Relatório Projeto 4.1 AED 2021/2022

Nome: João Miguel Fernandes Moura

Nº Estudante: 2020235800

PL (inscrição):7

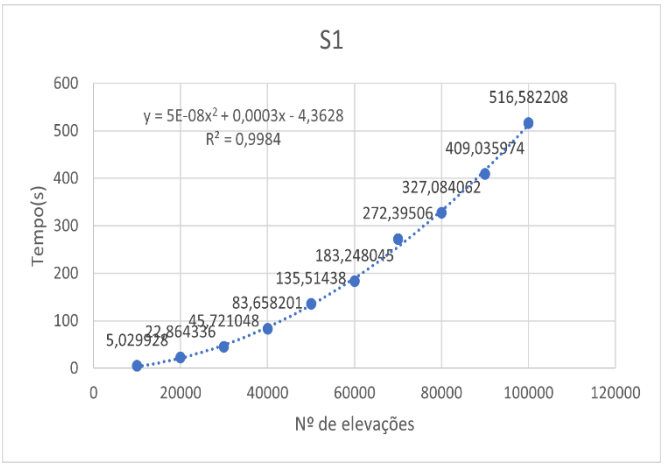
Login no *Mooshak*: 2020235800

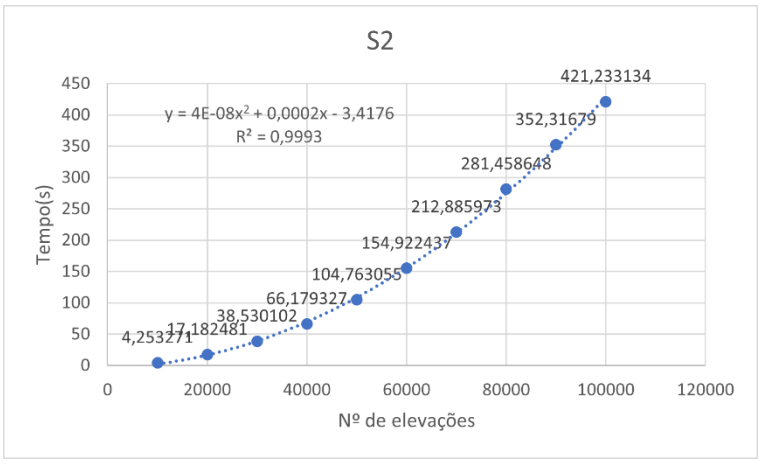
Tabela e Gráfico(S1)

Nº de elevações(S1)	Tempo(s)
10000	5,029928
20000	22,864336
30000	45,721048
40000	83,658201
50000	135,514380
60000	183,248045
70000	272,395060
80000	327,084062
90000	409,035974
100000	516,582208

Tabela e Gráfico(S2)

Nº de elevações(S2)	Tempo(s)
10000	4,253271
20000	17,182481
30000	38,530102
40000	66,179327
50000	104,763055
60000	154,922437
70000	212,885973
80000	281,458648
90000	352,316790
100000	421,233134





A expressão O(f(n)) para a complexidade temporal está de acordo com o esperado para as soluções S1 e S2? Justifique.

Pela análise dos resultados obtidos, vemos em ambas as soluções uma regressão polinomial de grau 2, ou seja, uma complexidade temporal n^2 o que se encontra de acordo

com o previsto teoricamente. Na solução 1 (sem ordenação), esta complexidade deve-se à utilização de dois ciclos for, um para percorrer o *array* de valores para procurar os valores para calcular o percentil e outro para percorrer um *array* com todos os percentis a ser calculados. Na solução 2 a complexidade deve-se à utilização do *insertion sort* que possui complexidade n^2.

Qual a expressão O(f(n)) para a complexidade espacial nas soluções S1 e s2? Justifique.

A complexidade espacial em ambas as soluções é O(1) dado que não é utilizado espaço adicional ou *arrays* adicionais. Só é utilizado o *array* de *input*, sem necessitar de nenhuma estrutura de dados adicional na memória.

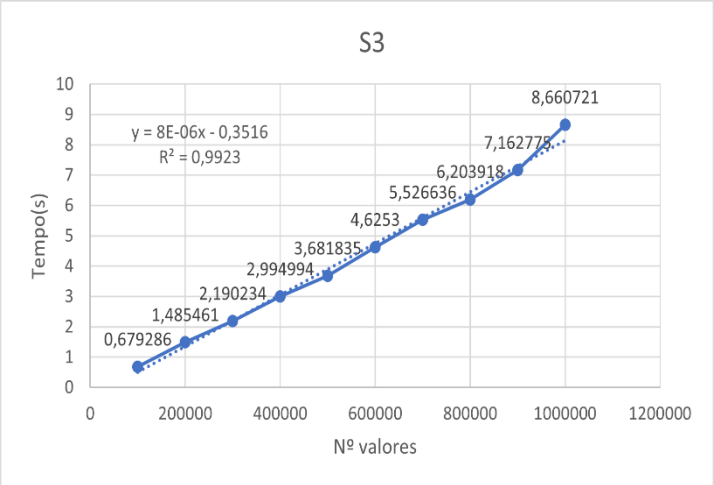
Relatório Projeto 4.2 AED 2021/2022

Nome: João Miguel Fernandes Moura
Nº Estudante: 2020235800
PL (inscrição): 7 *Login* no *Mooshak:2020235800*

Tabela (S3)

Nº de valores	Tempo(s)
100000	0,679286
200000	1,485461
300000	2,190234
400000	2,994994
500000	3,681835
600000	4,625300
700000	5,526636
800000	6,203918
900000	7,162775
1000000	8,660721

Gráfico (S3)



(1) Descreva sucintamente as otimizações feitas ao QuickSort. A expressão O(f(n)) está de acordo com o esperado? Justifique.

Uma das melhorias aplicadas faz com que quer o índice mais baixo (low) como o mais alto (high) caso um deles encontre um elemento igual ao pivot param ambos o avanço e trocam os elementos que é o que conduz a uma complexidade temporal melhor. Outra melhoria implementada é a utilização do *insertionSort* que é mais eficiente para *arrays* de pequena dimensão (<30). A utilização de chamadas recursivas do *quicksort*, permite que o algoritmo seja aplicado a cada uma das partições cada vez mais pequenas. Para o cálculo da mediana também há uma melhoria uma vez que se faz a mediana entre o ínicio e meio do *array* e o meio e o fim sendo que o valor da mediana fica como pivot.

Analisando os resultados obtidos claramente vemos que se obtém uma regressão linear, ou seja, uma complexidade O(n) o que é uma boa aproximação do teoricamente previsto que seria O(nlog(n)), sendo que o n é no pior caso, ou seja, quando o *pivot* é o maior ou menor elemento da sequência, havendo n níveis de comparação. O log(n) é devido ao melhor caso em que o *pivot* divide em duas sequências do mesmo tamanho havendo log(n) chamadas recursivas.

Qual a expressão O(f(n)) para a complexidade espacial na solução S3? Justifique.

A complexidade espacial da solução é O(1) dado que não é utilizado espaço adicional ou arrays adicionais. Só é utilizado o array de input, sem necessitar de nenhuma estrutura de dados adicional na memória.

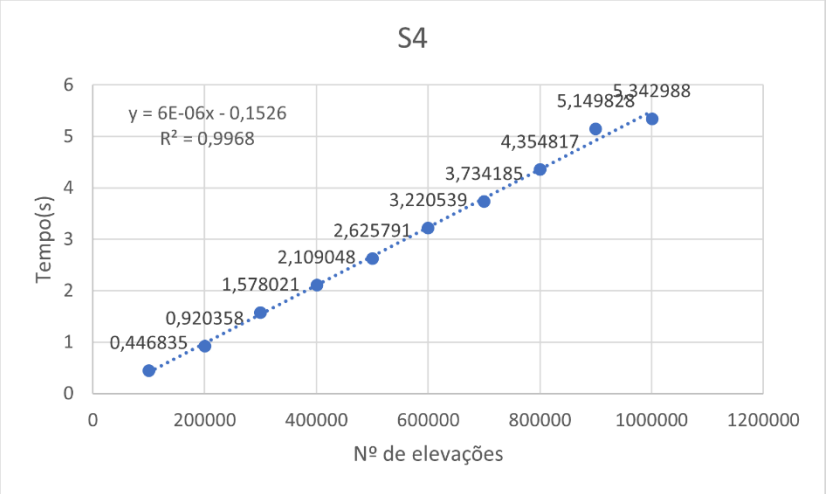
Relatório Projeto 4.3 AED 2021/2022

Nome: João Miguel Fernandes Moura
Nº Estudante: 2020235800
PL (inscrição): 7 *Login* no *Mooshak:2020235800*

Tabela (S4)

Nº de elevações(S4)	Tempo(s)
100000	0,446835
200000	0,920358
300000	1,578021
400000	2,109048
500000	2,625791
600000	3,220539
700000	3,734185
800000	4,354817
900000	5,149828
1000000	5,342988

Gráfico (S4)



A expressão O(f(n)) está de acordo com o esperado? Justifique.

Analisando a regressão obtida, vemos que a complexidade temporal obtida experimentalmente é O(n) o que está de acordo com o teoricamente previsto, dado que o algoritmo utilizado (*counting sort)* tem complexidade temporal linear. Este algoritmo tem uma complexidade temporal média de O(n+k), sendo K o intervalo de valores dos elementos (maior elemento-menor elemento). Contar as ocorrências de cada elemento leva K tempo e procurar o valor do seu índice leva n tempo, no entanto, caso haja elementos significativamente maiores que outros, uma vez que o intervalo de valores é elevado, a complexidade temporal vai se aproximando cada vez mais de O(k) (consoante a ordem do k pode levar O(n+k^2), O(n+k^3), etc.). No melhor caso, a complexidade será linear, isto é caso a dispersão de valores seja baixa ou entao o K seja 1. Neste teste experimental a dispersão é baixa, uma vez que a complexidade temporal é linear.

Qual a expressão O(f(n)) para a complexidade espacial na solução S4? Justifique.

A complexidade espacial é O(k), sendo k o elemento máximo do a*rray* de *input*. Isto porque o *counting sort* cria um *array* auxiliar com as ocorrências de cada elemento, logo a complexidade espacial piora consoante maior seja o k, ou seja, maior seja a gama de valores possíveis para contar as ocorrências.

Relatório Projeto 4.4 AED 2021/2022

Nome: João Miguel Fernandes Moura
Nº Estudante:2020235800
PL (inscrição):7 *Login* no *Mooshak:2020235800*

S1 - Tabela (complexidade temporal)

Nº de elevações(S1)	Tempo(s)
10000	5,029928
20000	22,864336
30000	45,721048
40000	83,658201
50000	135,514380
60000	183,248045
70000	272,395060
80000	327,084062
90000	409,035974
100000	516,582208

S2 - Tabela (complexidade temporal)

Nº de elevações(S2)	Tempo(s)
10000	4,253271
20000	17,182481
30000	38,530102
40000	66,179327
50000	104,763055
60000	154,922437
70000	212,885973
80000	281,458648
90000	352,316790
100000	421,233134

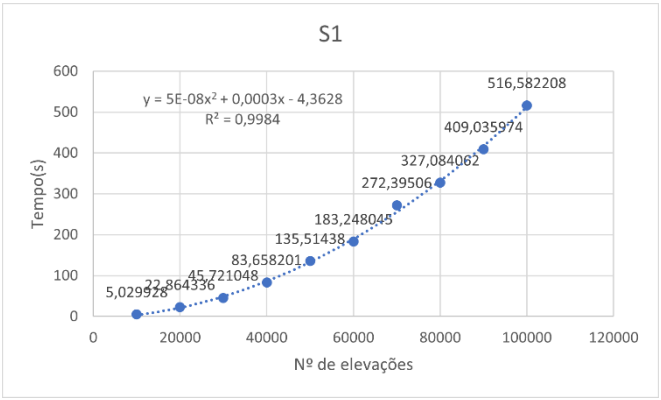
S3 - Tabela (complexidade temporal)

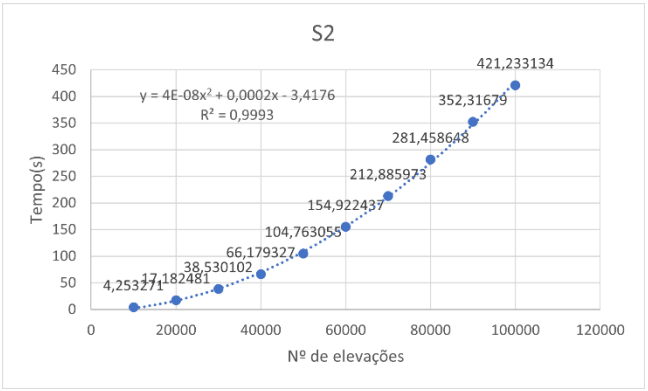
Nº de valores	Tempo(s)
100000	0,679286
200000	1,485461
300000	2,190234
400000	2,994994
500000	3,681835
600000	4,625300
700000	5,526636
800000	6,203918
900000	7,162775
1000000	8,660721

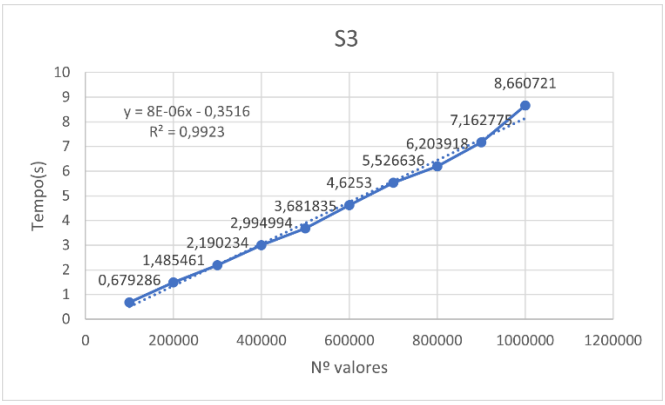
S4 - Tabela (complexidade temporal)

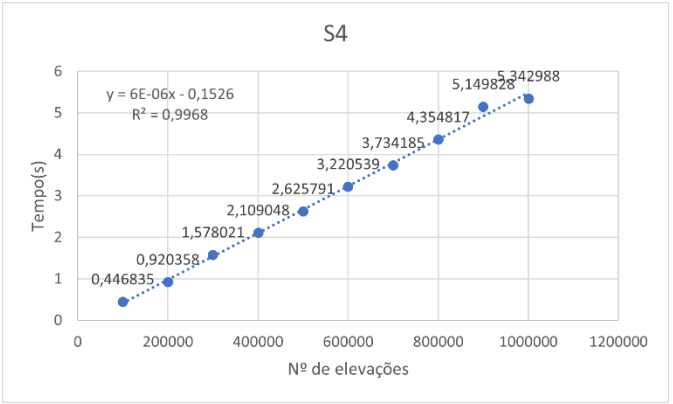
Nº de elevações(S4)	Tempo(s)
100000	0,446835
200000	0,920358
300000	1,578021
400000	2,109048
500000	2,625791
600000	3,220539
700000	3,734185
800000	4,354817
900000	5,149828
1000000	5,342988

Gráfico de Complexidade Temporal S1 .. S4 (escala logarítmica)









Explique sucintamente a implementação "força bruta" implementada em S1. E a solução implementada em S4.

Desenvolva os comentários que considere relevantes sobre a complexidade temporal vs espacial das várias implementações da solução.

Na solução S1, o programa não ordena o Raster de entrada sendo a complexidade temporal O(n^2). Todas as operações realizadas vão demorar mais tempo a executar uma vez que o array percorrido não está ordenado e consequentemente o percentil não vai usar o *binary search*. A solução S1 tem uma complexidade O(n) uma vez que é criado um *array* de espaço "n". A solução S2 apresenta uma complexidade O(1) dado que não gera nenhum *array* auxiliar durante a execução do algoritmo. Nenhuma das soluções gera *arrays* adicionais utilizados ao longo do programa.

Na solução S4, foi utilizado o algoritmo "Counting sort" para ordenar o Raster antes da execução de operações. A complexidade temporal deste algoritmo no pior dos casos é O(k) como comprovado teoricamente e pelos resultados obtidos, isto acontece quando o maior elemento do array é consideravelmente superior as outros sendo k o alcance dos elementos. O melhor caso verifica-se então quando todos os elementos são do mesmo alcance, ou seja, o valor de **k** ser 1. A complexidade é então O(n). O caso médio é então O(n+k). A complexidade espacial é O(n) porque é utilizado um array auxiliar com o tamanho do máximo elemento do *input* para a contagem de ocorrências.

Podemos facilmente verificar que a solução S4 é muito mais eficiente a nível de tempo de execução dado ter uma melhor complexidade temporal quer em relação à solução S1 quer ao *quicksort*. No entanto a complexidade espacial piora na solução S4 visto requerer espaço adicional. Para concluir devemos ter em conta que cada solução se aplica em contextos diferentes, um *insertion sort* será mais eficiente para poucos dados que um *quicksort,* por exemplo.