.000 o tempo de ordenação do selection se torna extremamente mais desvantajoso que os outros métodos



Teste estatístico, como fazer

Pareamento entre os 3 métodos	Calcular média e desvio padrão de cada pareamento	Calcular intervalo de confiança

Merge Sort =0,0006s

Selection Sort=0,003s

Pareamento Entre Merge e Selection

= 0,0006 – 0,003 = -0,0024

$$t_{[1-\frac{0.05}{2};20-1]} = 2,093$$

$$IC(95\%) = \bar{x} \pm 2,093 * \left(\frac{s}{\sqrt{20}}\right)$$

Interpretações

Caso 0 esteja no intervalo: Os dois métodos possuem desempenho similar para este tamanho de instância.

Caso o intervalo esteja à esquerda de 0: O método que representa o subtraendo possui o pior desempenho.

Caso o intervalo esteja à direita de 0: O método que representa o minuendo possui o pior desempenho.

	Info	Conclusões
100	<table><tr><th>PAREAMENTO</th><th>MÉDIA</th><th>DES.PADRÃO</th></tr><tr><td>Merge - Radix</td><td>0,000008</td><td>0,000012</td></tr><tr><td>Radix - Selection</td><td>-0,000010</td><td>0,000012</td></tr><tr><td>Merge - Selection</td><td>-0,000001</td><td>0,000002</td></tr></table>	Merge < Radix Selection < Radix Merge = Selection
1.000	<table><tr><th>PAREAMENTO</th><th>MÉDIA</th><th>DES.PADRÃO</th></tr><tr><td>Merge - Radix</td><td>0,000062</td><td>0,000028</td></tr><tr><td>Radix - Selection</td><td>-0,001169</td><td>0,000061</td></tr><tr><td>Merge - Selection</td><td>-0,001115</td><td>0,000060</td></tr></table>	Merge < Radix Selection < Radix Selection < Merge
10.000	<table><tr><th>PAREAMENTO</th><th>MÉDIA</th><th>DES.PADRÃO</th></tr><tr><td>Merge - Radix</td><td>0,000840</td><td>0,002271</td></tr><tr><td>Radix - Selection</td><td>-0,124276</td><td>0,010452</td></tr><tr><td>Merge - Selection</td><td>-0,123502</td><td>0,012141</td></tr></table>	Merge = Radix Selection < Radix Selection < Merge
100.000	<table><tr><th>PAREAMENTO</th><th>MÉDIA</th><th>DES.PADRÃO</th></tr><tr><td>Merge - Radix</td><td>0,007031</td><td>0,006976</td></tr><tr><td>Radix - Selection</td><td>-12,405228</td><td>0,462620</td></tr><tr><td>Merge - Selection</td><td>-12,400361</td><td>0,463068</td></tr></table>	Merge < Radix Selection < Radix Selection < Merge
1.000.000	<table><tr><th>PAREAMENTO</th><th>MÉDIA</th><th>DES.PADRÃO</th></tr><tr><td>Merge - Radix</td><td>0,075980</td><td>0,023175</td></tr><tr><td>Radix - Selection</td><td>-1.153,166584</td><td>6,307111</td></tr><tr><td>Merge - Selection</td><td>-1.153,097258</td><td>6,315128</td></tr></table>	Merge < Radix Selection < Radix Selection < Merge

Conclusões:

- Radix > Merge > Selection
- Análise estatística:
  Instância 10.000 empate entre Radix e Merge, possível erro estatístico.