UNIVERSIDADE FEDERAL DE OURO PRETO
CIÊNCIA DA COMPUTAÇÃO

PROFESSOR:

Anderson Almeida Ferreira

Pedro Henrique Oliveira da Silva - 20.1.4005
Pedro Lucas Damasceno Silva - 20.1.4003
Robson Novato Lobão - 20.1.4018

TRABALHO PRÁTICO - PROJETO E ANÁLISE DE ALGORITMOS (BCC241)

Avaliação Empírica

Ouro Preto, Minas Gerais

05 de setembro de 2022

Resumo

Neste trabalho foi realizado o estudo empírico de três algoritmos de ordenação e apresentado o seu resultado obtido. Variando o tamanho da instância de entrada e o método de ordenação utilizado, pudemos analisar estatisticamente seus resultados.

Introdução

O problema que vamos lidar é que precisamos fazer a análise de complexidade dos 3 métodos de ordenação, sendo eles o MergeSort, RadixSort e SelectionSort além de demonstrar estatisticamente seus resultados, em 20 execuções diferentes, com tamanhos das instâncias sendo: 100, 1.000, 10.000, 100.000 e 1.000.000, objetivamos mostrar de forma ilustrativa e simplista o desempenho dos algoritmos em uma máquina específica, além da sua análise de complexidade.

Descrição dos métodos

Os métodos de ordenação abordados, assim como dito anteriormente, foram o MergeSort, RadixSort e SelectionSort. Falando um pouco sobre o Selection Sort ele é um algoritmo que tem complexidade O(n^2) e funciona baseado no princípio de passar sempre o menor valor do vetor para a primeira posição, o segundo menor para a segunda e assim sucessivamente até os dois últimos elementos.

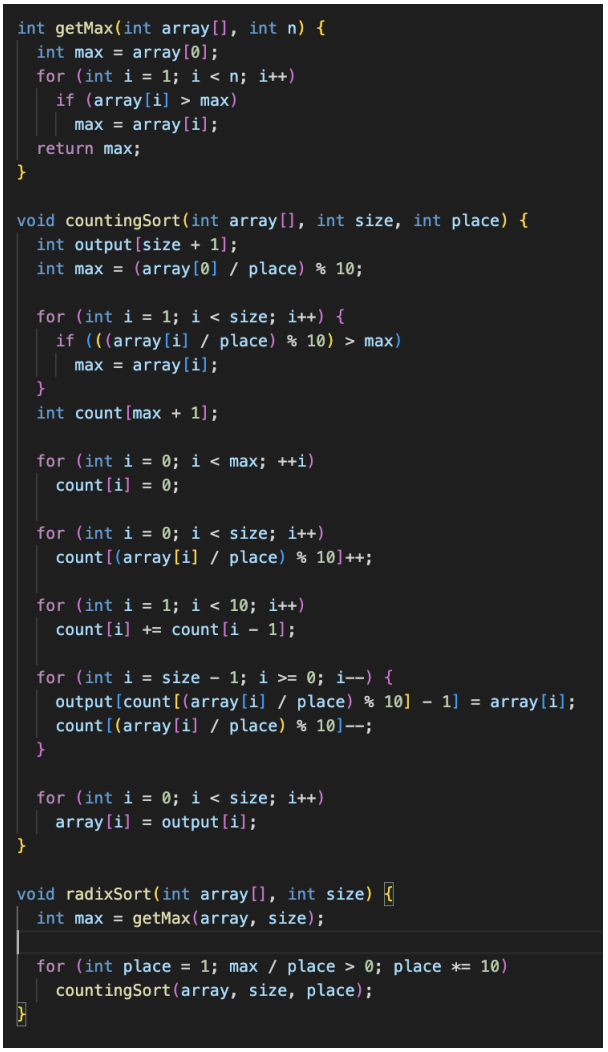
```
void selectionSort (int *vetor, int n) {
    int aux;
    int menorAtual;
    for(int i = 0; i < n; i++) {
        menorAtual = i;
        for (int j = i + 1; j < n; j++) {
            if (vetor[j] < vetor[menorAtual])
                menorAtual = j;
        }
        if(menorAtual != i) {
     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

Agora falando sobre o MergeSort, ele é um algoritmo dividir para conquistar e baseado em ordenação por comparação, ele subdivide o problema em subproblemas e os resolve em recursão, depois os junta, causando consumo de memória e tempo de execução, possui complexidade O(n log n) e foi implementado da seguinte forma:

```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);
}
```

Sobre o RadixSort, ele é um algoritmo de ordenação que funciona de forma diferente, levando em conta a inteiros processando dígitos individuais , como os inteiros podem representar strings compostas. Como nosso radix é LSD, dígito menos significativo, ele ordena da seguinte forma: começa do dígito menos significativo até o mais significativo, ordenando tipicamente da seguinte forma: chaves curtas vem antes de chaves longas, e chaves de mesmo tamanho são ordenadas lexicograficamente. Isso coincide com a ordem normal de representação dos inteiros, como a sequência "1, 2, 3, 4, 5, 6, 7, 8, 9, 10". O nosso RadixSort foi implementado assim:



```
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];
}

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

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

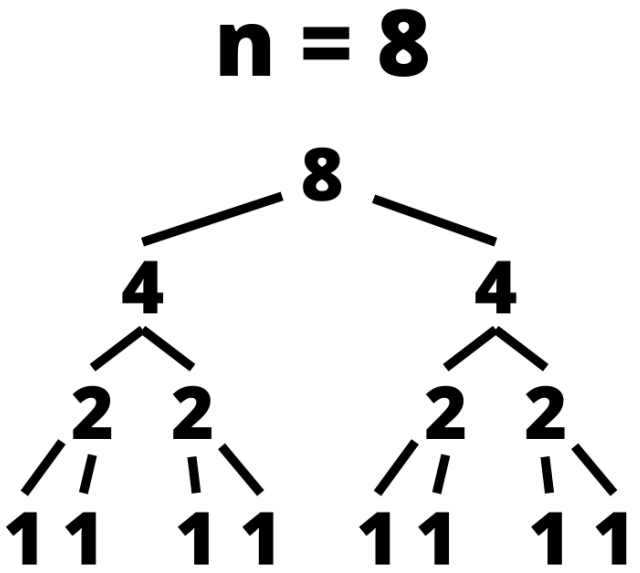
Ele é assintoticamente O(n+k), onde n é o número de chaves, e k é o comprimento médio da chave.

Análise assintótica dos métodos

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

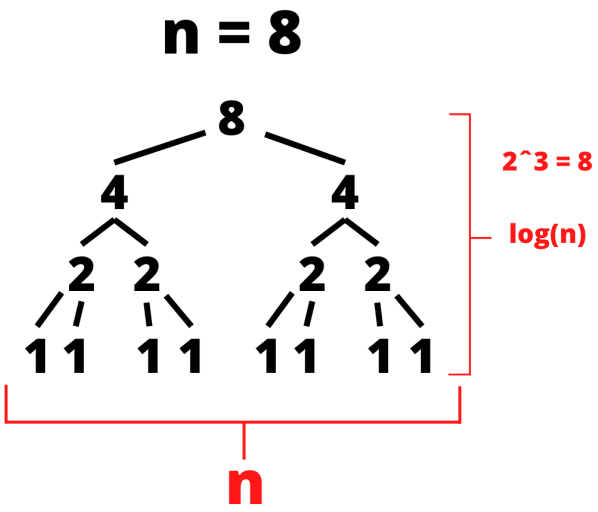
MergeSort:

Para começar o algoritmo de mergeSort é dividido em duas partes, uma de divisão e o outro de merge. Na parte de divisão ele "splita" a lista de números não ordenados até chegar em um caso base onde o número de itens no vetor é 1, o que torna aquela mini porção do vetor já ordenado, então ele volta para a segunda fase do merge e junta com a lista já ordenada que foi divida anteriormente, podendo ser vista na árvore exemplo:



No caso base 1, é uma lista ordenada que passa para o caso base 2 que junta com o outro 1 e torna no nível a lista ordenada, e assim sucessivamente. Veja que, como nossa entrada é 8, e ele vai se dividindo até chegar no caso base, a sua complexidade vai se alterando em valor de log, exemplo: temos 8 de entrada e o algoritmo funciona em 3 níveis de divisão, o nível dos quatros, dos dois e dos uns. Temos que os 3 níveis podem ser descritos como o lg(8) = 3 ou o log(n) é a análise

assintótica feita da parte de divisão do MergeSort. E como podemos ver na parte das folhas o tamanho da entrada é que vai ditar a quantidade de operações realizadas nas folhas básicas para achar a mínima lista ordenada. Ocasionando um algoritmo O(n * log n).



SelectionSort:

Com relação ao Selection Sort sua análise assintótica é mais simples, pois os dois laços que o selection possui serão sempre executados resultando em uma análise assintótica de O(n^2).

```
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;
   }
  }
}
```

RadixSort:

A análise assintótica do RadixSort é um pouco mais complexa, sua análise é O(n+k) onde o n é o tamanho do vetor e k é a quantidade de dígitos do maior número do vetor, e o + funciona como um "ou", ou seja, o valor depende do que for maior. O algoritmo itera sobre k número de dígitos do maior número do vetor e para cada iteração faz as seguintes ações:

- 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.

Nós ignoramos as constantes e temos: $O(k \cdot (b + n))$

O custo é independente de como os números de entrada são organizados. Se eles são distribuídos aleatoriamente ou pré-ordenados não faz diferença para o algoritmo. O melhor caso, o caso médio e o pior caso são, portanto, idênticos. A fórmula parece complicada no início. Mas duas das três variáveis não são variáveis na maioria das situações. Por exemplo, se ordenarmos longs com base 10, podemos substituir k por 19 (o valor máximo possível para um long é 9.223.372.036.854.775.807) e b por 10. A fórmula então se torna $O(19 \cdot (10 + n))$. Podemos omitir constantes; assim, obtemos: A complexidade de tempo para RadixSort com um comprimento máximo conhecido dos elementos para classificar e com base fixa é: O(n).

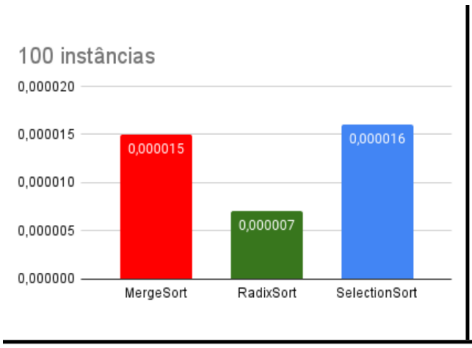
Avaliação experimental

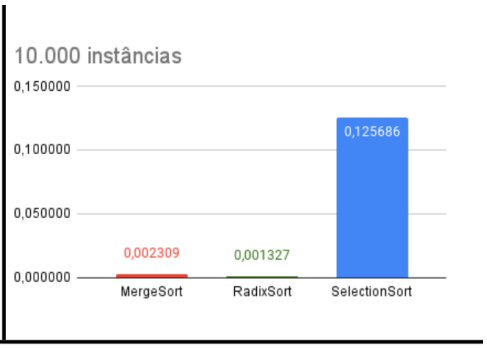
Inicialmente, é importante pontuar as especificações as quais trabalhamos para encontrar os resultados: Foi utilizada a Linguagem C, no computador com as seguintes especificações:

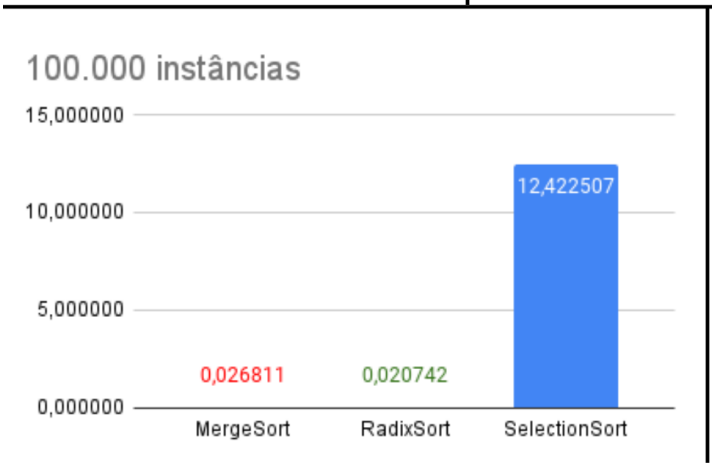
- Intel Core i5-4690 4x3,5 GHz
- 8GB RAM DDR3 1666MHz
- Sistema Operacional: Ubuntu 20.04 LTS (WSL)
- Compilador: GCC 9.4
- Gerador de instâncias aleatória com função rand() da linguagem C e com o seed rand no time(NULL).

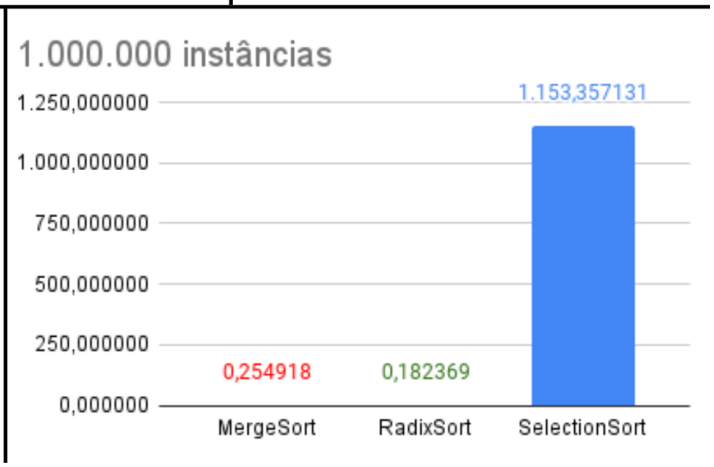
```c
srand(time(NULL));
clock_t start, end;
double time[3];
int n = atoi(argv[1]);
int **arr = criarMatriz(3, n);

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

Vamos começar avaliando o tamanho da instância e o resultado em média obtido após 20 execuções dos algoritmos Merge, Selection e Radix em segundos:

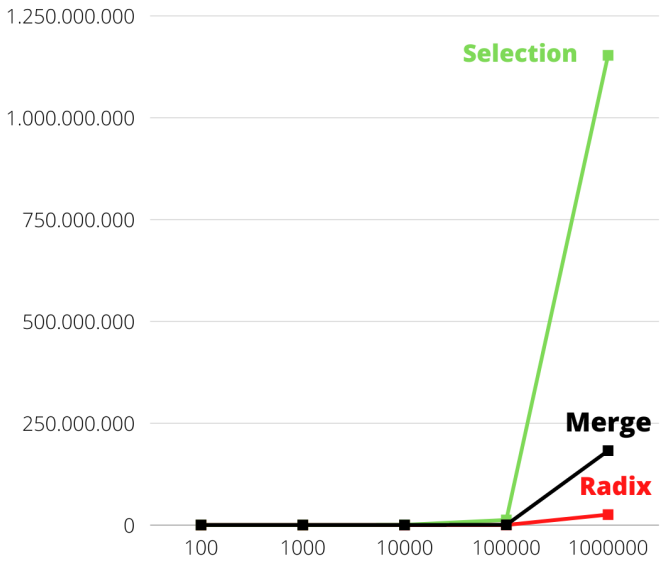








Com instâncias pequenas averigua-se que a diferença de desempenho, embora exista, é bem pouca como com 100 e até 1.000 instâncias, entretanto, à medida que a entrada aumenta a diferença de desempenho fica nítida, com 1.000.000, 100.000 e até 10.000 o tempo de ordenação do selection se torna extremamente mais desvantajoso que os outros métodos, como pode ser visto no gráfico:



Teste estatístico

O teste estatístico t pareado com 95% de confiança consiste em 4 etapas:

1. Pareamento entre os 3 métodos. Como cada método contém 20 instâncias, basta fazer uma diferença entre 2 instâncias de métodos diferentes. Ex:
   Merge Sort = 0,0006s
   Selection Sort = 0,003s
   Pareamento Entre Merge e Selection = 0,0006 – 0,003 = -0,0024
2. Calcular a média e o desvio padrão para cada um dos 3 pareamentos.
3. Calcular o Intervalo de Confiança com 95% de confiança, onde:

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

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

4. Baseado no intervalo obtido, pode ser feito as seguintes interpretações:
    a. Caso 0 esteja no intervalo: Os dois métodos possuem desempenho similar para este tamanho de instância.
    b. Caso o intervalo esteja à esquerda de 0: O método que representa o subtraendo possui o pior desempenho.
    c. Caso o intervalo esteja à direita de 0: O método que representa o minuendo possui o pior desempenho.

Tendo o teste esclarecido, os cálculos para as instâncias de tamanho 100, 1000, 10000, 100000 e 1000000 seguem abaixo.

PARA TAM = 100

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

Tendo os resultados para instâncias de tamanho 100 para os 3 métodos, basta fazer um pareamento entre os métodos.

Metodo	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
MergeSort	0,000007	0,000008	0,000009	0,000009	0,000008	0,000009	0,000008	0,000009	0,000008	0,000009
RadixSort	-0,000007	-0,000010	-0,000010	-0,000011	-0,000009	-0,000011	-0,000009	-0,000010	-0,000009	-0,000010
SelectionSort	0,000000	-0,000002	-0,000001	-0,000002	-0,000001	-0,000002	-0,000001	-0,000001	-0,000001	-0,000001
Metodo	**T11**	**T12**	**T13**	**T14**	**T15**	**T16**	**T17**	**T18**	**T19**	**T20**
MergeSort	0,000006	0,000009	0,000007	0,000008	0,000005	0,000005	-0,000044	0,000008	0,000010	0,000009
RadixSort	-0,000007	-0,000011	-0,000009	-0,000010	-0,000006	-0,000007	0,000045	-0,000010	-0,000018	-0,000010
SelectionSort	-0,000001	-0,000002	-0,000002	-0,000002	-0,000001	-0,000002	0,000001	-0,000002	-0,000008	-0,000001

Tendo o resultado de cada pareamento, basta calcular a média e desvio padrão em cada pareamento.

PAREAMENTO	MÉDIA	DES.PADRAO
Merge - Radix	0,000008	0,000012
Radix - Selection	-0,000010	0,000012
Merge - Selection	-0,000001	0,000002

Com isso, basta aplicar a fórmula para encontrar cada intervalo com 95% de confiança:

MERGE - RADIX

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

$$Linf = 0,000008 - 2,093 * \left(\frac{0,000012}{\sqrt{20}}\right)$$

$$Linf = 0,000008 - 2,093 * 0,000003$$

$$Linf = 0,000008 - 0,000005 = 0,000003$$

$$Lsup = 0,000008 + 2,093 * \left(\frac{0,000012}{\sqrt{20}}\right)$$

$$Lsup = 0,000008 + 2,093 * 0,000003$$

$$Linf = 0,000008 + 0,000005 = 0,000013$$

$$IC(95\%) = [0,000003; 0,000013]$$

Como o intervalo está à direita de 0, dizemos que o algoritmo que representa o minuendo tem o pior desempenho, ou seja, para instâncias de tamanho 100, o Merge Sort possui desempenho pior que o Radix Sort.

RADIX - SELECTION

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

$$Linf = - 0,000010 - 2,093 * \left(\frac{0,000012}{\sqrt{20}}\right)$$

$$Linf = - 0,000010 - 2,093 * 0,000003$$

$$Linf = - 0,000010 - 0,000006 = - 0,000016$$

$$Lsup = - 0,000010 + 2,093 * \left(\frac{0,000012}{\sqrt{20}}\right)$$

$$Lsup = - 0,000010 + 2,093 * 0,000003$$

$$Lsup = - 0,000010 + 0,000006 = - 0,000004$$

$$IC(95\%) = [- 0,000016; - 0,000004]$$

Como o intervalo está à esquerda de 0, dizemos que o algoritmo que representa o subtraendo tem o pior desempenho, ou seja, para instâncias de tamanho 100, o Selection Sort possui desempenho pior que o Radix Sort.

MERGE - SELECTION

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

$$Linf = -0,000001 - 2,093 * \left(\frac{0,000002}{\sqrt{20}}\right)$$

$$Linf = -0,000001 - 2,093 * 0,0000004$$

$$Linf = -0,000001 - 0,000001 = -0,000002$$

$$Lsup = -0,000001 + 2,093 * \left(\frac{0,000002}{\sqrt{20}}\right)$$

$$Lsup = -0,000001 + 2,093 * 0,0000004$$

$$Lsup = -0,000001 + 0,000001 = 0,0$$

$$IC(95\%) = [-0,000002; 0,0]$$

Como 0 está contido no intervalo, dizemos que o Merge Sort e o Selection Sort têm desempenhos similares para instâncias de tamanho 100.

PARA TAM = 1000

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

Tendo os resultados para instâncias de tamanho 1000 para os 3 métodos, basta fazer um pareamento entre os métodos.

Pareamento	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Merge - Radix	0,000051	0,000070	0,000019	0,000067	0,000072	0,000057	0,000003	0,000017	0,000064	0,000063
Radix - Selection	-0,001166	-0,001178	-0,001212	-0,001195	-0,001205	-0,001169	-0,001118	-0,001317	-0,001156	-0,001277
Merge - Selection	-0,001115	-0,001108	-0,001193	-0,001128	-0,001133	-0,001112	-0,001115	-0,001300	-0,001092	-0,001214
Pareamento	**T11**	**T12**	**T13**	**T14**	**T15**	**T16**	**T17**	**T18**	**T19**	**T20**
Merge - Radix	0,000063	0,000005	0,000070	0,000015	0,000025	0,000067	0,000093	0,000067	0,000003	0,000061
Radix - Selection	-0,001169	-0,001129	-0,001194	-0,001130	-0,001074	-0,001196	-0,001112	-0,001166	-0,001058	-0,001207
Merge - Selection	-0,001106	-0,001124	-0,001124	-0,001115	-0,001049	-0,001129	-0,001019	-0,001099	-0,001055	-0,001146

Tendo o resultado de cada pareamento, basta calcular a média e desvio padrão em cada pareamento.

PAREAMENTO	MÉDIA	DES.PADRAO
Merge - Radix	0,000062	0,000028
Radix - Selection	-0,001169	0,000061
Merge - Selection	-0,001115	0,000060

Com isso, basta aplicar a fórmula para encontrar cada intervalo com 95% de confiança:

MERGE - RADIX

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

$$Linf = 0,000062 - 2,093 * \left(\frac{0,000028}{\sqrt{20}}\right)$$

$$Linf = 0,000062 - 2,093 * 0,000006$$

$$Linf = 0,000062 - 0,000013 = 0,000049$$

$$Lsup = 0,000062 + 2,093 * \left(\frac{0,000028}{\sqrt{20}}\right)$$

$$Lsup = 0,000062 + 2,093 * 0,000006$$

$$Lsup = 0,000062 + 0,000013 = 0,000075$$

$$IC(95\%) = [0,000049; 0,000075]$$

Como o intervalo está à direita de 0, dizemos que o algoritmo que representa o minuendo tem o pior desempenho, ou seja, para instâncias de tamanho 1000, o Merge Sort possui desempenho pior que o Radix Sort.

RADIX - SELECTION

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

$$Linf = -0,001169 - 2,093 * \left(\frac{0,000061}{\sqrt{20}}\right)$$

$$Linf = -0,001169 - 2,093 * 0,000014$$

$$Linf = -0,001169 - 0,000029 = -0,001198$$

$$Lsup = -0,001169 + 2,093 * \left(\frac{0,000061}{\sqrt{20}}\right)$$

$$Lsup = -0,001169 + 2,093 * 0,000014$$

$$Lsup = -0,001169 + 0,000029 = -0,001140$$

$$IC(95\%) = [-0,01198; -0,001140]$$

Como o intervalo está à esquerda de 0, dizemos que o algoritmo que representa o subtraendo tem o pior desempenho, ou seja, para instancias de tamanho 1000, o Selection Sort possui desempenho pior que o Radix Sort.

MERGE - SELECTION

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

$$Linf = -0,001115 - 2,093 * \left(\frac{0,000060}{\sqrt{20}}\right)$$

$$Linf = -0,001115 - 2,093 * 0,000013$$

$$Linf = -0,001115 - 0,000028 = -0,001143$$

$$Lsup = -0,001115 + 2,093 * \left(\frac{0,000060}{\sqrt{20}}\right)$$

$$Lsup = -0,001115 + 2,093 * 0,000013$$

$$Lsup = -0,001115 + 0,000028 = -0,001087$$

$$IC(95\%) = [-0,001143; -0,001087]$$

Como o intervalo está à esquerda de 0, dizemos que o algoritmo que representa o subtraendo tem o pior desempenho, ou seja, para instâncias de tamanho 1000, o Selection Sort possui desempenho pior que o Merge Sort.

PARA TAM = 10000

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

Tendo os resultados para instâncias de tamanho 10000 para os 3 métodos, basta fazer um pareamento entre os métodos.

Pareamento	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Merge - Radix	0,000503	0,000998	0,000624	0,000923	0,000987	-0,009077	0,001365	0,000464	0,000768	0,001467
Radix - Selection	-0,120932	-0,147021	-0,118078	-0,126257	-0,141637	-0,160550	-0,124068	-0,124018	-0,123893	-0,122016
Merge - Selection	-0,120429	-0,146023	-0,117454	-0,125334	-0,140650	-0,169627	-0,122703	-0,123554	-0,123125	-0,120549

Pareamento	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
Merge - Radix	0,000624	0,000893	0,000414	0,001365	0,000609	0,001053	0,000787	0,000606	0,002048	0,000977
Radix - Selection	-0,124073	-0,121105	-0,124478	-0,129144	-0,120407	-0,122352	-0,130913	-0,132376	-0,124832	-0,134053
Merge - Selection	-0,123449	-0,120212	-0,124064	-0,127779	-0,119798	-0,121299	-0,130126	-0,131770	-0,122784	-0,133076

Tendo o resultado de cada pareamento, basta calcular a média e desvio padrão em cada pareamento.

PAREAMENTO	MÉDIA	DES.PADRAO
Merge - Radix	0,000840	0,002271
Radix - Selection	-0,124276	0,010452
Merge - Selection	-0,123502	0,012141

Com isso, basta aplicar a fórmula para encontrar cada intervalo com 95% de confiança:

MERGE - RADIX

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

$$Linf = 0,000840 - 2,093 * \left(\frac{0,002271}{\sqrt{20}}\right)$$

$$Linf = 0,000840 - 2,093 * 0,000508$$

$$Linf = 0,000840 - 0,001063 = -0,000223$$

$$Lsup = 0,000840 + 2,093 * \left(\frac{0,002271}{\sqrt{20}}\right)$$

$$Lsup = 0,000840 + 2,093 * 0,000508$$

$$Lsup = 0,000840 + 0,001063 = 0,001903$$

$$IC(95\%) = [-0,000223; 0,001903]$$

Como 0 está contido no intervalo, dizemos que o Merge Sort e o Radix Sort têm desempenhos similares para instâncias de tamanho 10000.

RADIX - SELECTION

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

$$Linf = -0,124276 - 2,093 * \left(\frac{0,010452}{\sqrt{20}}\right)$$

$$Linf = -0,124276 - 2,093 * 0,002337$$

$$Linf = -0,124276 - 0,004891 = -0,129167$$

$$Lsup = -0,124276 + 2,093 * \left(\frac{0,010452}{\sqrt{20}}\right)$$

$$Lsup = -0,124276 + 2,093 * 0,002337$$

$$Lsup = -0,124276 + 0,004891 = -0,119385$$

$$IC(95\%) = [-0,129167; -0,119385]$$

Como o intervalo está à esquerda de 0, dizemos que o algoritmo que representa o subtraendo tem o pior desempenho, ou seja, para instâncias de tamanho 10000, o Selection Sort possui desempenho pior que o Radix Sort.

MERGE - SELECTION

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

$$Linf = -0,123502 - 2,093 * \left(\frac{0,012141}{\sqrt{20}}\right)$$

$$Linf = -0,123502 - 2,093 * 0,002715$$

$$Linf = -0,123502 - 0,005682 = -0,129183$$

$$Lsup = -0,123502 + 2,093 * \left(\frac{0,012141}{\sqrt{20}}\right)$$

$$Lsup = -0,123502 + 2,093 * 0,002715$$

$$Lsup = -0,123502 + 0,005682 = -0,117820$$

$$IC(95\%) = [-0,129183; -0,117820]$$

Como o intervalo está à esquerda de 0, dizemos que o algoritmo que representa o subtraendo tem o pior desempenho, ou seja, para instâncias de tamanho 10000, o Selection Sort possui desempenho pior que o Merge Sort.

PARA TAM = 100000

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

Tendo os resultados para instâncias de tamanho 100000 para os 3 métodos, basta fazer um pareamento entre os métodos.

Pareamento	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Merge - Radix	0,012325	0,005510	0,009559	0,009450	0,006993	0,017915	0,007348	0,007069	-0,018369	0,003050
Radix - Selection	-13,300882	-11,980166	-12,272637	-12,810373	-12,300488	-12,161220	-13,881884	-12,564219	-12,726297	-12,166916
Merge - Selection	-13,288557	-11,974656	-12,263078	-12,800923	-12,293495	-12,143305	-13,874536	-12,557150	-12,744666	-12,163866

Pareamento	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
Merge - Radix	0,004750	0,001731	0,003174	0,008002	0,009017	0,013232	0,006961	0,005467	0,002587	0,008480
Radix - Selection	-12,275419	-12,411770	-12,726160	-12,398685	-12,582206	-11,970537	-12,230090	-12,744876	-12,182173	-12,890262
Merge - Selection	-12,270669	-12,410039	-12,722986	-12,390683	-12,573189	-11,957305	-12,223129	-12,739409	-12,179586	-12,881782

Tendo o resultado de cada pareamento, basta calcular a média e desvio padrão em cada pareamento.

PAREAMENTO	MÉDIA	DES.PADRAO
Merge - Radix	0,007031	0,006976
Radix - Selection	-12,405228	0,462620
Merge - Selection	-12,400361	0,463068

Com isso, basta aplicar a fórmula para encontrar cada intervalo com 95% de confiança:

MERGE - RADIX

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

$$Linf = 0,007031 - 2,093 * \left(\frac{0,006976}{\sqrt{20}}\right)$$

$$Linf = 0,007031 - 2,093 * 0,001560$$

$$Linf = 0,007031 - 0,003265 = 0,003766$$

$$Lsup = 0,007031 + 2,093 * \left(\frac{0,006976}{\sqrt{20}}\right)$$

$$Lsup = 0,007031 + 2,093 * 0,001560$$

$$Lsup = 0,007031 + 0,003265 = 0,010296$$

$$IC(95\%) = [0,003766; 0,010296]$$

Como o intervalo está à direita de 0, dizemos que o algoritmo que representa o minuendo tem o pior desempenho, ou seja, para instancias de tamanho 100000, o Merge Sort possui desempenho pior que o Radix Sort.

RADIX - SELECTION

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

$$Linf = -12,405228 - 2,093 * \left(\frac{0,462620}{\sqrt{20}}\right)$$

$$Linf = -12,405228 - 2,093 * 0,103445$$

$$Linf = -12,405228 - 0,216510 = -12,621738$$

$$Lsup = -12,405228 + 2,093 * \left(\frac{0,462620}{\sqrt{20}}\right)$$

$$Lsup = -12,405228 + 2,093 * 0,103445$$

$$Lsup = -12,405228 + 0,216510 = -12,188717$$

$$IC(95\%) = [-12,621738; -12,188717]$$

Como o intervalo está à esquerda de 0, dizemos que o algoritmo que representa o subtraendo tem o pior desempenho, ou seja, para instâncias de tamanho 100000, o Selection Sort possui desempenho pior que o Radix Sort.

MERGE - SELECTION

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

$$Linf = -12,400361 - 2,093 * \left(\frac{0,463068}{\sqrt{20}}\right)$$

$$Linf = -12,400361 - 2,093 * 0,103545$$

$$Linf = -12,400361 - 0,216720 = -12,617081$$

$$Lsup = -12,400361 + 2,093 * \left(\frac{0,463068}{\sqrt{20}}\right)$$

$$Lsup = -12,400361 + 2,093 * 0,103545$$

$$Lsup = -12,400361 + 0,216720 = -12,183641$$

$$IC(95\%) = [-12,617081; -12,183641]$$

Como o intervalo está à esquerda de 0, dizemos que o algoritmo que representa o subtraendo tem o pior desempenho, ou seja, para instâncias de tamanho 100000, o Selection Sort possui desempenho pior que o Merge Sort.

PARA TAM = 1000000

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

Tendo os resultados para instâncias de tamanho 1000000 para os 3 métodos, basta fazer um pareamento entre os métodos.

Pareamento	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
Merge - Radix	0,083301	0,037450	0,082262	0,034776	0,076719	0,110546	0,076669	0,066997	0,063359	0,079561
Radix - Selection	-1.150,082688	-1.160,392545	-1.157,438240	-1.147,833153	-1.147,760862	-1.140,338985	-1.154,037791	-1.153,050916	-1.159,305603	-1.149,596662
Merge - Selection	-1.149,999387	-1.160,355095	-1.157,355978	-1.147,798377	-1.147,684143	-1.140,228439	-1.153,961122	-1.152,983919	-1.159,242244	-1.149,517101

Pareamento	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
Merge - Radix	0,090300	0,036022	0,076320	0,063266	0,071656	0,038628	0,076846	0,075641	0,017657	0,089493
Radix - Selection	-1.156,996565	-1.160,578287	-1.150,301593	-1.143,292575	-1.153,282253	-1.166,415992	-1.154,292979	-1.148,303478	-1.149,107436	-1.156,503843
Merge - Selection	-1.156,906265	-1.160,542265	-1.150,225274	-1.143,229310	-1.153,210597	-1.166,377364	-1.154,216133	-1.148,227837	-1.149,089779	-1.156,414350

Tendo o resultado de cada pareamento, basta calcular a média e desvio padrão em cada pareamento.

PAREAMENTO	MÉDIA	DES.PADRAO
Merge - Radix	0,075980	0,023175
Radix - Selection	-1.153,166584	6,307111
Merge - Selection	-1.153,097258	6,315128

Com isso, basta aplicar a fórmula para encontrar cada intervalo com 95% de confiança:

MERGE - RADIX

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

$$Linf = 0,075980 - 2,093 * \left(\frac{0,023175}{\sqrt{20}}\right)$$

$$Linf = 0,075980 - 2,093 * 0,005182$$

$$Linf = 0,075980 - 0,010846 = 0,065134$$

$$Lsup = 0,075980 + 2,093 * \left(\frac{0,023175}{\sqrt{20}}\right)$$

$$Lsup = 0,075980 + 2,093 * 0,005182$$

$$Lsup = 0,075980 + 0,010846 = 0,086827$$

$$IC(95\%) = [0,065134; 0,086827]$$

Como o intervalo está à direita de 0, dizemos que o algoritmo que representa o minuendo tem o pior desempenho, ou seja, para instâncias de tamanho 1000000, o Merge Sort possui desempenho pior que o Radix Sort.

RADIX - SELECTION

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

$$Linf = -1.153,166584 - 2,093 * \left(\frac{6,307111}{\sqrt{20}}\right)$$

$$Linf = -1.153,166584 - 2,093 * 1,410313$$

$$Linf = -1.153,166584 - 2,951785 = -1.156,118369$$

$$Lsup = -1.153,166584 + 2,093 * \left(\frac{6,307111}{\sqrt{20}}\right)$$

$$Lsup = -1.153,166584 + 2,093 * 1,410313$$

$$Lsup = -1.153,166584 + 2,951785 = -1.150,214799$$

$$IC(95\%) = [-1.156,118369; -1.150,214799]$$

Como o intervalo está à esquerda de 0, dizemos que o algoritmo que representa o subtraendo tem o pior desempenho, ou seja, para instâncias de tamanho 1000000, o Selection Sort possui desempenho pior que o Radix Sort.

MERGE - SELECTION

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

$$Linf = -1.153,097258 - 2,093 * \left(\frac{6,315128}{\sqrt{20}}\right)$$

$$Linf = -1.153,097258 - 2,093 * 1,412105$$

$$Linf = -1.153,097258 - 2,955537 = -1.156,052794$$

$$Lsup = -1.153,097258 + 2,093 * \left(\frac{6,315128}{\sqrt{20}}\right)$$

$$Lsup = -1.153,097258 + 2,093 * 1,412105$$

$$Lsup = -1.153,097258 + 2,955537 = -1.150,141721$$

$$IC(95\%) = [-1.156,052794; -1.150,141721]$$

Como o intervalo está à esquerda de 0, dizemos que o algoritmo que representa o subtraendo tem o pior desempenho, ou seja, para instâncias de tamanho 1000000, o Selection Sort possui desempenho pior que o Merge Sort.

Conclusão

Assim como se confirma pela análise assintótica, o Radix Sort apresenta o melhor desempenho comparado aos outros 2 algoritmos, mesmo com instâncias de menor tamanho testado. O Merge Sort é o algoritmo que apresenta o desempenho mediano, enquanto o Selection Sort é aquele que apresenta o pior desempenho.

Na análise estatística t pareada com 95% entre Radix e o Merge para tamanho de instância 10000, os 2 métodos deram empate estatístico. Entretanto, isso pode ser um erro do tipo 2 do teste de hipótese, onde aceita-se a hipótese (Desempenho do Radix ser equivalente ao Merge para instâncias de tamanho 10000) quando esta é falsa.