UNIVERSIDADE FEDERAL DE OURO PRETO
DEPARTAMENTO DE COMPUTAÇÃO

AVALIAÇÃO EMPÍRICA

Projeto e Análise de Algoritmos - BCC241

Caio Monteiro de Oliveira - 20.1.4110
Gabriel Carvalho Domingos da Conceição - 20.1.4114
Pablo Martins Coelho - 20.1.4113

Professor:
Anderson Almeida Ferreira

OURO PRETO – MG 2022

Resumo

O presente trabalho tem como objetivo apresentar a avaliação empírica de 3 algoritmos para ordenação das chaves de um vetor. Dessa forma, os 3 algoritmos utilizados na avaliação são o Merge Sort, o Selection Sort e o Radix Sort. Durante o desenvolvimento do trabalho são elucidados os objetivos da pesquisa, os resultados que foram encontrados - utilizando do método de teste ‘t’ pareado com 95% de confiança -, e também, a forma como o trabalho foi organizado. Além disso, é mostrado como foi obtida a ordem de complexidade de cada algoritmo, com a descrição detalhada de cada um deles e seu funcionamento, por fim, é explicada a implementação do código para a obtenção dos tempos de execução de cada algoritmo, com a finalidade de obter os resultados procurando avaliar a eficiência dos mesmos. Como forma de demonstrar visualmente o desempenho de cada algoritmo com os diferentes tamanhos dos vetores, foi utilizado de gráficos onde é possível averiguar o tempo de execução da ordenação de cada um deles, e, através disso realizar a comparação do gráfico de cada algoritmo. Dessa forma, a avaliação foi feita comparando os resultados obtidos, sendo usadas 20 instâncias de mesmo tamanho com números aleatórios, com seus tamanhos variando de 100, aumentando a cada teste em potência de 10, até 1.000.000.

Palavras - chave: complexidade, resultados, comparação, avaliação.

SUMÁRIO

Introdução

O problema em questão a ser solucionado com o trabalho é avaliar qual dos três métodos de ordenação - Merge Sort, Selection Sort e Radix Sort - é mais eficiente sendo analisado em diferentes testes, neste caso, testamos a ordenação para 20 vetores com o mesmo tamanho e o conteúdo dentro deles em ordem aleatória. Também, é importante ressaltar que alguns algoritmos podem funcionar bem para instâncias pequenas, porém para instâncias grandes ele requer um tempo muito grande para poder ordenar o vetor, além de ocupar um espaço enorme durante a ordenação. Da mesma forma, um algoritmo pode se comportar de maneira menos eficiente para instâncias de pequenos tamanhos e na medida em que essas instâncias crescem o método se mostra bastante eficiente.

O seguinte trabalho tem como objetivo a análise e comparação dos resultados obtidos através dos experimentos realizados com cada um dos métodos de ordenação propostos em diferentes casos de teste. Através dessa análise é possível identificar em quais casos cada algoritmo se mostra mais eficaz.

Por meio da utilização do teste estatístico t pareado com 95% de confiança, pode-se verificar se houve empate estatístico ou informar qual método obteve estatisticamente o melhor desempenho. Dessa forma, os dados que foram utilizados como amostra para averiguar os testes, foram o tempo de execução de cada um dos algoritmos sob as mesmas condições. Após a execução de todos os testes com as amostras usando diferentes tamanhos, é evidente que o Radix Sort é o algoritmo mais eficiente dentre os três avaliados, sendo assim, a complexidade de tempo dele é $O(n \times k)$, em seguida, tem-se o Merge Sort com a complexidade de tempo $O(n \times \log n)$ e por fim, com o pior desempenho fica o Selection Sort que tem complexidade de tempo $O(n^2)$.

O seguinte trabalho está organizado em três partes, que são, os métodos e seus custos de complexidade, as métricas utilizadas para avaliação dos experimentos realizados e por fim os resultados obtidos através dos experimentos.

Descrição dos métodos com suas análises de complexidades

MergeSort

O merge sort, ou ordenação por mistura, 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).

Os três passos úteis dos algoritmos de divisão e conquista, que se aplicam ao merge sort são:
1 - Dividir: Calcula o ponto médio do sub-arranjo, o que demora um tempo constante O(1);
2 - Conquistar: Recursivamente resolve dois subproblemas, cada um de tamanho n / 2, o que contribui com 2T(n / 2) para o tempo de execução;
3 - Combinar: Unir os sub-arranjos em um único conjunto ordenado, que leva o tempo O(n);

Análise de complexidade do MergeSort:
O MergeSort usa duas funções, uma como principal a “mergeSort” e a que é chamada pela principal, a “merge”. De início, analisando a função “merge”, da linha 171 até a linha 177 a complexidade é O(1), tendo custo constante. Na linha 180 e na linha 183, o custo é O(n / 2), pois o laço de repetição for, caminha primeiro uma metade do vetor e depois a outra, da linha 185 até a 188 o custo é constante, ou seja, O(1), da linha 190 até a linha 199, o laço de repetição while será executado n vezes, como o custo dentro do laço é constante(O(1)), então a complexidade fica O(n). Da linha 201 até a 205 o custo do laço de repetição é O(n / 2), dessa mesma maneira da linha 207 até a linha 211. Com isso, se encerra a função “merge”. Fazendo a soma das complexidades obtidas na função, temos:

$$O(1) + O(n / 2) + O(n / 2) + O(1) + O(n) + O(n / 2) + O(n / 2) = O(n)$$

Na soma da análise de complexidade, o resultado é o termo de maior complexidade, que nesse caso é o O(n), ou seja, a complexidade da função “merge” é O(n).

Analisando agora a função principal do MergeSort, a “mergeSort”, temos a atribuição da linha 216 que tem o custo constante(O(1)), logo após temos a chamada da função “mergeSort” recursivamente para a primeira metade do vetor, ou seja, o custo desta chamada recursiva é T(n / 2), na linha 219 temos a mesma chamada, só que agora para a metade final do vetor, com isso, o custo desta chamada recursiva é também T(n / 2). Na linha 220 temos a chamada da função

“merge”, em que sua complexidade foi analisada acima e temos que a mesma tem custo O(n). Desta forma, se encerra a função “mergeSort” com as seguintes complexidades:

$T(n / 2) + T(n / 2) + O(n) = 2T(n / 2) + O(n)$

Para o cálculo da complexidade acima é utilizado o teorema mestre simplificado, onde é verificado a diferença entre os dois valores seguintes, log de 2 na base 2 e 1, como log de 2 na base 2 tem como resultado 1, é possível observar que os dois valores comparados são iguais, com isso, a complexidade é O(n x log n).
Portanto, o MergeSort tem como complexidade de tempo **O(n log n)** e de espaço também O(n log n).

SelectionSort

A ordenação por seleção (do inglês, selection sort) 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 n - 1 elementos restantes, até os últimos dois elementos. O algoritmo é composto por dois laços, um laço externo e outro interno. O laço externo serve para controlar o índice inicial e o interno percorre todo o vetor. Na primeira iteração do laço externo o índice começa de 0 e a cada iteração ele soma uma unidade até o final do vetor e o laço mais interno percorre o vetor começando desse índice externo + 1 até o final do vetor.

Análise de complexidade do SelectionSort:

O SelectionSort usa apenas uma função para fazer a ordenação, com o nome de “selection_sort”. De início, na linha 228 tem-se um laço de repetição, um for que vai de 1 até n, ou seja, ele repete n vezes, dentro dele, na linha 230 tem uma atribuição, com o custo constante(O(1)) e logo após, na linha 231, tem-se outro for que vai de i+1 até n, ou seja, ele repete n-1 vezes, que de toda forma, podemos considerar n vezes para prosseguir com a análise, da linha 237 até a linha 241 o custo é constante, dessa maneira, se encerra a função “selection_sort” e temos os seguintes custos de complexidade da mesma:

$O(n) \times (O(1) + O(n) + O(1)) = O(n) \times O(n) = O(n^2)$

Portanto, a complexidade de tempo do SelectionSort é **$O(n^2)$** e sua complexidade de espaço é O(n).

RadixSort

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.
Na ciência da computação, radix sort é um algoritmo de ordenação que ordena inteiros processando dígitos individuais. Como os inteiros podem representar strings compostas de caracteres (como nomes ou datas) e pontos flutuantes especialmente formatados, radix sort não é limitado somente a inteiros.
O algoritmo radix sort 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.

Análise de Complexidade do RadixSort:

O RadixSort usa apenas uma função para realizar a ordenação, com o nome de “radixsort”. De início, na linha 248 tem-se apenas atribuições, com isso o custo é constante. Na linha 250 tem-se um laço de repetição que vai de 1 até n, ou seja ele repete n vezes, todas as instruções dentro do laço têm o custo O(1), com isso o custo desse laço é O(n). Na linha 258, tem-se outro laço de repetição, o while, ele roda k vezes, seja k a quantidade de vezes em que entrara no laço de repetição, ou seja, o número de algarismos da maior chave, com isso o while tem o custo O(k), dentro do while tem-se 5 laços de repetições(for) que todos eles são percorridos n vezes, ou seja, custo O(n), após isso, a função chega ao fim e temos as complexidades a serem calculadas:

$O(1) + O(n) + O(k) \times (O(n) + O(n) + O(n) + O(n) + O(n)) = O(1) + O(n) + O(k) \times (O(n)) = O(1) + O(n) + O(n \times k) = O(n \times k)$

Portanto, a complexidade de tempo do RadixSort é **$O(n \times k)$** e a complexidade de espaço é $O(n + s)$, sendo ‘k’ a quantidade de algarismos da maior chave e ‘s’ o tamanho do alfabeto.

Avaliação experimental

Foi realizada uma análise empírica considerando os três métodos de ordenação para instâncias de tamanho N variando de 100 a 1000000, onde foi possível perceber o comportamento de cada método e seu respectivo custo expresso em ciclos de máquina. Os testes foram executados nas seguintes circunstâncias:

- Máquina: Acer ASPIRE 5 2018
    - Intel® Core i7-8550U 1.8GHz with Turbo Boost up to 4.0GHz
    - NVIDIA® GeForce® MX130 with 2 GB VRAM
    - Memória 8GB DDR4
    - 1000GB HDD

- SO: Linux
    - Ubuntu 20.04 LTS

- Linguagem: C 17
- Compilador: GCC 9.4.0
- Geração de instâncias: através da função rand( )

```
for (int i = 1; i <= n; i++) {
    vGerador[i] = rand() % n + 1;
}
```

Gerador

Os dados detalhados podem ser encontrados na seguinte tabela: Amostras. Abaixo segue uma média aritmética dos resultados obtidos:

	N				
Métodos	100	1000	10000	100000	1000000
Merge	66,25	346,75	4127,95	48367	560145,5
Selection	63,55	2063,6	194648,95	19382466	1937767470
Radix	25,6	139,05	1762,85	21305,25	251504,35

Tabela Amostral

Com esses dados tabulados, foi possível gerar os respectivos gráficos onde é possível visualizar como o método de ordenação merge sort é ligeiramente mais custoso que o radix sort. Ao adicionar o selection sort na comparação, é possível notar no gráfico 2 que ele possui um custo de tempo extremamente superior aos outros métodos.

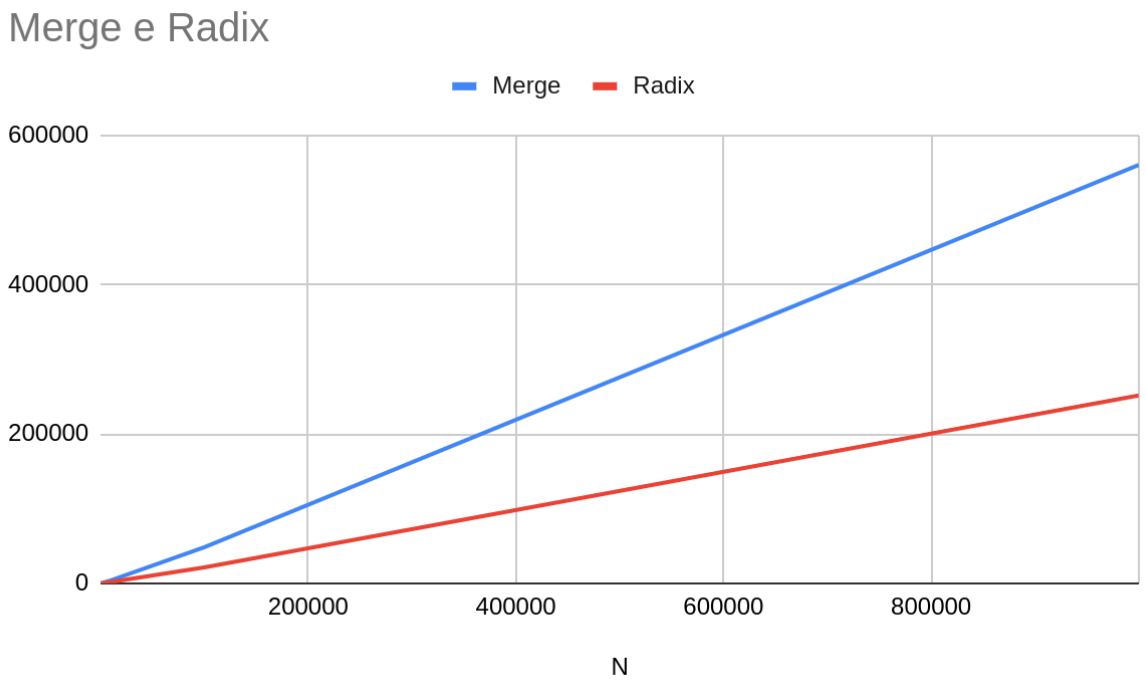

Gráfico 1

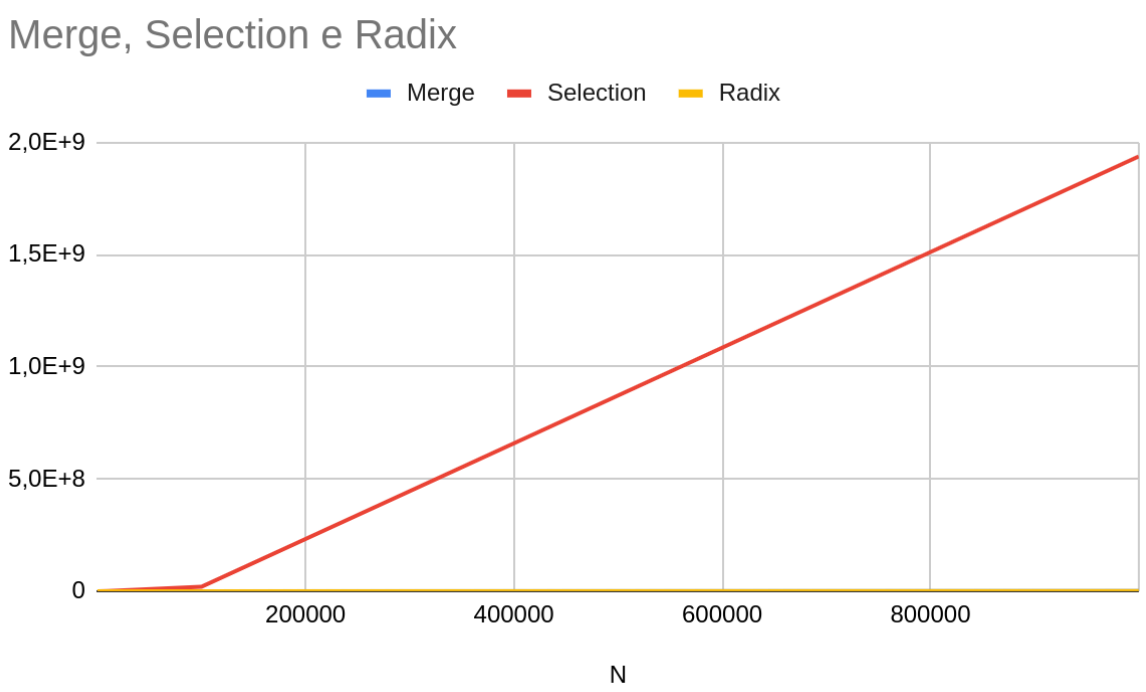

Gráfico 2

Após a coleta da amostra de tamanho 1000000, foi feita a tentativa de aumentar a instância para 2000000, todavia não havia memória o suficiente para alocação de memória, sendo assim o tamanho máximo de instâncias foi 1000000.

Métrica de avaliação

Para aferir o custo de cada método, foi contabilizado o tempo de execução. Para tal, utilizou-se a biblioteca times.h, mais especificamente sua função clock ( ). Com isso foi possível capturar os ciclos de máquina gastos através da diferença entre o horário de início e fim da ordenação. Para exemplificar, segue abaixo o trecho do código em que esse processo é feito:

```
tempo = clock();
mergeSort(v1, 1, n);
tempoGasto = clock() - tempo;
printf("%12lld\t", tempoGasto);
```

Imagem 2

Resultados Obtidos

Para avaliarmos a eficiência de cada método usando de parâmetro o tamanho das instâncias (100, 1.000, 10.000, 100.000 e 1.000.000), foi realizado o uso do Teste T pareado com confiança de 95%. Um Teste T pareado é sempre utilizado para calcular a diferença entre observações emparelhadas, o teste coleta seus dados amostrais de dois grupos e os resume no valor-t. Os cálculos comparam suas médias amostrais com a hipótese nula e incorporam o tamanho amostral e a variabilidade nos dados. Nesse caso, a variável de interesse é numérica e o objetivo é verificar se existe diferença significativa dessa variável entre dois grupos de interesse, onde a primeira informação será pareada com a segunda informação, com a terceira e assim por diante.

Ao encontrar o intervalo de 'T' entre dois algoritmos 'A' e 'B', se o valor der positivo, então o algoritmo 'A' é quem tem os maiores valores, caso contrário, o algoritmo 'B' é quem tem os maiores valores. Considerando que o teste é feito a fim de avaliar a melhor eficiência entre os métodos, então, o menor valor será o mais efetivo, já que gastou menos tempo.

Para N = 100:

Merge - Selection

$Média\ das\ diferenças\ =\ 2,7$

$Desvio\ padrão\ das\ diferenças\ =\ 12,46$

$T\ =\ 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras\ -\ 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n\ -\ 1\ =\ 19)$

$$Limite\ inferior\ =\ Média\ -\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$$

$$Limite\ superior\ =\ Média\ +\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$$

Dessa forma,

$$Linf\ =\ 2,7\ -\ 2,093\ *\ \frac{12,46}{\sqrt{20}}$$

$$Linf\ =\ 2,7\ -\ 2,093\ *\ 2,802$$

$$Linf\ =\ -\ 1,65$$

$$Lsup\ =\ 2,7\ +\ 2,093\ *\ \frac{12,46}{\sqrt{20}}$$

$$Lsup\ =\ 2,7\ +\ 2,093\ *\ 2,802$$

$$Lsup\ =\ 7,05$$

$$IC(95\%)\ =\ [-\ 1,65;\ 7,05]$$

Diante disso, considerando que o intervalo sai de um número negativo, passa por 0 e vai até um número positivo, a hipótese nula não pode ser desconsiderada. Com isso, pode-se concluir que não há diferença significativa entre o MergeSort e o SelectionSort sob essas condições impostas.

Merge - Radix

$$Média\ das\ diferenças\ =\ 40,65$$

$$Desvio\ padrão\ das\ diferenças\ =\ 14,37$$

$$T\ =\ 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras\ -\ 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n\ -\ 1\ =\ 19)$$

$$Limite\ inferior\ =\ Média\ -\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$$

$$Limite\ superior\ =\ Média\ +\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$$

Dessa forma,

$$Linf\ =\ 40,65\ -\ 2,093\ *\ \frac{14,37}{\sqrt{20}}$$

$$Linf\ =\ 40,65\ -\ 2,093\ *\ 3,21$$

$$Linf\ =\ 33,93$$

$$Lsup\ =\ 40,65\ +\ 2,093\ *\ \frac{14,37}{\sqrt{20}}$$

$$Lsup\ =\ 40,65\ +\ 2,093\ *\ 3,21$$

$$Lsup\ =\ 47,36$$

$$IC(95\%)\ =\ [33,93;\ 47,36]$$

Diante disso, considerando que o intervalo já começa de um número positivo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o MergeSort e o RadixSort sob essas condições impostas, com o Radix sendo ligeiramente superior em questão de desempenho.

Selection - Radix

$Média\ das\ diferenças\ =\ 38,45$

$Desvio\ padrão\ das\ diferenças\ =\ 8,10$

$T\ =\ 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras\ -\ 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n\ -\ 1\ =\ 19)$

$Limite\ inferior\ =\ Média\ -\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$

$Limite\ superior\ =\ Média\ +\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$

Dessa forma,

$Linf\ =\ 38,45\ -\ 2,093\ *\ \frac{8,10}{\sqrt{20}}$

$Linf\ =\ 38,45\ -\ 2,093\ *\ 1,812$

$Linf\ =\ 34,65$

$Lsup\ =\ 38,45\ +\ 2,093\ *\ \frac{8,10}{\sqrt{20}}$

$Lsup\ =\ 38,45\ +\ 2,093\ *\ 1,812$

$Lsup\ =\ 42,24$

$IC(95\%)\ =\ [34,65;\ 42,24]$

Diante disso, considerando que o intervalo já começa de um número positivo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o SelectionSort e o RadixSort sob essas condições impostas, com o Radix sendo ligeiramente superior em questão de desempenho.

Para N = 1.000:

Merge - Selection

$Média\ das\ diferenças\ =\ -\ 1.716,85$

$Desvio\ padrão\ das\ diferenças\ =\ 41,13$

$T\ =\ 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras\ -\ 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n\ -\ 1\ =\ 19)$

$Limite\ inferior\ =\ Média\ -\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$

$Limite\ superior\ =\ Média\ +\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$

Dessa forma,

$Linf\ =\ -\ 1.716,85\ -\ 2,093\ *\ \frac{41,13}{\sqrt{20}}$

$Linf\ =\ -\ 1.716,85\ -\ 2,093\ *\ 9,20$

$Linf\ =\ -\ 1.736,10$

$Lsup\ =\ -\ 1.716,85\ +\ 2,093\ *\ \frac{41,13}{\sqrt{20}}$

$Lsup\ =\ -\ 1.716,85\ +\ 2,093\ *\ 9,20$

$Lsup\ =\ -\ 1.697,59$

$IC(95\%) = [-1.736,10; \ -1.697,59]$

Diante disso, considerando que o intervalo começa e acaba em um número negativo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o MergeSort e o SelectionSort sob essas condições impostas, com o Merge sendo ligeiramente superior em questão de desempenho.

Merge - Radix

$Média\ das\ diferenças = 207,7$
$Desvio\ padrão\ das\ diferenças = 28,72$
$T = 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras - 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n - 1 = 19)$

$Limite\ inferior = Média - t * (\frac{desvio\ padrão}{\sqrt{n}})$
$Limite\ superior = Média + t * (\frac{desvio\ padrão}{\sqrt{n}})$

Dessa forma,

$Linf = 207,7 - 2,093 * \frac{28,72}{\sqrt{20}}$
$Linf = 207,7 - 2,093 * 6,42$
$Linf = 194,26$
$Lsup = 207,7 + 2,093 * \frac{28,72}{\sqrt{20}}$
$Lsup = 207,7 + 2,093 * 6,42$
$Lsup = 221,13$
$IC(95\%) = [194,26;\ 221,13]$

Diante disso, considerando que o intervalo já começa de um número positivo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o MergeSort e o RadixSort sob essas condições impostas, com o Radix sendo ligeiramente superior em questão de desempenho.

Selection - Radix

$Média\ das\ diferenças = 1.924,55$
$Desvio\ padrão\ das\ diferenças = 36,07$
$T = 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras - 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n - 1 = 19)$

$Limite\ inferior = Média - t * (\frac{desvio\ padrão}{\sqrt{n}})$
$Limite\ superior = Média + t * (\frac{desvio\ padrão}{\sqrt{n}})$

Dessa forma,

$Linf = 1.924,55 - 2,093 * \frac{36,07}{\sqrt{20}}$

$$Linf = 1.924,55 - 2,093 * 8,07$$
$$Linf = 1.907,66$$
$$Lsup = 1.924,55 + 2,093 * \frac{36,07}{\sqrt{20}}$$
$$Lsup = 1.924,55 + 2,093 * 8,07$$
$$Lsup = 1.941,44$$
$$IC(95\%) = [1.907,66;\ 1.941,44]$$

Diante disso, considerando que o intervalo já começa de um número positivo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o SelectionSort e o RadixSort sob essas condições impostas, com o Radix sendo ligeiramente superior em questão de desempenho.

Para N = 10.000:

Merge - Selection

$$Média\ das\ diferenças = -190.521$$
$$Desvio\ padrão\ das\ diferenças = 214,61$$
$$T = 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras - 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n - 1 = 19)$$
$$Limite\ inferior = Média - t * (\frac{desvio\ padrão}{\sqrt{n}})$$
$$Limite\ superior = Média + t * (\frac{desvio\ padrão}{\sqrt{n}})$$

Dessa forma,

$$Linf = -190.521 - 2,093 * \frac{214,61}{\sqrt{20}}$$
$$Linf = -190.521 - 2,093 * 48,01$$
$$Linf = -190.621,48$$
$$Lsup = -190.521 + 2,093 * \frac{214,61}{\sqrt{20}}$$
$$Lsup = -190.521 + 2,093 * 48,01$$
$$Lsup = -190.420,51$$
$$IC(95\%) = [-190.621,48;\ -190.420,51]$$

Diante disso, considerando que o intervalo começa e acaba em um número negativo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o MergeSort e o SelectionSort sob essas condições impostas, com o Merge sendo ligeiramente superior em questão de desempenho.

Merge - Radix

$$Média\ das\ diferenças = 2.365,1$$
$$Desvio\ padrão\ das\ diferenças = 161,69$$
$$T = 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras - 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n - 1 = 19)$$

$$Limite\ inferior\ =\ Média\ -\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$$

$$Limite\ superior\ =\ Média\ +\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$$

Dessa forma,

$$Linf\ =\ 2.365,1\ -\ 2,093\ *\ \frac{161,69}{\sqrt{20}}$$

$$Linf\ =\ 2.365,1\ -\ 2,093\ *\ 36,17$$

$$Linf\ =\ 2.289,39$$

$$Lsup\ =\ 2.365,1\ +\ 2,093\ *\ \frac{161,69}{\sqrt{20}}$$

$$Lsup\ =\ 2.365,1\ +\ 2,093\ *\ 36,17$$

$$Lsup\ =\ 2.440,80$$

$$IC(95\%)\ =\ [2.289,39;\ 2.440,80]$$

Diante disso, considerando que o intervalo já começa de um número positivo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o MergeSort e o RadixSort sob essas condições impostas, com o Radix sendo ligeiramente superior em questão de desempenho.

Selection - Radix

$$Média\ das\ diferenças\ =\ 192.886,1$$

$$Desvio\ padrão\ das\ diferenças\ =\ 258,15$$

$$T\ =\ 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras\ -\ 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n\ -\ 1\ =\ 19)$$

$$Limite\ inferior\ =\ Média\ -\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$$

$$Limite\ superior\ =\ Média\ +\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$$

Dessa forma,

$$Linf\ =\ 192.886,1\ -\ 2,093\ *\ \frac{258,15}{\sqrt{20}}$$

$$Linf\ =\ 192.886,1\ -\ 2,093\ *\ 57,75$$

$$Linf\ =\ 192.765,23$$

$$Lsup\ =\ 192.886,1\ +\ 2,093\ *\ \frac{258,15}{\sqrt{20}}$$

$$Lsup\ =\ 192.886,1\ +\ 2,093\ *\ 57,75$$

$$Lsup\ =\ 193.006,97$$

$$IC(95\%)\ =\ [192.765,23;\ 193.006,97]$$

Diante disso, considerando que o intervalo já começa de um número positivo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o SelectionSort e o RadixSort sob essas condições impostas, com o Radix sendo ligeiramente superior em questão de desempenho.

Para N = 100.000:

Merge - Selection

$Média\ das\ diferenças\ =\ -19.334.009$

$Desvio\ padrão\ das\ diferenças\ =\ 23.894,4$

$T\ =\ 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras\ -\ 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n\ -\ 1\ =\ 19)$

$Limite\ inferior\ =\ Média\ -\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$

$Limite\ superior\ =\ Média\ +\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$

Dessa forma,

$Linf\ =\ -19.334.009\ -\ 2,093\ *\ \frac{23.894,4}{\sqrt{20}}$

$Linf\ =\ -19.334.009\ -\ 2,093\ *\ 5.345,5$

$Linf\ =\ -19.345.197,13$

$Lsup\ =\ -19.334.009\ +\ 2,093\ *\ \frac{23.894,4}{\sqrt{20}}$

$Lsup\ =\ -19.334.009\ +\ 2,093\ *\ 5.345,5$

$Lsup\ =\ -19.322.820,87$

$IC(95\%)\ =\ [-19.345.197,13;\ -19.322.820,87]$

Diante disso, considerando que o intervalo começa e acaba em um número negativo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o MergeSort e o SelectionSort sob essas condições impostas, com o Merge sendo ligeiramente superior em questão de desempenho.

Merge - Radix

$Média\ das\ diferenças\ =\ 27.061,75$

$Desvio\ padrão\ das\ diferenças\ =\ 1.651,13$

$T\ =\ 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras\ -\ 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n\ -\ 1\ =\ 19)$

$Limite\ inferior\ =\ Média\ -\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$

$Limite\ superior\ =\ Média\ +\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$

Dessa forma,

$Linf\ =\ 27.061,75\ -\ 2,093\ *\ \frac{1.651,13}{\sqrt{20}}$

$Linf\ =\ 27.061,75\ -\ 2,093\ *\ 369,29$

$Linf\ =\ 26.288,82$

$Lsup\ =\ 27.061,75\ +\ 2,093\ *\ \frac{1.651,13}{\sqrt{20}}$

$Lsup\ =\ 27.061,75\ +\ 2,093\ *\ 369,29$

$Lsup = 27.834,67$
$IC(95\%) = [26.288,82;\ 27.834,67]$

Diante disso, considerando que o intervalo já começa de um número positivo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o MergeSort e o RadixSort sob essas condições impostas, com o Radix sendo ligeiramente superior em questão de desempenho.

Selection - Radix
$Média\ das\ diferenças = 19.361.160,75$
$Desvio\ padrão\ das\ diferenças = 23.768,37$
$T = 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras - 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n - 1 = 19)$

$Limite\ inferior = Média - t * (\frac{desvio\ padrão}{\sqrt{n}})$
$Limite\ superior = Média + t * (\frac{desvio\ padrão}{\sqrt{n}})$
Dessa forma,
$Linf = 19.361.160,75 - 2,093 * \frac{23.768,37}{\sqrt{20}}$
$Linf = 19.361.160,75 - 2,093 * 5.317,31$
$Linf = 19.350.031,62$
$Lsup = 19.361.160,75 + 2,093 * \frac{23.768,37}{\sqrt{20}}$
$Lsup = 19.361.160,75 + 2,093 * 5.317,31$
$Lsup = 19.372.289,88$
$IC(95\%) = [19.350.031,62;\ 19.372.289,88]$

Diante disso, considerando que o intervalo já começa de um número positivo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o SelectionSort e o RadixSort sob essas condições impostas, com o Radix sendo ligeiramente superior em questão de desempenho.

Para N = 1.000.000:

Merge - Selection
$Média\ das\ diferenças = -1.937.207.324,35$
$Desvio\ padrão\ das\ diferenças = 423.537,06$
$T = 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras - 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n - 1 = 19)$

$Limite\ inferior = Média - t * (\frac{desvio\ padrão}{\sqrt{n}})$

$Limite\ superior\ =\ Média\ +\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$

Dessa forma,

$Linf\ =\ -\ 1.937.207.324,35\ -\ 2,093\ *\ \frac{423.537,06}{\sqrt{20}}$

$Linf\ =\ -\ 1.937.207.324,35\ -\ 2,093\ *\ 94.751,02$

$Linf\ =\ -\ 1.937.405.638,23$

$Lsup\ =\ -\ 1.937.207.324,35\ +\ 2,093\ *\ \frac{423.537,06}{\sqrt{20}}$

$Lsup\ =\ -\ 1.937.207.324,35\ +\ 2,093\ *\ 94.751,02$

$Lsup\ =\ -\ 1.937.009.010,46$

$IC(95\%)\ =\ [-\ 1.937.405.638,23;\ -\ 1.937.009.010,46]$

Diante disso, considerando que o intervalo começa e acaba em um número negativo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o MergeSort e o SelectionSort sob essas condições impostas, com o Merge sendo ligeiramente superior em questão de desempenho.

Merge - Radix

$Média\ das\ diferenças\ =\ 308.641,15$

$Desvio\ padrão\ das\ diferenças\ =\ 16.286,87$

$T\ =\ 2,093\ (basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras\ -\ 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n\ -\ 1\ =\ 19)$

$Limite\ inferior\ =\ Média\ -\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$

$Limite\ superior\ =\ Média\ +\ t\ *\ (\frac{desvio\ padrão}{\sqrt{n}})$

Dessa forma,

$Linf\ =\ 308.641,15\ -\ 2,093\ *\ \frac{16.286,87}{\sqrt{20}}$

$Linf\ =\ 308.641,15\ -\ 2,093\ *\ 3.643,59$

$Linf\ =\ 301.015,11$

$Lsup\ =\ 308.641,15\ +\ 2,093\ *\ \frac{16.286,87}{\sqrt{20}}$

$Lsup\ =\ 308.641,15\ +\ 2,093\ *\ 3.643,59$

$Lsup\ =\ 316.267,18$

$IC(95\%)\ =\ [301.015,11;\ 316.267,18]$

Diante disso, considerando que o intervalo já começa de um número positivo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o MergeSort e o RadixSort sob essas condições impostas, com o Radix sendo ligeiramente superior em questão de desempenho.

Selection - Radix

$Média\ das\ diferenças\ =\ 1.937.515.965,5$

$Desvio\ padrão\ das\ diferenças\ =\ 425.078,58$

$T = 2,093$ ($basta\ analisar\ a\ tabela\ para\ 95\%\ de\ confiança\ e\ considerar\ o\ número\ amostras - 1,\ como\ grau\ de\ liberdade,\ que\ seria\ n - 1 = 19$)

$$Limite\ inferior = Média - t * (\frac{desvio\ padrão}{\sqrt{n}})$$

$$Limite\ superior = Média + t * (\frac{desvio\ padrão}{\sqrt{n}})$$

Dessa forma,

$$Linf = 1.937.515.965,5 - 2,093 * \frac{425.078,58}{\sqrt{20}}$$

$$Linf = 1.937.515.965,5 - 2,093 * 95.095,75$$

$$Linf = 1.937.316.930,09$$

$$Lsup = 1.937.515.965,5 + 2,093 * \frac{425.078,58}{\sqrt{20}}$$

$$Lsup = 1.937.515.965,5 + 2,093 * 95.095,75$$

$$Lsup = 1.937.715.000,9$$

$$IC(95\%) = [1.937.316.930,09;\ 1.937.715.000,9]$$

Diante disso, considerando que o intervalo já começa de um número positivo, a hipótese nula deve ser desconsiderada. Com isso, pode-se concluir que há diferença significativa entre o SelectionSort e o RadixSort sob essas condições impostas, com o Radix sendo ligeiramente superior em questão de desempenho.

Diante de todos esses resultados experimentais, é perceptível que o Radix Sort é o algoritmo mais eficiente, para o que se propõe, entre os 3 algoritmos analisados.

Conclusão

Neste trabalho foi abordado o assunto da avaliação empírica de três diferentes algoritmos de ordenação, desta forma, pode-se concluir que os experimentos comprovaram o maior objetivo que era achar o algoritmo mais eficiente dentre os três abordados. Diante de todas as avaliações feitas - como as análises de complexidade, testes estatísticos, além da execução dos algoritmos -, elas convergiram de forma a apontar o Radix Sort como o melhor método de ordenação entre os analisados.

Por meio dos testes estatísticos analisando todas as instâncias com todos os diferentes tamanhos, utilizando do teste t pareado com 95% de confiança, o Radix foi superior em cada um dos experimentos. Através das análises de complexidade de tempo de cada algoritmo em individual foi possível obter seu custo, onde o radix também foi superior aos outros, tendo uma complexidade menor de tempo, isso pode ser observado nos tempos gastos na ordenação de vetores iguais em cada método, onde o radix faz a ordenação mais rápida, depois vem o mergesort e por último o selectionsort, que tem o custo de complexidade de tempo maior.

Este trabalho foi de suma importância para nosso aprendizado, uma vez que, nos permitiu compreender ainda mais sobre a análise de complexidade de algoritmos e testes estatísticos que evidenciam os resultados obtidos através das amostras utilizadas e dos critérios para a avaliação dos testes.

Referências Bibliográficas

http://www.learningaboutelectronics.com/Artigos/Calculadora-de-teste-T-para-amostras-dependentes.php

https://blog.minitab.com/pt/nocoes-basicas-sobre-testes-t-para-uma-amostra-para-duas-amostras-e-pareados#:~:text=Um%20teste-t%20pareado%20simplesmente,1%20amostra%20sobre%20as%20diferenças

https://statplace.com.br/blog/testes-estatisticos-para-amostras-pareadas/

https://pt.wikipedia.org/wiki/Merge_sort

https://pt.wikipedia.org/wiki/Selection_sort

https://pt.wikipedia.org/wiki/Radix_sort

http://wurthmann.blogspot.com/2015/04/medir-tempo-de-execucao-em-c.html