Tarefa 1 de Estrutura de Dados
Aluna: Ana Luiza Seabra V. Soares

Questão 1: Pesquise e implemente o BubbleSort que tenha complexidade de melhor caso O(n). Explique com suas palavras, porque esse algoritmo é melhor do que a versão apresentada em aula.

O algoritmo que eu utilizei dispensa que o array seja revisto e comparado diversas vezes caso não haja trocas na primeira repetição, ou seja, caso esteja ordenado. Ao usar um array base com os números já ordenados, {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}, temos uma situação de complexidade de melhor caso O(n), pois dispensamos as trocas e a repetição que comparará os números será realizada apenas uma vez. Além disso, as comparações são feitas em diferentes pares de elementos, ao invés de vários pares com o primeiro elemento de comparação fixo.
Abaixo há a demonstração de uma leitura do passo a passo do código realizado em aula e o que eu escolhi implementar, ambos utilizando o mesmo vetor já ordenado.

Array de exemplo: {1, 2, 3}

Bubble Sort ensinado em aula: $O(n^2)$	Bubble Sort que eu implementei: O(n)
For i=0;i<2;i++	For i=0; i<3-1 && trocou == 1; j++
For j=1; j<3; j++	Trocou = 0;
If 1>2 troca -> false j++; j=2	For j=0; j<3-i-1; j++
If 1>3 troca -> false j++; j=3 ~~for~~	If 1>2 troca -> false j++; j =1
i++; i =1;	If 2>3 troca -> false j++; j =2 ~~for~~
For j=2; j<3; j++	Trocou = 0; ~~For~~
If 2>3 troca -> false j++ ; j=3 ~~for~~	
I++; I = 2; ~~for~~	

Podemos analisar que no primeiro código foi necessário comparar 1 termo com todos e só depois partir para o próximo, o que em um array maior se tornaria trabalhoso. Já no outro código, a posição de comparação não é fixa. Outra vantagem é que no segundo código, só em 1 repetição do “for” é possível notar que o array está ordenado, pelo fato de não haver trocas, e o código logo encerra por conta da condição do “for”, de trocou ser igual a 1(verdadeiro).

Questão 2: Pesquise e implemente o InsertionSort. Explique com suas palavras o funcionamento dele e a(s) diferenças entre ele e os demais algoritmos de ordenação vistos em aula (todos). Mostre também as complexidades de pior, de caso médio e de melhor caso.

O insertionSort possui um algoritmo que analisa cada elemento do vetor e verifica se o seu anterior é maior do que o número em questão, caso seja, é realizado uma troca (somente se o elemento não estiver na posição 0, pois assim não teria como comparar com o anterior). Após essa troca, os elementos devem ser realocados de acordo com a necessidade, então o contador vai da posição em que a troca foi realizada até a posição 0 realocando o vetor (realizando trocas) até encontrar a posição correta do elemento (quando o anterior é menor do que ele). Esse método é comparado com a organização de um baralho de cartas, ao chegar

um elemento (carta) novo, fazemos comparações para achar seu local ideal e realocamos as demais.

A diferença dele para o BubbleSort é o número de trocas necessárias em cada um, onde o insertion realiza menos comparações e trocas, principalmente se a listar estiver quase ordenada. Apesar do algoritmo ser menor e mais simples no BubbleSort, a eficiência do insertion se mostra superior justamente por isso.
Já entre o Selection e o Insertion, a diferença está também no método, pois o selection passa primeiro por todo vetor procurando o menor e coloca ele na primeira posição, fazendo isso com os demais. Já o insertion, não há essa procura pelo menor, os elementos são rearranjados a partir da comparação com o elemento anterior. O insertion apresenta um desempenho melhor, pois faz menos comparações/trocas.
A semelhança entre o Bubble, Selection e o Insertion é o pedaço do código que realiza as trocas, com a ajuda de um auxiliar.
Apesar de a complexidade de pior caso dos 3 algoritmos, insertion, bubble e selection serem semelhantes, na prática o insertion apresenta um desempenho melhor.
Especificamente em comparação com o Merge e o Quick, o algoritmo do Insertion não utiliza o método de divisão e conquista, muito menos de intercalação ou concatenação.

Seus níveis de complexidade são:

Melhor caso	**Caso médio**	**Pior caso**
O(n)	$O(n^2)$	$O(n^2)$
Quando o array já está ordenado em ordem crescente	O array não está ordenado em nenhuma das formas	Quando o array está ordenado em ordem decrescente
O algoritmo não chega a entrar no while	É uma média entre todas as entradas possíveis	O algoritmo realiza todas as iterações