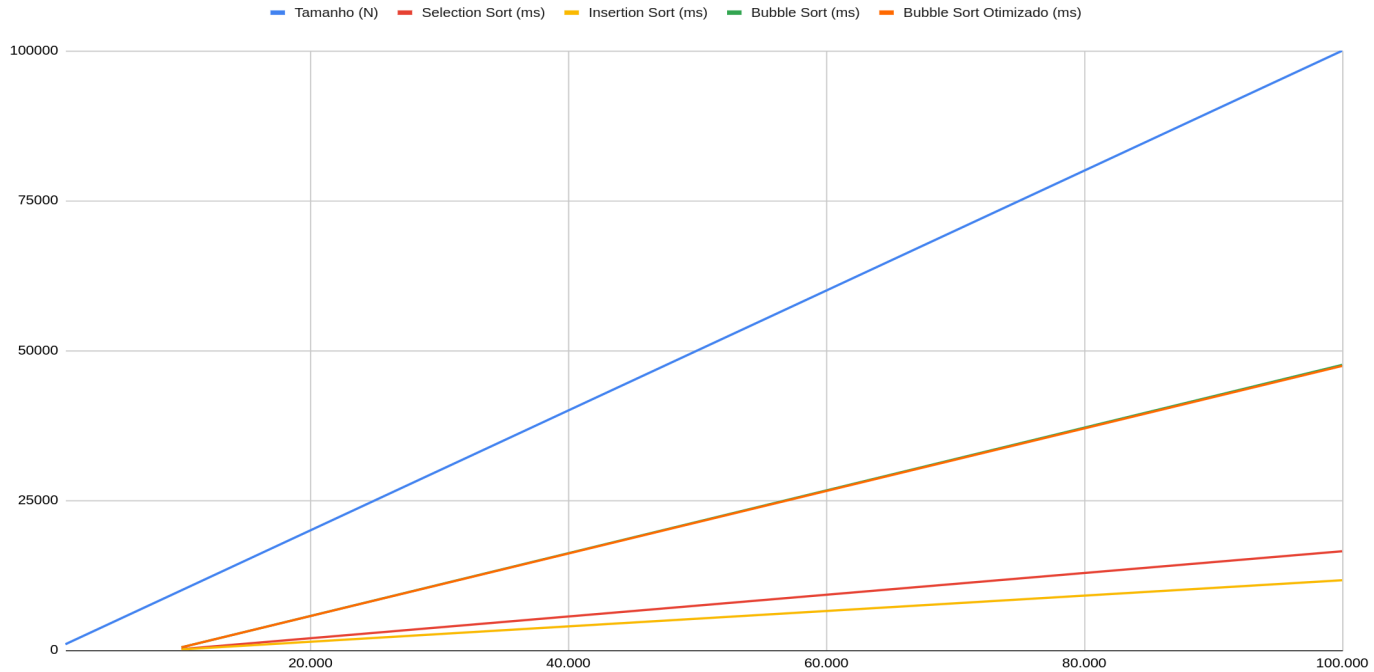
Atividade de Algoritmos e Estruturas de Dados 1

Nome: Leonardo Furlan Berton RA:2727536
Nome: Evellyn Cipriano RA:2719550

Tempo: Gráfico de Linha (*Tempo x Tamanho*) para o cenário Aleatório



Análise:
Todas as quatro curvas apresentam um crescimento exponencial (parábola), o que confirma que todos os algoritmos testados possuem complexidade assintótica(N2).

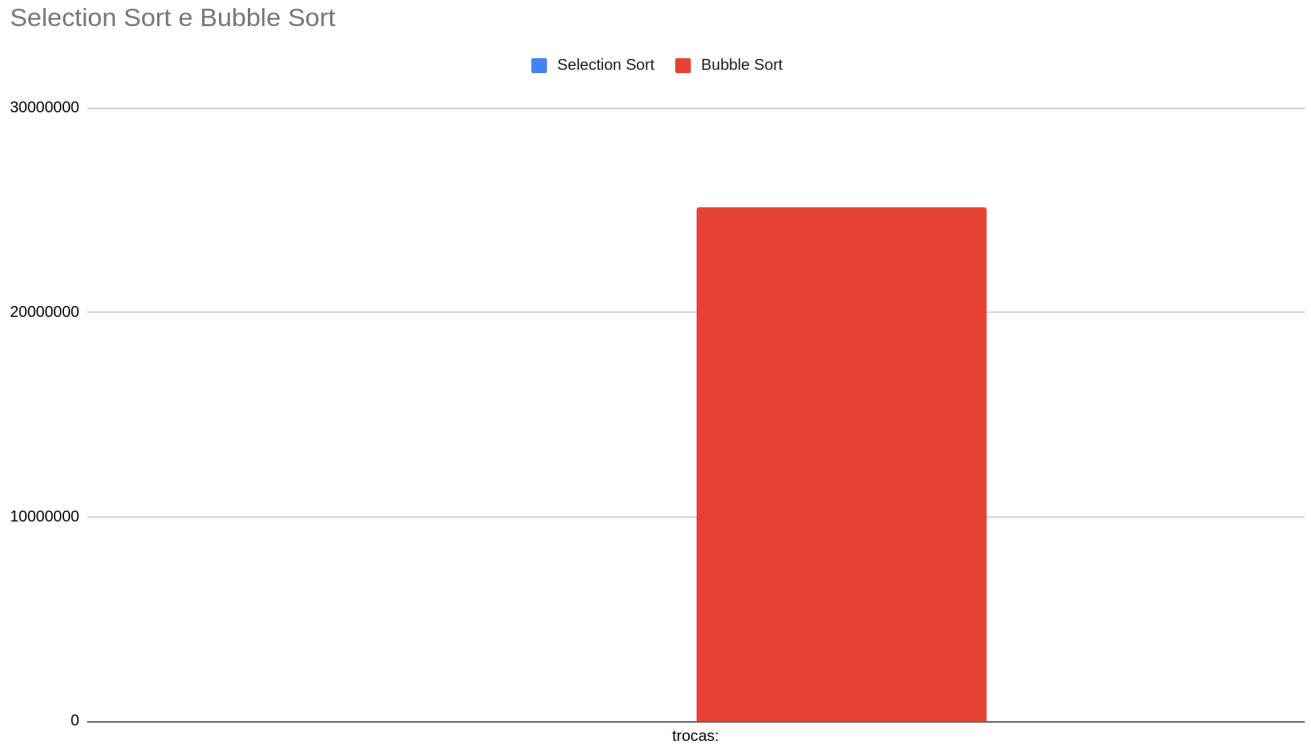
- À medida que o tamanho do vetor (N) aumenta de 10.000 para 100.000 (10x maior), o tempo não aumenta apenas 10x, mas sim aproximadamente 100x.
    - *Exemplo:* O Selection Sort pulou de 173 ms para 16.496 ms (aumento de quase 100 vezes).

Ranking de Desempenho:
1º Lugar: Insertion Sort (11,6s para N=100k). No cenário aleatório, o Insertion Sort se destaca porque, em média, ele só precisa percorrer metade da parte já ordenada para inserir um elemento.
2º Lugar: Selection Sort (16,5s para N=100k). É consistentemente mais lento que o Insertion Sort neste cenário, pois sempre faz todas as comparações possíveis (N(N−1)/2), independentemente dos valores. Porém, é muito mais rápido que o Bubble Sort devido ao baixo número de trocas.
3º e 4º Lugares (Mais Lentos): Bubble Sort e Bubble Sort Otimizado (47,5s para N=100k). Ambos apresentam o pior desempenho, levando quase 4 vezes mais tempo que o Insertion Sort.

Conclusão:A análise gráfica confirma experimentalmente a complexidade O(N2) para todos os algoritmos, evidenciada pelo crescimento do tempo em fator de 100 para um aumento de 10 vezes na entrada.

Entretanto, as diferenças de implementação geraram um ranking de desempenho claro no cenário aleatório: o Insertion Sort provou ser a opção mais eficiente, superando o Selection Sort. Em contrapartida, o Bubble Sort (em ambas as versões) demonstrou ser inviável para grandes volumes de dados, exigindo cerca de 4 vezes mais tempo que o Insertion Sort, o que evidencia sua ineficiência prática apesar da mesma classificação teórica.

Trocas (Swaps): Gráfico de Barras comparando o número de trocas realizadas pelo Selection Sort vs. Bubble Sort (Padrão) no cenário Aleatório (Tamanho Médio)



Análise:
Disparidade Gigantesca O gráfico evidencia uma diferença brutal na quantidade de operações de escrita na memória.

- Selection Sort: Realizou apenas 10 mil trocas (proporcional a N).
- Bubble Sort: Realizou mais de 25 milhões de trocas (proporcional a N2).

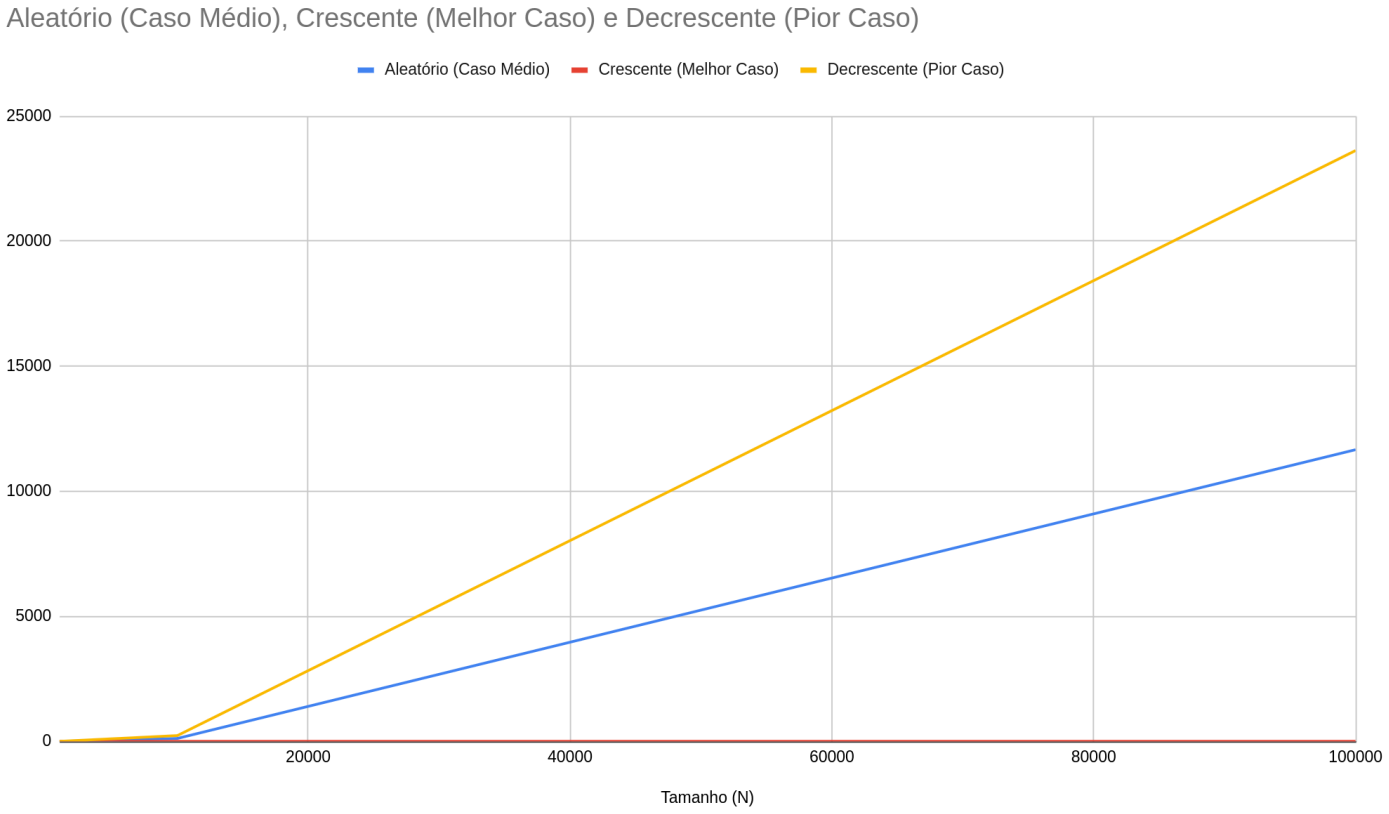
Comportamento dos Algoritmos

- O **Selection Sort** é "cirúrgico": ele varre o vetor inteiro comparando, mas só move o elemento para sua posição final **u**ma vez por rodada.
- O **Bubble Sort** é "ineficiente": ele move elementos excessivamente. Para levar um elemento pequeno do final para o início do vetor, ele precisa realizar milhares de trocas sucessivas.

Conclusão: Os dados comprovam que a complexidade teórica de comparações (O(N2)) não é o único fator determinante para o desempenho. A **eficiência na movimentação de dados** revelou-se o diferencial crítico entre os dois algoritmos.

Enquanto o **Selection Sort** demonstra uma estratégia otimizada de escrita (complexidade de trocas O(N)), realizando apenas o trabalho estritamente necessário para ordenar, o **Bubble Sort** desperdiça ciclos de processamento com uma quantidade massiva de movimentações redundantes (complexidade de trocas O(N2)).

Cenários: Gráfico comparando o desempenho do Insertion Sort nos cenários *Aleatório*, *Crescente* e *Decrescente*.



Análise:
O Abismo entre o Melhor e o Pior Caso O gráfico ilustra a característica mais marcante do Insertion Sort: ele é adaptativo. O desempenho muda drasticamente dependendo de como os dados já estão organizados.

- **Cenário Crescente (Linha "Colada" no Zero):** É o **Melhor Caso** (O(N)). O vetor já está ordenado, então o algoritmo apenas "confere" os números sem fazer nenhuma troca. O tempo é virtualmente zero (0,5ms para 100k elementos).
- **Cenário Decrescente (Linha Mais Alta):** É o **pior Caso** (O(N2)). O vetor está invertido. Para cada número, o algoritmo precisa percorrer e deslocar todo o sub vetor já ordenado. É o cenário mais lento possível (23s).

Cenário Aleatório (Caminho do Meio)

- A curva do cenário Aleatório fica entre os dois extremos (11s). Estatisticamente, espera-se que ele faça metade do esforço do pior caso, pois, em média, um elemento precisa ser deslocado apenas até a metade do sub vetor ordenado.

Conclusão: O gráfico prova que o **Insertion Sort é excelente para listas quase ordenadas** (cenário Crescente), superando até algoritmos mais complexos (como QuickSort ou MergeSort) nessas situações específicas, mas é péssimo para listas invertidas ou muito desordenadas.

Discussão Crítica:

Análise: Impacto das Trocas no Tempo Final (Selection Sort vs. Bubble Sort)

1. Contexto Teórico vs. Prático Teoricamente, tanto o Selection Sort quanto o Bubble Sort possuem a mesma complexidade assintótica de tempo O(N2), o que significa que ambos realizam um número quadrático de comparações. No cenário *Grande Aleatório* (N=100.000), ambos realizaram aproximadamente 5 bilhões de comparações (4.999.950.000).

No entanto, a complexidade de movimentação de dados (trocas) é drasticamente diferente:

- **Selection Sort:** Realiza O(N) trocas.
- **Bubble Sort:** Realiza O(N2) trocas no caso médio.

2. Evidência nos Dados Experimentais Os dados coletados no experimento *Grande Aleatório* demonstram isoladamente o custo dessa diferença:

- **Selection Sort:** Realizou 99.999 trocas e completou a ordenação em 16,5 segundos.
- **Bubble Sort:** Realizou 2,5 bilhões de trocas (2.496.135.185) e levou 47,6 segundos.

3. Conclusão do Impacto: O excesso de trocas teve um impacto devastador no desempenho do Bubble Sort, tornando-o quase 3 vezes mais lento (um aumento de ~188% no tempo), mesmo processando a mesma quantidade de comparações.

Análise: Eficácia do Bubble Sort Otimizado

1. No Cenário Aleatório: Houve ganho real? Resposta: Não**.** Nos testes realizados, o desempenho foi praticamente idêntico, e em alguns casos o "Otimizado" foi até ligeiramente mais lento.

- **Dados (Grande Aleatório):**
    - Bubble Sort Padrão: 47.581 ms
    - Bubble Sort Otimizado: 47.411 ms
- **Explicação Técnica:** A otimização do Bubble Sort consiste em uma variável de controle (flag) que verifica se houve trocas na rodada. Se não houver, ele encerra. No cenário aleatório, a probabilidade de uma rodada inteira ocorrer sem trocas é estatisticamente nula até o final da ordenação. Portanto, a verificação extra torna-se um custo inútil (overhead), sem trazer benefício prático.

2. No Cenário Crescente: Houve ganho real? Resposta: Sim, um ganho bastante considerável. A diferença de desempenho é brutal, transformando um algoritmo inviável em um instantâneo.

- **Dados (Grande Crescente):**
    - Bubble Sort Padrão: 16.605 ms (~16,6 segundos).
    - Bubble Sort Otimizado: 0,36 ms (menos de 1 milissegundo).
- **Explicação Técnica:** Este é o "Melhor Caso" do algoritmo.
    - O **Padrão** continua realizando cegamente O(N2) comparações, ignorando que o vetor já está pronto.
    - O **Otimizado** percebe na primeira passada (O(N)) que nenhuma troca foi feita e encerra a execução imediatamente. O gráfico abaixo ilustra essa diferença de magnitude (note a escala logarítmica).
    -

Análise: Custo de Ordenação vs. Benefício da Busca Binária

1. O Dilema: Custo Fixo vs. Custo Variável A decisão entre utilizar Busca Sequencial ou Busca Binária é um problema clássico de *trade-off*.

- **Busca Sequencial:** Tem custo de preparação **zero** , mas o custo por operação é alto.
- **Busca Binária:** Tem custo por operação **ínfimo**, mas exige um alto custo de preparação inicial (ordenar o vetor).

2. Dados do Experimento (Cenário: Grande Aleatório, N=100.000) Para essa análise, utilizaremos o tempo do Insertion Sort como referência para a ordenação, pois ele foi o algoritmo quadrático mais rápido neste cenário específico.

- **Custo para Ordenar:** 11.660 ms (11,66 segundos).
- **Tempo da Busca Sequencial :** 0,4 ms (média).
- **Tempo da Busca Binária:** 0,001 ms (praticamente instantâneo).

OBS: Caso queira olhar mais detalhadamente os gráficos e as tabelas com os valores, clique no link a seguir, do Google Planilhas.
https://docs.google.com/spreadsheets/d/1T2NcNAxK961k5I3iJDrv6g7g5bN2TfU8h0COYzycZMA/edit?usp=sharing