Relatório Projeto 3 AED 2023/2024

Nome: Miguel Curto Castela Email: uc2022212972@student.uc.pt

Nº Estudante: 2022212972 PL : PL8

1. Planeamento

	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5
Insertion Sort	X				
Heap Sort			X		
Quick Sort			X		
Finalização Relatório				X	X

2. Recolha de Resultados

Insertion sort

Tempo (µs) / nº de elementos	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
Crescente	60,3	186,4	180,9	273	301,3	361,3	421,2	481,4	560	617,4
Decrescente	280960	1135850	2511130	4421060	6865870	9808850	13429800	17778900	22050400	27521100
Random	143473	550308	1218290	2209090	3409550	4951000	6689420	8735940	11134400	13571200

Heap sort

Tempo (µs) / nº de elementos	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
Crescente	3505,7	7002,8	16221,5	15513,1	18705,5	23839,9	32294	33361,3	40097,1	39585,6
Decrescente	3324,4	8507,5	12795,7	14721,9	17633,4	21912,9	25472,1	34126,5	35619,1	37939
Random	4299,3	9379,8	14566,1	17171,9	20963	25857,3	30459,8	36725,9	42867,7	45761,6

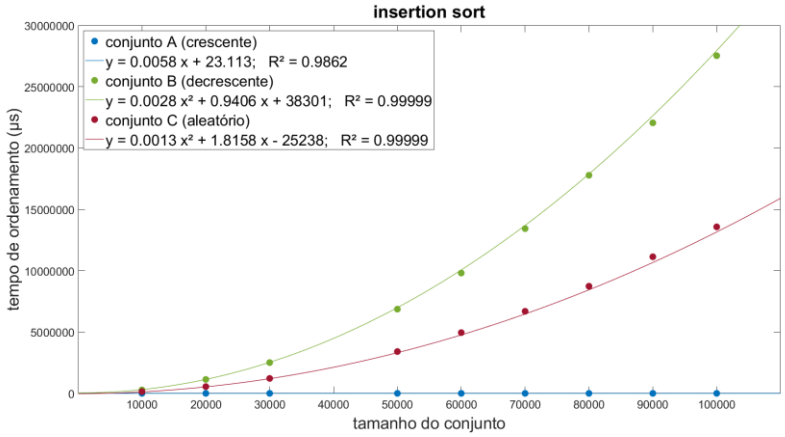
Quick sort

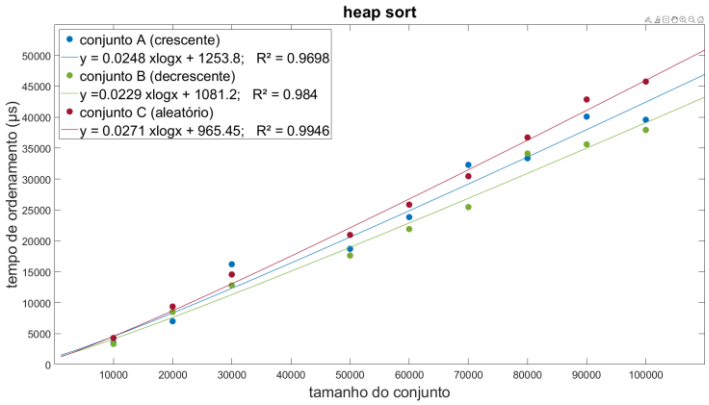
Tempo (µs) / nº de elementos	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
Crescente	418272	1609375	3509800	6308360	9954180	14054800	18787400	25119700	31471400	38687600
Decrescente	253752	1017820	2311460	3972530	6241100	8877310	12005000	15715500	19902400	27521100
Random	1794	3262,2	5190,1	6855,9	8495,5	11264	12569,8	16032,4	19169,9	24870,2

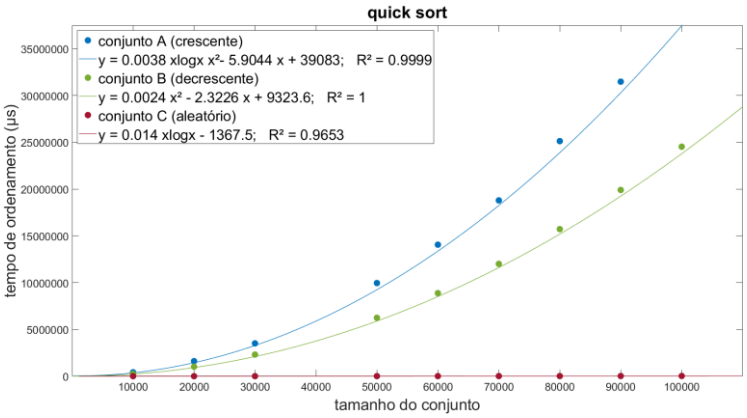
Quick sort (otimizado)

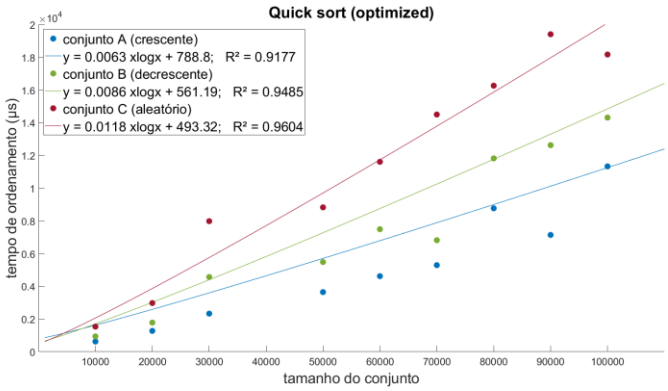
Tempo (µs) / nº de elementos	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
Crescente	623,3	1273,9	2329,6	2836,4	3645,1	4618,9	5292,3	8767,5	7139,3	11329,9
Decrescente	937,9	1780,5	4567	4300,8	5482,7	7488,7	6816,5	11817,7	12625,7	14311,8
Random	1535,4	2975,3	7980,6	7011,1	8825,4	11609,6	14500,5	16264,1	19406,3	18168

3. Visualização de Resultados (por algoritmo)

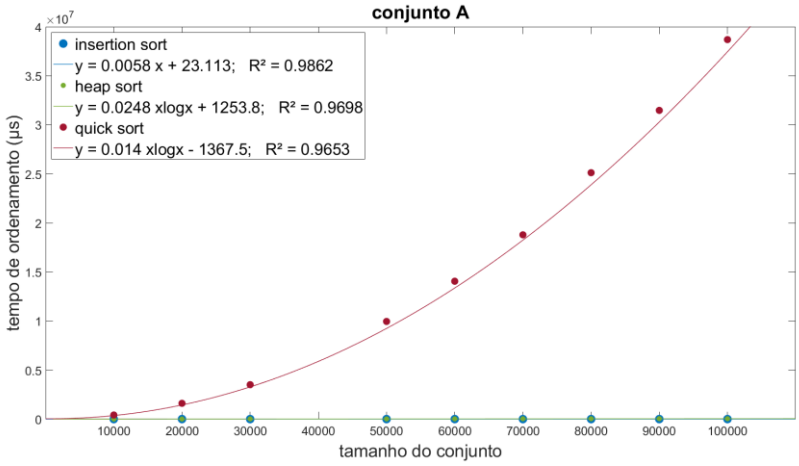


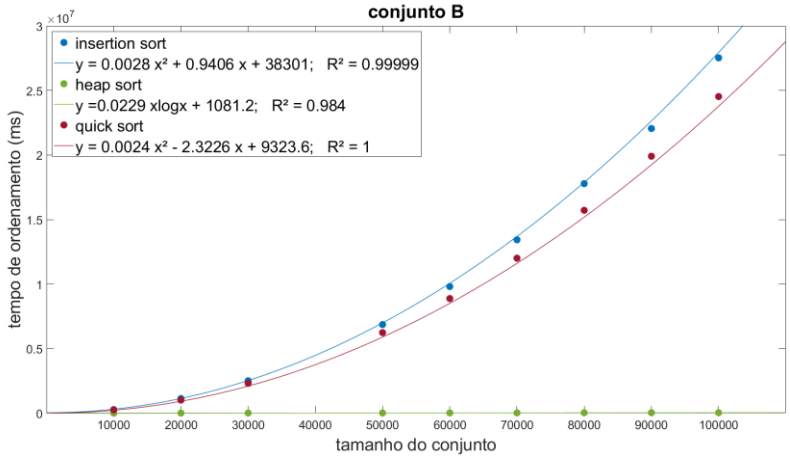


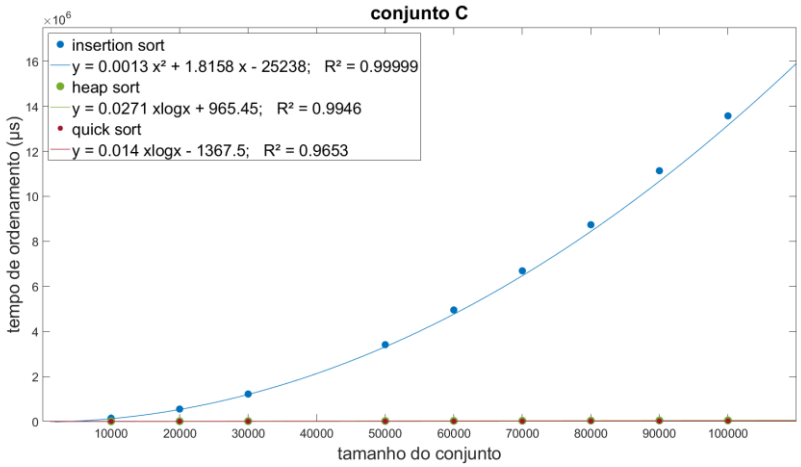




3. Visualização de Resultados (por conjunto)







4. Conclusões

1)Insertion Sort

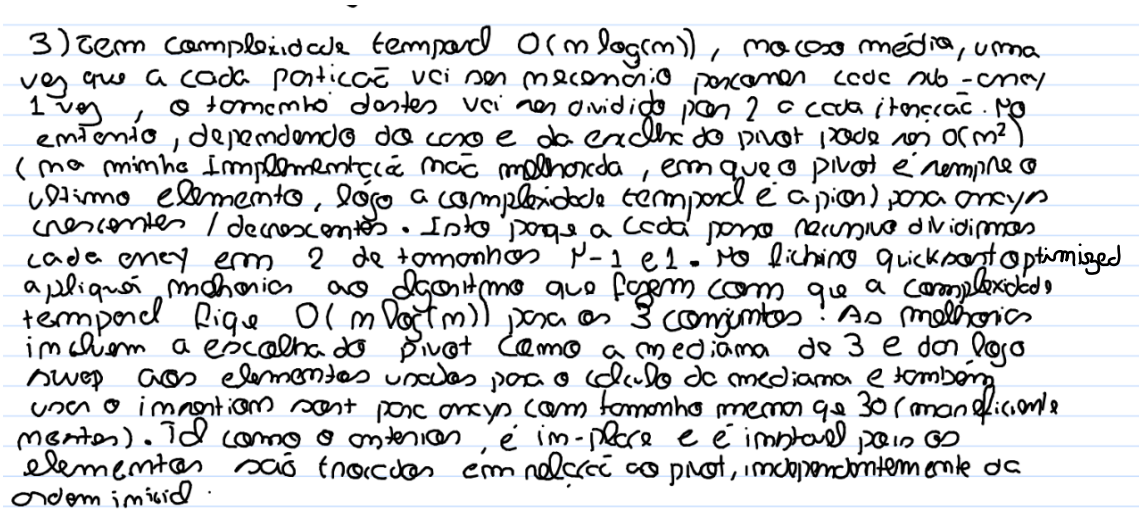
1) Tem complexidade temporal $O(n^2)$, dado que no seu pior caso (ordem inversa), é preciso percorrer todo o array (até ao momento), até comparar o valor relacionado com um inferior. No seu melhor caso (array crescente), a complexidade é $O(n)$ (linear), pois basta percorrer o array já ordenado apenas uma vez. Para os restantes comprimentos tem um desempenho mau devido à complexidade quadrática. É eficaz para comprimentos pequenos, estável pois a ordem de 2 elementos só é alterada quando um é estritamente maior e é "in-place" (memória adicional necessária é $O(1)$).

2)Heap Sort

2) tem complexidade temporal $O(n \log(n))$ para todos os casos, dado que os seus ramos são 1 -> construção da heap ($O(n)$) e, para cada um dos n elementos, a remoção do maior elemento e volta a percorrer a árvore a restabelecer as propriedades da max heap é $O(\log(n))$, daí a complexidade final $O(n \log(n))$. Isto explica a pouca diferença nos tempos de cada comprimento. Relativamente ao quicksort, envolve mais comparações na maioria dos casos e por isso tende a demorar mais tempo. Embora seja in-place*, não é um algoritmo estável pois pode alterar a ordem inicial, dependendo do elemento que for encontrado 1º e consequentemente colocado no final.

* (pois a heap está guardada no próprio array, não sendo preciso memória adicional

3)Quick Sort



Anexo A - Delimitação de Código de Autor

Todo o código foi implementado por mim com a ajuda dos slides e videos apresentados nas aulas teóricas.

Anexo B - Referências

Slides das aulas

Visualização:

(Quick Sort)

https://www.youtube.com/watch?v=Hoixgm4P4M&pp=ygUXcXVpY2tzb3J0IHZpc3VhbGl6YXRpb24%3D

(Heap Sort)

https://www.youtube.com/watch?v=2DmK_H7IdTo&pp=ygUXaGVhcCBzb3J0IHZpc3VhbGl6YXRpb24%3D

(Insertion Sort)

https://www.youtube.com/watch?v=JU767SDMDvA&pp=ygUcaW5zZXJ0aW9uIHNvcnQgdmlzdWFsaXphdGlvbg%3D%3D