



Universidade do Vale do Itajaí – UNIVALI
Escola do Mar, Ciência e Tecnologia – EMCT
Ciência da Computação – Estruturas de Dados – Prof. Eduardo Alves da Silva

Atividade de Implementação 3- Comparativo SORT BENCHMARK

Acadêmica: Pamela Bandeira Gerber

29/06/21

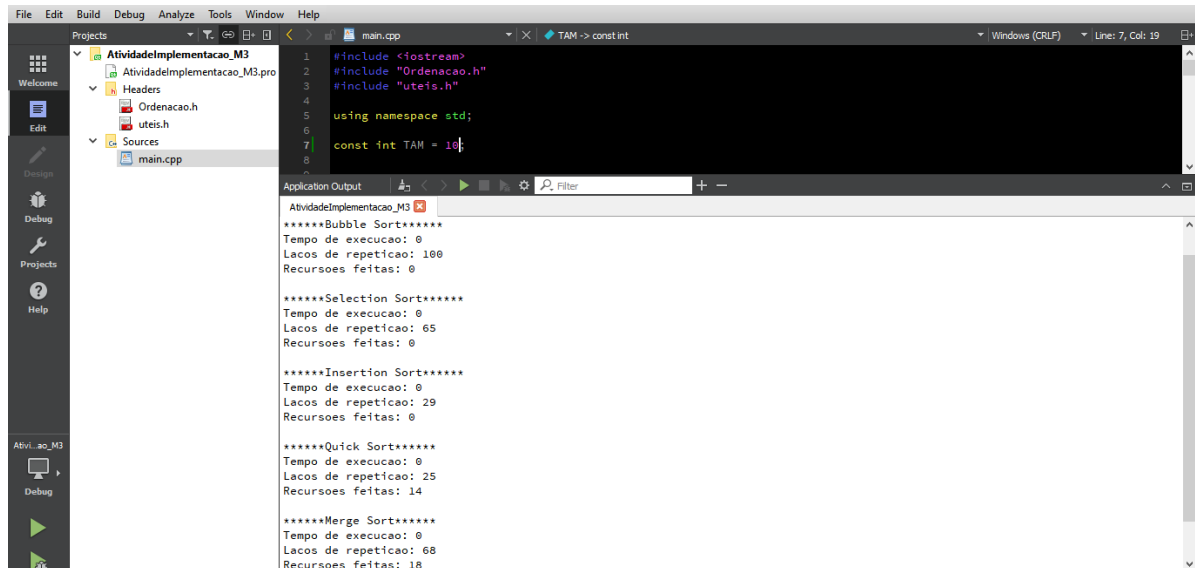
Introdução

O presente trabalho é para implementar os algoritmos de ordenação que são: Bubble Sort, Selection Sort, Insertion Sort, Quick Sort e Merge Sort.

Analisaremos e compararemos os resultados obtidos na execução, feitas sob diferentes cenários como os de pequenos volumes, médios volumes, grandes volumes e volumes massivos de dados, assim ordenando vetores.

Desenvolvimento

Abaixo teremos prints com análises e correlacionando cada algoritmo com os cenários.



The screenshot shows a C++ IDE with a project named 'AtividadeImplementacao_M3'. The source file 'main.cpp' contains the following code:

```
1 #include <iostream>
2 #include "Ordenacao.h"
3 #include "uteis.h"
4
5 using namespace std;
6
7 const int TAM = 10;
```

The 'Application Output' window displays the results of the execution for five sorting algorithms:

```
AtividadeImplementacao_M3
*****Bubble Sort*****
Tempo de execucao: 0
Lacos de repeticao: 100
Recursos feitos: 0

*****Selection Sort*****
Tempo de execucao: 0
Lacos de repeticao: 65
Recursos feitos: 0

*****Insertion Sort*****
Tempo de execucao: 0
Lacos de repeticao: 29
Recursos feitos: 0

*****Quick Sort*****
Tempo de execucao: 0
Lacos de repeticao: 25
Recursos feitos: 14

*****Merge Sort*****
Tempo de execucao: 0
Lacos de repeticao: 68
Recursos feitos: 18
```

Na lista de 10 elementos, nenhum dos algoritmos conseguiram pegar tempo de execução por ter sido muito rápido a aplicação não obtendo sucesso. O algoritmo Bubble Sort tem 100 laços de repetição. No algoritmo Selection Sort houve uma redução de 35% menos na execução de laços.

No algoritmo Insertion Sort terá um aumento de 55,38% na utilização de laços comparado com o Selection Sort. No algoritmo Quick Sort terá uma queda de 13,79% na execução de laços em comparação ao Insertion Sort e para finalizar, o algoritmo Merge Sort terá 63,235% na taxa de laços de repetição em comparação ao Quick Sort.

Analisando a taxa de recursividade do algoritmo Merge Sort de 18, o algoritmo de ordenação Quick Sort terá 22,222% na ocorrência de recursão. Entretanto, já que o algoritmo do Bubble, Selection e Insertion não obtém valor de recursões feitas, houve comparações com o algoritmo de ordenação Merge Sort.

```
1 #include <iostream>
2 #include "Ordenacao.h"
3 #include "uteis.h"
4
5 using namespace std;
6
7 const int TAM = 100;
```

AtividadeImplementacao_M3

*****Bubble Sort*****

Tempo de execucao: 0

Lacos de repeticao: 10000

Recursoes feitas: 0

*****Selection Sort*****

Tempo de execucao: 0

Lacos de repeticao: 5150

Recursoes feitas: 0

*****Insertion Sort*****

Tempo de execucao: 0

Lacos de repeticao: 1729

Recursoes feitas: 0

*****Quick Sort*****

Tempo de execucao: 0

Lacos de repeticao: 648

Recursoes feitas: 144

*****Merge Sort*****

Tempo de execucao: 0

Lacos de repeticao: 1344

Recursoes feitas: 198

Na lista de 100 elementos, nenhum dos algoritmos conseguiram pegar tempo de execução por ter sido muito rápido a aplicação não obtivendo sucesso. O algoritmo Bubble Sort tem 10000 laços de repetição. No algoritmo Selection Sort houve uma redução de 48,5% menos na execução de laços.

No algoritmo Insertion Sort terá um aumento de 66,427% na utilização de laços comparado com o Selection Sort. No algoritmo Quick Sort terá uma redução de 62,521% na execução de laços comparado com Insertion Sort e para finalizar, o algoritmo Merge Sort terá 51,785% na taxa de laços de repetição em comparação ao Quick Sort.

Analisando a taxa de recursividade do algoritmo Merge Sort tem 144, o algoritmo de ordenação Quick Sort terá 27,272% na ocorrência de recursão. Entretanto, já que o algoritmo do Bubble, Selection e Insertion não obtém valor de recursões feitas, houve comparações com o algoritmo de ordenação Merge Sort.

```
1 #include <iostream>
2 #include "Ordenacao.h"
3 #include "uteis.h"
4
5 using namespace std;
6
7 const int TAM = 1000;
```

AtividadeImplementacao_M3

*****Bubble Sort*****

Tempo de execucao: 0.006

Lacos de repeticao: 1000000

Recursoes feitas: 0

*****Selection Sort*****

Tempo de execucao: 0.002

Lacos de repeticao: 501500

Recursoes feitas: 0

*****Insertion Sort*****

Tempo de execucao: 0.001

Lacos de repeticao: 66584

Recursoes feitas: 0

*****Quick Sort*****

Tempo de execucao: 0

Lacos de repeticao: 10134

Recursoes feitas: 1362

*****Merge Sort*****

Tempo de execucao: 0

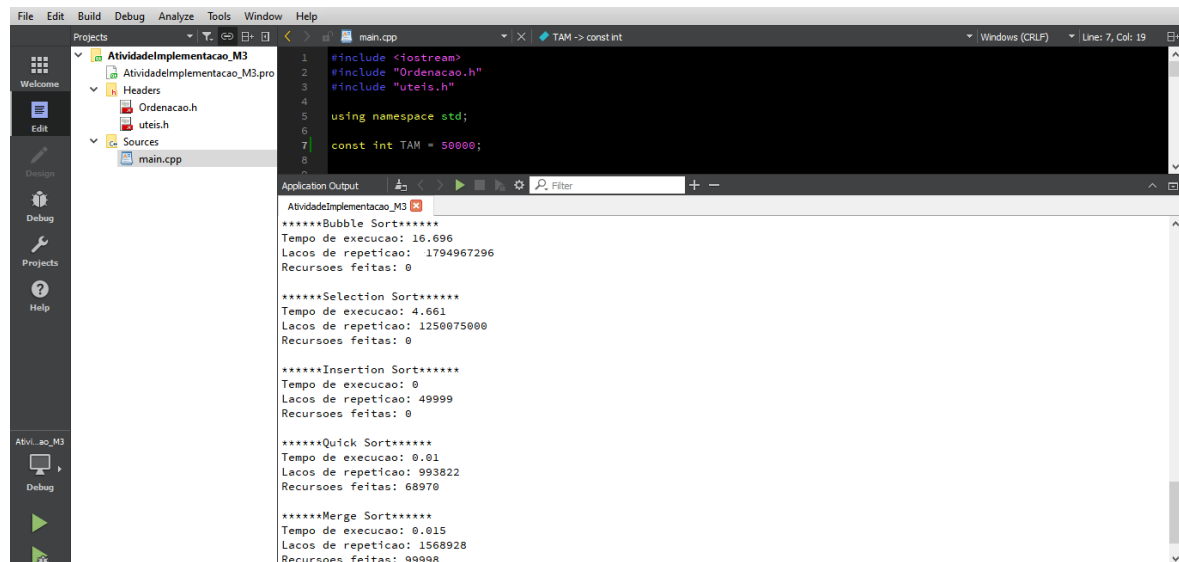
Lacos de repeticao: 19952

Recursoes feitas: 1998

Na lista de 1000 elementos, o algoritmo Bubble Sort tem-se os dados 0,006 segundos de tempo executado e 1000000 que foram executados. No algoritmo Selection Sort teve uma redução de 66,666% no tempo de execução e 49,85% menos laços comparados ao algoritmo anterior. No algoritmo Insertion Sort terá uma queda de 50% de tempo executado e 86,723% na utilização de laços em comparação ao algoritmo anterior.

No algoritmo Quick Sort teremos 100% do tempo executado e 84,780% na execução de laços em comparação com Insertion Sort e para finalizar, o algoritmo Merge Sort terá 0% do tempo executado pelo fato da execução ter sido muito rápido e não conseguir pegar a aplicação e 96,881% na taxa de laços de repetição em comparação ao Quick Sort.

Analisando a taxa de recursividade do algoritmo de ordenação Quick Sort tem 1362, o algoritmo Merge Sort terá 46,696% na ocorrência de recursão. Entretanto, já que o algoritmo do Bubble, Selection e Insertion não obtém valor de recursões feitas, houve comparações com o algoritmo de ordenação Quick Sort.



```
1 #include <iostream>
2 #include "Ordenacao.h"
3 #include "uteis.h"
4
5 using namespace std;
6
7 const int TAM = 50000;
8
```

Application Output

```
*****Bubble Sort*****
Tempo de execucao: 16.696
Lacos de repeticao: 1794967296
Recursos feitas: 0

*****Selection Sort*****
Tempo de execucao: 4.661
Lacos de repeticao: 1250075000
Recursos feitas: 0

*****Insertion Sort*****
Tempo de execucao: 0
Lacos de repeticao: 49999
Recursos feitas: 0

*****Quick Sort*****
Tempo de execucao: 0.01
Lacos de repeticao: 993822
Recursos feitas: 68970

*****Merge Sort*****
Tempo de execucao: 0.015
Lacos de repeticao: 1568928
Recursos feitas: 99998
```

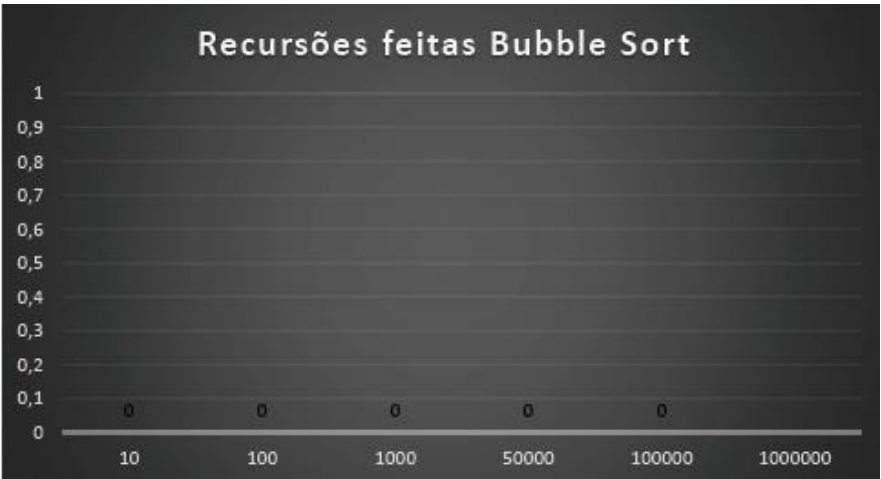
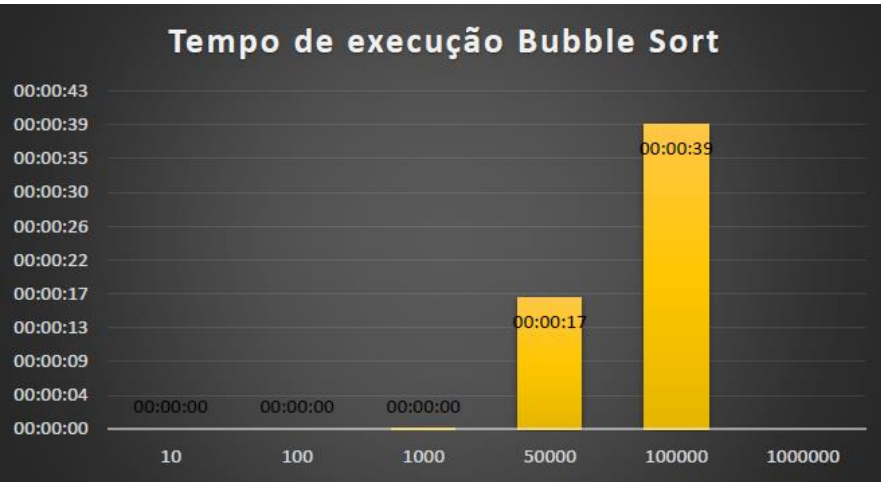
Na lista de 50000 elementos, o algoritmo Bubble Sort tem-se os dados 16.696 segundos de tempo executado e 1794967296 de laços que foram executados. No algoritmo Selection Sort teve uma redução de 72,083% no tempo de execução e 30,356% menos laços repetidos em comparação ao algoritmo anterior. No algoritmo Insertion Sort terá 100% de tempo executado, e terá uma redução de 99,996% na utilização de laços em comparação ao algoritmo anterior.

No algoritmo Quick Sort não conseguiu pegar tempo de execução por ter sido muito rápido a aplicação e 94,969% na execução de laços, em comparação ao Insertion Sort e para finalizar, o algoritmo Merge Sort terá 50% de redução no tempo de execução e 57,868% na taxa de laços de repetição em comparação ao Quick Sort.

Analisando a taxa de recursividade do algoritmo de ordenação Quick Sort tem 68970, o algoritmo Merge Sort terá 44,987% na ocorrência de recursão. Entretanto, já que o algoritmo do Bubble, Selection e Insertion não obtém valor de recursões feitas, houve comparações com o algoritmo de ordenação Quick Sort.

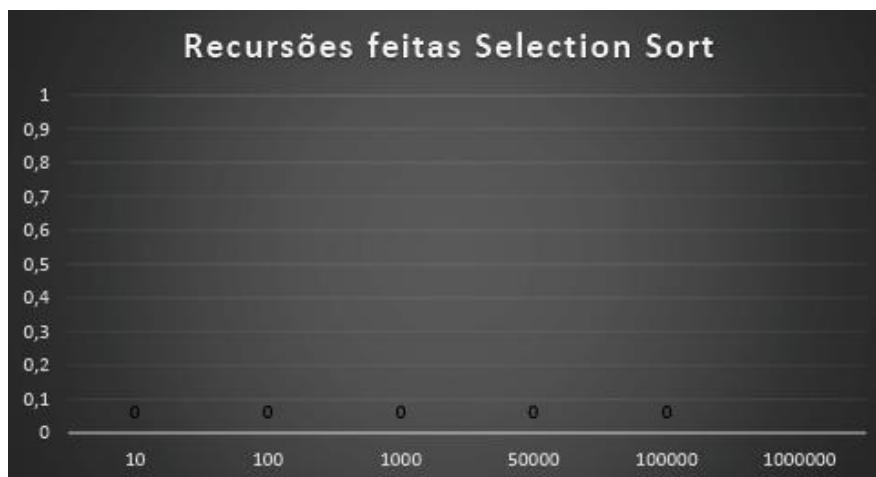
Logo abaixo teremos gráficos de evolução comportamental de cada algoritmo de ordenação com os dados dos cenários obtidos.

Bubble Sort:



É o mais simples dos algoritmos de ordenação, porém é um dos mais ineficientes para listas muito grandes percorrendo quantas vezes for necessária, passando de um segundo de tempo de execução na maioria dos casos, chegando até a mais de dezesseis segundos de tempo de execução. O bubble sort tem como vantagem a simplicidade do código, podendo ser facilmente alterado conforme a vontade do usuário.

Selection Sort:



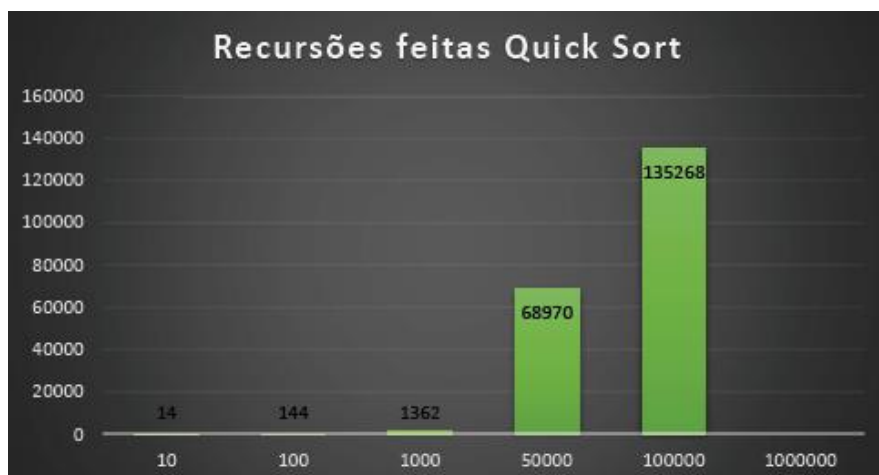
Esse algoritmo teve um desempenho quase igual do bubble, sendo o segundo pior, as operações básicas foram parecidas para os dois testes em cada caso, porém muito lento em vetores muito grandes (como foram testados no trabalho).

Insertion Sort:



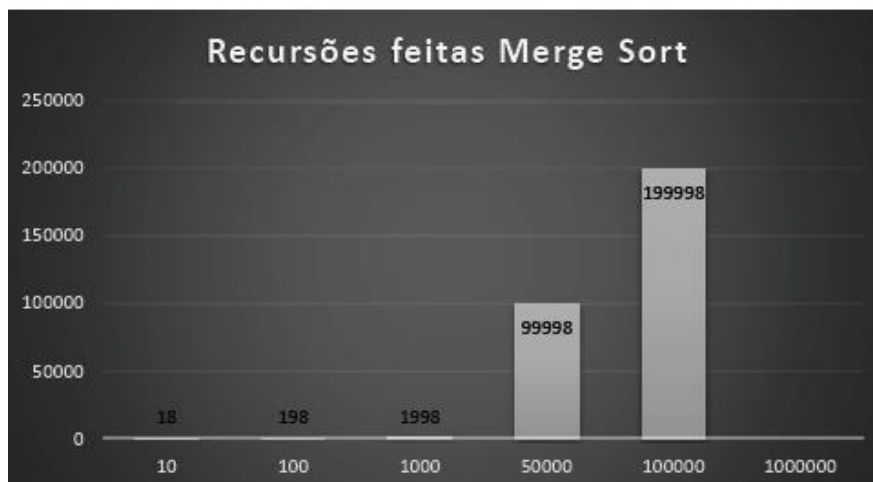
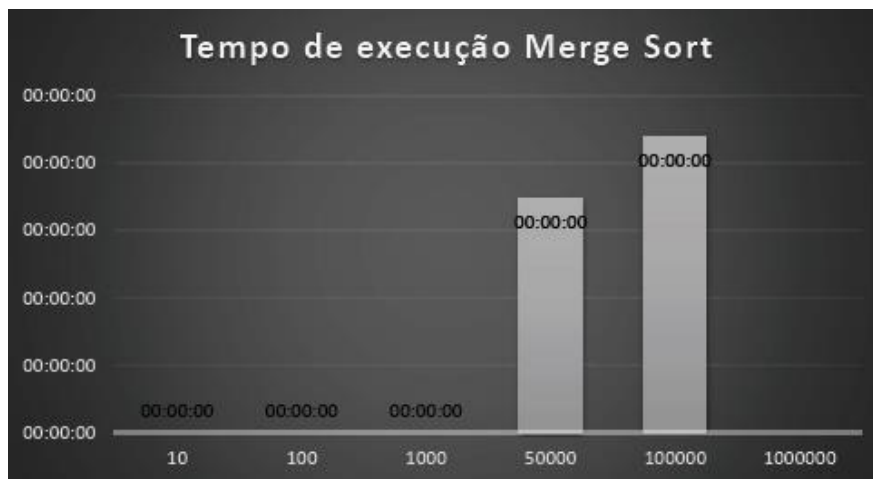
Esse algoritmo teve uma eficiência melhorada quando aplicado em pequenas listas em comparação com o bubble e selection em relação tanto ao tempo quanto aos laços de repetição que foram menores, além de ser simples sua codificação. Em geral, o insertion sort foi mais rápido que todos os anteriores.

Quick Sort:



O tempo de execução foi extremamente rápido em todos os casos, não passando de um segundo em nenhum caso. Esse algoritmo é muito rápido e também relativamente complexo no código, só deve-se tomar cuidado com casos específicos. Nos pequenos elementos na lista ele já foi ordenado, nos grandes elementos será dividido e recursivamente aconteceria o mesmo processo de escolha de um pivô.

Merge Sort:



Foi extremamente rápido em todos os casos, não passando de um segundo de tempo de execução em nenhum caso. É um algoritmo sem muita variação, podendo ser usado em diversos casos sem se preocupar com casos específicos, podendo ser iterativas ou recursivas, porém é um código relativamente complexo.

As estratégias utilizadas para execução dos algoritmos que estão na Ordenação.h terá o primeiro parâmetro sendo o tamanho que vai ser genérico para todos os tipos de algoritmos e o segundo parâmetro é sobre coleta de dados para saber quantas vezes a função é chamada ou repetição de laço, para assim analisar a quantidade de tempo, quantidade de laço e recursões feitas.

No algoritmo de ordenação Bubble Sort dentro do laço *for*, teremos laço para repetir o processo para garantir a ordenação, o 1 serve para fazer a troca em si, fazendo a troca de toda, a lista volta para o laço e faz a troca da lista de novo se for a mais interna.

No algoritmo de ordenação Selection Sort, foi passado a lista com parâmetro e a variável para armazenar os dados para análise (coleta), o laço *for* vai percorrer, fazendo o índice e o segundo laço *for* vai percorrer o restante da lista a partir do índice para procurar o menor, depois de achar o menor, ele faz a troca e acaba. O `dados[1].vezesLaco++`; é para pegar a quantidade de laços que foram executados para fazer a análise de dados.

No algoritmo de ordenação Insertion Sort, esta procura na lista o menor elemento e joga para as primeiras posições.

O particionar é um algoritmo que vai receber o laço *for* que vai percorrer a lista e fazer esse quesito de jogar para o lado esquerdo e lado direito, vai retornar o índice do pivô, como por exemplo o meio da lista dessa nova lista. Poderia ter feito tudo no mesmo algoritmo e função do Quick Sort, mas para ter um melhor entendimento, foi separado.

No algoritmo de ordenação Quick Sort divide a lista em duas, joga tudo que é menor que o pivô que escolheu para a esquerda e tudo que for maior que o pivô para a direita.

No algoritmo de ordenação Merge Sort, segue o mesmo princípio que o do algoritmo Quick Sort, apenas com a diferença de que gasta mais memória e com o conceito de conquistar, a primeira coisa que ele faz é dividir/separar a lista toda em listas menores, a primeira rodada ele vai pegar e dividir toda a lista em duas listas e pegar essas duas listas e dividir em quatro e assim por diante até ter apenas um elemento na lista. Após chegar nessa lista, vai começar a voltar, só que quando fizer isso, vai juntar a lista já ordenando ela.

Conclusão

Neste trabalho foi cumprido todos os objetivos que haviam proposto, observando os comportamentos de todos os algoritmos. O que se mostrou mais ineficiente foi o Bubble Sort, mesmo com sua vantagem de ter o código simples e ser facilmente alterado, tem um tempo de execução muito longo. O Selection Sort consegue ter uma execução mais rápida que o Bubble Sort. Sendo mais rápido que todos os algoritmos acima citados, o Insertion Sort teve um comportamento muito parecido nos três tipos de vetores ordenados, além de conseguir ter uma execução mais rápida que o Bubble Sort e Selection Sort, porém parte do processamento iria causar muito stress no processador. O Quick Sort teve um melhor desempenho em todas as análises e cenários, assim como o Merge Sort por terem o princípio da recursão que é mais fácil resolver problemas menores e que o Merge Sort pode resolver certos problemas os dividindo em duas partes ou mais, até que um elemento seja deixado.

Com base nos testes realizados foram obtidas as seguintes conclusões, para usar aplicação em uma lista simples de no máximo 1000 elementos por exemplo, não vale a pena gastar recursos para fazer um processador que aguente um Quick Sort ou Merge Sort, pois eles usam recursividades e até mesmo iteratividade, precisando de mais memórias, então terá como uma opção optar por utilizar Bubble, Selection ou Insertion Sort dependendo muito sempre do que se quer ordenar.

Enfim, pode-se concluir que a implementação do algoritmo varia muito dependendo da função que quer executar e a parte mais difícil de fazer no trabalho foi as implementações dos algoritmos Quick e Merge Sort, além de ir a fundo nas pesquisas, obteve ideia de realizar o código encima de um vídeo que ensinava a fazer em linguagem Python, já que os dois tipos de algoritmos usam recursão e afins, sendo mais complexos que os outros algoritmos transformando para linguagem C++.

Link do vídeo: <https://youtu.be/lkg1jIQBf3g>