Grupo que realizou o experimento:

Nome:	**Prontuário**
Beatriz Galvão Verçosa	SP3064531
Alicia Beatriz Silva	SP3065651
Giovana Dias Valentini	SP3075451
Gisele Pontes da Silva	SP3065707

Esse experimento foi realizado em uma máquina com um processador: Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz 3.60 GHz, uma placa de vídeo: GTX1650TI e uma memória RAM de 8,00GB.

InsertionSort e QuickSort

InsertionSort (método de ordenação pouco eficiente): percorre um vetor de elementos da esquerda para a direita e à medida que avança vai ordenando os elementos à esquerda, sendo ele um método de ordenação estável.

- Os resultados apresentados na semente 13 mudaram de acordo com o tamanho do vetor, neste caso observamos que com base nos vetores 10.000, 30.000, 90.000 e 270.000, o programa demorou respectivamente 55, 350, 3077 e 27386 milissegundos para ser executado.

- Já na semente 21, os resultados também variaram de acordo com o tamanho dos vetores, neste caso observamos que com base nos vetores 10.000, 30.000, 90.000 e 270.000, o programa demorou respectivamente 42, 88, 776 e 8035 milissegundos para ser executado.

- Já na semente 34, os resultados também variaram de acordo com o tamanho dos vetores, neste caso observamos que com base nos vetores 10.000, 30.000, 90.000 e 270.000, o programa demorou respectivamente 10, 88, 880 e 7712 milissegundos para ser executado.

- Na semente 55, os resultados também variaram de acordo com o tamanho dos vetores, neste caso observamos que com base nos vetores 10.000, 30.000, 90.000 e 270.000, o programa demorou respectivamente 10, 89, 792 e 7713 milissegundos para ser executado.

- Ocorrendo assim o mesmo processo na sementea 89, os resultados também variaram de acordo com o tamanho dos vetores, neste caso observamos que com base nos vetores 10.000, 30.000, 90.000 e 270.000, o programa demorou respectivamente 11, 86, 792 e 7615 milissegundos para ser executado.

- Por fim, na semente 144, tendo o mesmo processo, os resultados também variaram de acordo com o tamanho dos vetores, neste caso observamos que com base nos vetores 10.000, 30.000, 90.000 e 270.000, o programa demorou respectivamente 11, 86, 841 e 27384 milissegundos para ser executado.

QuickSort (método de ordenação mais eficiente): algoritmo de comparação com a estratégia de "divisão e conquista", divide o problema de ordenar um conjunto com diversos itens em dois problemas menores, esses problemas menores são ordenados independente se os resultados são combinados para produzir a solução final.

- Os resultados apresentados na semente 13 foram variando de acordo com o tamanho do vetor, contudo neste caso analisamos sete vetores diferentes, sendo eles: 10.000, 30.000, 90.000, 270.000, 810.000, 2.430.000 e 65.610.000, o programa demorou respectivamente 4, 14, 29, 44, 104, 293 e 18951 milissegundos para ser executado.

- Na semente 21, da mesma forma os resultados variaram de acordo com o tamanho do vetor, neste caso observamos sete vetores, sendo eles: 10.000, 30.000, 90.000, 270.000, 810.000, 2.430.000 e 65.610.000, o programa demorou respectivamente 1, 2, 8, 28, 129, 272 e 15133 milissegundos para ser executado.

- Já semente 34, nós analisados novamente os vetores 10.000, 30.000, 90.000, 270.000, 810.000, 2.430.000 e 65.610.000, tendo ciência que os resultados sempre variaram de acordo com o tamanho do vetor, neste caso o programa demorou respectivamente 1, 3, 9, 26, 130, 301 e 27323 milissegundos para ser executado.

- Na semente 55, os resultados também variaram de acordo com o tamanho dos vetores, neste caso observamos sete vetores, sendo eles: 10.000, 30.000, 90.000, 270.000, 810.000, 2.430.000 e 65.610.000, o programa demorou respectivamente 1, 2, 8, 27, 119, 287 e 19607 milissegundos para ser executado.

- Ocorrendo assim o mesmo processo na semente 89, da mesma forma os resultados também variaram de acordo com o tamanho dos vetores, neste caso observamos sete vetores, sendo eles: 10.000, 30.000, 90.000, 270.000, 810.000, 2.430.000 e 65.610.000, neste caso o programa demorou respectivamente 1, 3, 8, 25, 131, 263 e 29042 milissegundos para ser executado.

- E por fim, na semente 144 os resultados também variaram de acordo com o tamanho dos vetores, analisamos nesse caso, sete valores, sendo eles: 10.000, 30.000, 90.000, 270.000, 810.000, 2.430.000 e 65.610.000, neste caso o programa demorou respectivamente 1, 3, 12, 56, 116, 310 e 28627 milissegundos para ser executado.

BubleSort e ShellSort

BubleSort (método de ordenação pouco eficiente): funciona examinando cada conjunto de elementos adjacentes na string, da esquerda para a direita, trocando suas posições se estiverem fora de ordem.

- Os resultados presentes na semente 13 mudaram de acordo com o tamanho do vetor, então observamos que diante dos vetores 10.000, 30.000, 90.000 e 270.000, o programa demorou respectivamente 167, 1518, 13877 e 118692 milissegundos para ser executado.

- Utilizando a semente 21, o vetor com 10.000 números de espaços teve um tempo de execução de 132 ms, o vetor com 30.000 foi executado em 1240 ms, o com 90.000 espaços teve um tempo de 11403 ms e o vetor com 270.000 números de espaços foi executado em 111254 ms.

- Observamos que com a semente 34, os vetores com 10.000, 30.000, 90.000 e 270.000 números de espaços demoraram, respectivamente, 133, 1227, 11413 e 111617 milissegundos para ser executado.

- Com a semente 55, os vetores 10.000, 30.000, 90.000 e 270.000 tiveram um tempo de execução de 129, 1234, 11857 e 112244 milissegundos.

- Analisando o caso da semente 89, o vetor com 10.000 números de espaços teve um tempo de execução de 124 ms, o vetor com 30.000 foi executado em 1232 ms, o com 90.000 espaços teve um tempo de 11615 ms e o vetor com 270.000 números de espaços foi executado em 111486 ms.

- Por último, utilizando a semente 144, o vetor de 10.000 espaços teve um tempo de 2137 ms, o de 30.000 foi executado em 1276 ms, o de 90.000 em 11755 ms e o vetor com 270.000 espaços foi executado em 118453 ms.

ShellSort (método de ordenação mais eficiente): permite a troca de registros distantes um do outro, diferente do algoritmo de ordenação por inserção que possui a troca de itens adjacentes para determinar o ponto de inserção.

- Os resultados presentes na semente 13 mudaram de acordo com o tamanho do vetor, então observamos que diante dos vetores 10.000, 30.000, 90.000, 270.000, 810.000, 2.430.000, 65.610.000, o programa demorou respectivamente 10, 14, 22, 61, 171, 574 e 22826 milissegundos para ser executado.

- Utilizando a semente 21, o vetor com 10.000 números de espaços teve um tempo de execução de 2 ms, o vetor com 30.000 foi executado em 4 ms, o com 90.000 espaços teve um tempo de 12 ms, o vetor com 270.000 números de espaços foi executado em 43 ms, o de 810.000 demorou 178 ms, 2.430.000 demorou 568 ms e o com 65.610.000 espaços demorou 24279 ms.

- Observamos que com a semente 34, os vetores com 10.000, 30.000, 90.000, 270.000, 810.000, 2.430.000, 65.610.000, números de espaços demoraram, respectivamente, 1, 4, 11, 44, 171, 567 e 22844 milissegundos para ser executado.

- Com a semente 55, os vetores 10.000, 30.000, 90.000, 270.000, 810.000, 2.430.000 e 65.610.000, tiveram um tempo de execução de 1, 4, 12, 45, 187, 623 e 22126 milissegundos.

- Analisando o caso da semente 89, os vetores com 10.000, 30.000, 90.000, 270.000, 810.000, 2.430.000 e 65.610.000 números de espaços, demoraram 1, 4, 13, 43, 179, 617 e 23432 milissegundos para serem executados.

- Por último, utilizando a semente 144, o vetor de 10.000 espaços teve um tempo de 1 ms, o de 30.000 foi executado em 4 ms, o de 90.000 em 13 ms, o vetor com 270.000 espaços foi executado em 57 ms, o com 810.000 demorou 188 ms, 2.430.000 demorou 567 ms e o vetor com 65.610.000 espaços demorou 24922 ms para ser executado.

MergeSort e SelectionSort

MERGESORT

- Os resultados apresentados na semente 13 mudaram de acordo com o tamanho do vetor, neste caso observamos que com base nos vetores 10.000, 30.000, 90.000 e 270.000, 810000, 2430000, 65610000, o programa demorou respectivamente 5, 15, 41,81, 193, 514 e 11862 milissegundos para ser executado.

- Já na semente 21, os resultados também variaram de acordo com o tamanho dos vetores, neste caso observamos que com base nos vetores 10.000, 30.000, 90.000 e 270.000,810000, 2430000, 65610000 o programa demorou respectivamente 2, 10, 12, 47, 187, 504 e 12104 milissegundos para ser executado.

- Já na semente 34, os resultados também variaram de acordo com o tamanho dos vetores, neste caso observamos que com base nos vetores 10.000, 30.000, 90.000 e 270.000,810000, 2430000, 65610000, o programa demorou respectivamente 2, 5, 12, 46, 186, 526 e 12055 milissegundos para ser executado.

- Na semente 55, os resultados também variaram de acordo com o tamanho dos vetores, neste caso observamos que com base nos vetores 10.000, 30.000, 90.000 e 270.000, 810000, 2430000, 65610000 o programa demorou respectivamente 2, 4, 15, 45, 188, 513, 11961 milissegundos para ser executado.

SELECTSORT

- Os resultados apresentados na semente 13 foram variando de acordo com o tamanho do vetor, contudo neste caso analisamos sete vetores diferentes, sendo eles: 10.000, 30.000, 90.000, 270.000 o programa demorou respectivamente 61, 318, 3223, 27721 milissegundos para ser executado.

- Na semente 21, da mesma forma os resultados variaram de acordo com o tamanho do vetor, neste caso observamos sete vetores, sendo eles: 10.000, 30.000, 90.000, 270.000, o programa demorou respectivamente 40, 347, 3335, 33004 milissegundos para ser executado.

- Já semente 34, foram analisado novamente sete tamanhos diferentes de vetores, sendo eles: 10.000, 30.000, 90.000, 270.000, com os resultados sempre variaram de acordo com o

tamanho do vetor, e tendo eles recebido respectivamente os seguintes resultados:43, 340, 3351, 32970.

- Na semente 55, os resultados também variaram de acordo com o aumento do tamanho dos vetores, neste caso foi possível obter sete valores com tamanhos diferentes de vetores, sendo eles: 10.000, 30.000, 90.000, 270.000, tendo eles recebido respectivamente os seguintes resultados: 38, 394, 3276 e 33048.

- Ocorrendo assim o mesmo processo na semente 89, da mesma forma os resultados também variaram de acordo com o aumento do tamanho dos vetores, sendo nesse caso valores com sete tamanhos diferentes de vetores, sendo eles: 10.000, 30.000, 90.000, 270.000, tendo eles recebido respectivamente os seguintes resultados:39, 376,3426, 32483.

- E por fim, na semente 144 os resultados também variaram de acordo com o aumento do tamanho dos vetores, tendo obtido nesse caso, sete valores com tamanhos diferentes de vetores, sendo eles: 10.000, 30.000, 90.000, 270.000,, tendo eles recebido respectivamente os seguintes resultados: 40, 372, 3660, 27635.

Observações: Um fator observado que deve-se levar em consideração é que mesmo com sementes diferentes houveram alguns tempos em milissegundos que apareceram mais de uma vez ao longo dos diferentes processos de ordenação, além disso outro fator que foi ressaltado ao longo de todo o processos mais especificamente do método de ordenação pouco eficiente, são as quantidades de números de vetores utilizados, pelo método de ordenação ser pouco eficiente não é recomendado que se utilize valores muito grandes.