Estruturas de Dados

Carlos Eduardo Rocha Miranda *

2022, v-1.0.0

Resumo

Este artigo trata de todos os algoritmos de ordenação estudados na disciplina de Estrutura de Dados II do IF Goiano - Campus Morrinhos. Nele está explicitado o funcionamento do algoritmo, a sua quantidade de comparações, a sua quantidade de movimentações e o seu tempo de execução.

Palavras-chaves: algoritmo. ordenação. custo.

Introdução

aqui tem minha introdução

1 Insertion Sort

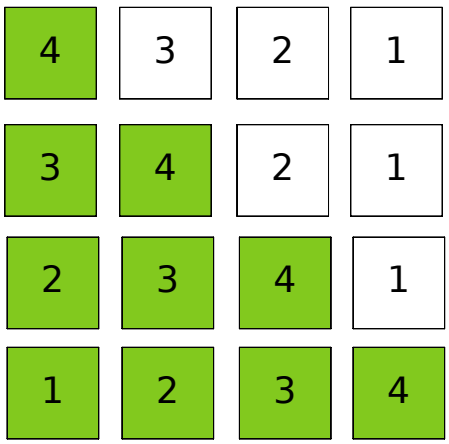
É um algoritmo de ordenação que divide os números em duas áreas. A área dos ordenados e a área dos não-ordenados.No começo a área dos ordenados possui apenas um valor, portanto está ordenada. Logo em seguida se pega o primeiro valor da área não ordenada e o comparamos com os valores presentes na área ordenada, se é maior ou menor. A depender do resultado o valor é posicionado dentro da área dos ordenados. Confira a figura 1.

*carlos.eduardo@estudante.ifgoiano.edu.br

Qtd. de Números	Tempo de Exec.	Movimentos	Comparações
5	00:00:00:00	13	9
100	00:00:00:00	2639	2540
1.000	00:00:00:00	251978	250979
10.000	00:00:00:01	25299924	25289925
50.000	00:00:01:05	625115591	625065592
100.000 (caso médio)	00:00:07:03	2504253129	2504153130
100.000 (pior caso)	00:00:12:07	4999983755	4999883756
100.000 (melhor caso)	00:00:00:00	199998	99999
500.000	00:11:47:04	62507458280	62506958281

Tabela 1 – Insertion Sort

Figura 1 – Insertion Sort



Fonte: os autores

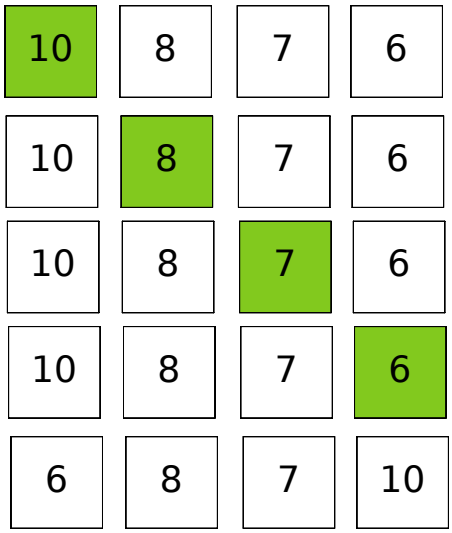
Sua implementação ocorreu sem nenhum tipo de problema. A consulta dos materiais disponibilizados pelo professor e também outros avulsos facilitaram o entendimento do algoritmo. Ele possui: no pior caso $O(n^2)$, no caso médio $O(n^2)$ e no melhor caso $O(n)$. Confira sua tabela de dados 1.

2 Selection Sort

Esse algoritmo é um pouco diferente do Insertion Sort e imita a maneira como uma criança ordena as coisas. Existe um laço de repetição que passa através dos números procurando o menor número possível. Ao encontrá-lo dentre todos, o reposiciona na primeira posição. Logo em seguida, a partir da segunda posição, ele procura o menor valor posição dentro da área dos não ordenadas e assim por diante. Confira a figura 2.

Sua implementação, semelhante ao Insertion Sort, ocorreu sem maiores problemas. Na realidade, a forma como o algoritmo foi pensado é extremamente natural e sua complexidade seguiu sendo verdadeira em sua tabela. Ele possui: no pior caso $O(n^2)$, no caso médio $O(n^2)$ e no melhor caso $O(n^2)$. Confira sua tabela de dados 2.

Figura 2 – Selection Sort



Fonte: os autores

Qtd. de Números	Tempo de Exec.	Movimentos	Comparações
5	00:00:00:00	12	29
100	00:00:00:00	297	10099
1.000	00:00:00:00	2997	1000999
10.000	00:00:00:01	29997	100009999
50.000	00:00:02:00	149997	2500049999
100.000 (caso médio)	00:00:08:05	299997	10000099999
100.000 (pior caso)	00:00:13:06	299997	10000099999
100.000 (melhor caso)	00:00:05:06	199998	10000099999
500.000	00:05:46:06	1499997	250000499999

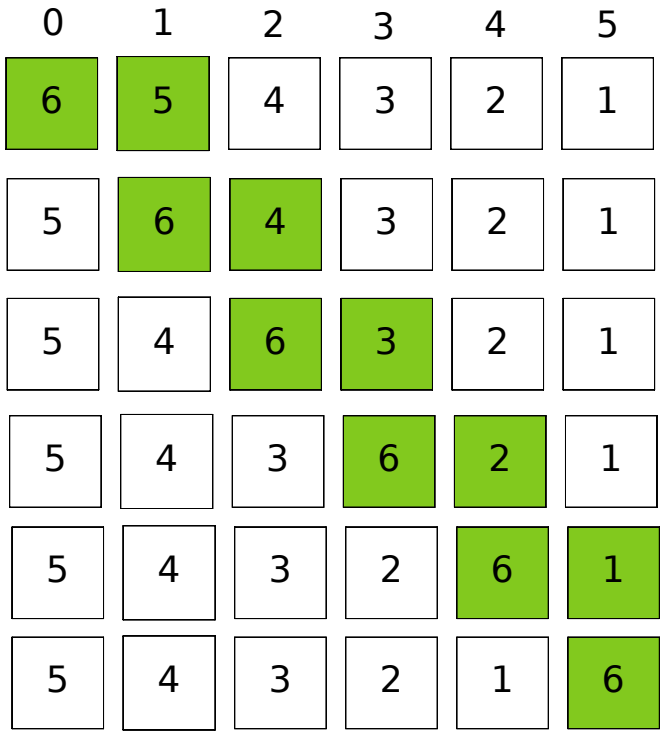
Tabela 2 – Selection Sort

3 Bubble Sort

É um algoritmo de ordenação que vai trocando de lugar os elementos adjacentes, sempre direcionando o maior valor possível para a última posição do array. Ele vai comparando cada valor procurando o maior deles. Ao achar o maior ele troca de lugar, avançando uma posição no array. Isso vai acontecer até o maior valor possível encontrar a última posição e nas repetições seguintes, o segundo maior valor, o terceiro maior valor, etc. Confira a figura 3:

Sua implementação é divertida. O ato de subir os valores mais altos para o final do array é extremamente interessante. Outra parte curiosa desse algoritmo é o fato de não haver trocas quando o array já está previamente ordenado. Ele possui: no pior caso $O(n^2)$, no caso médio $O(n^2)$ e no melhor caso $O(n)$. Confira sua tabela de dados 3.

Figura 3 – Bubble Sort



Fonte: os autores

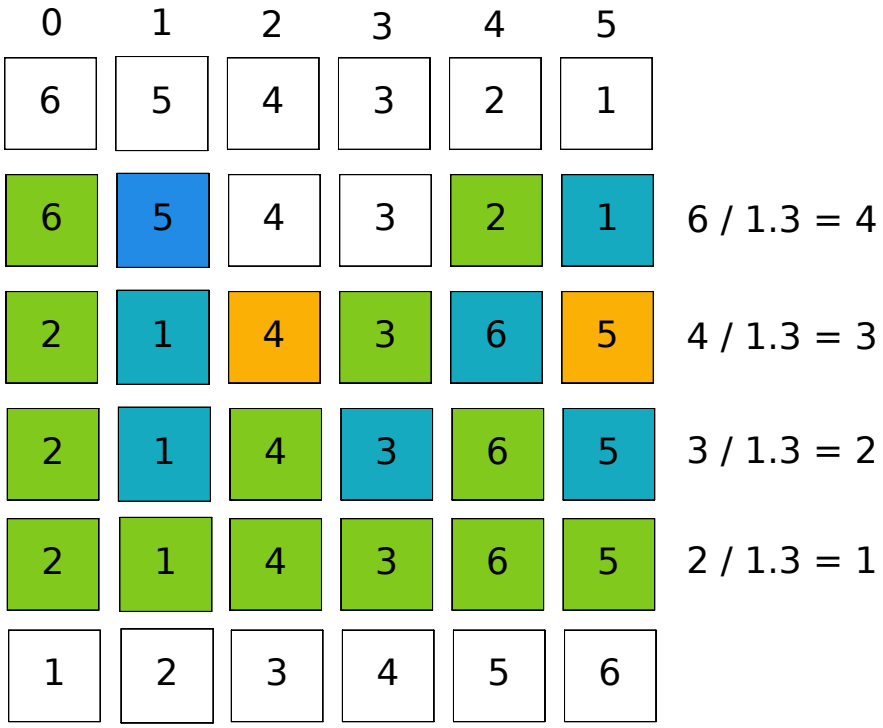
Qtd. de Números	Tempo de Exec.	Movimentos	Comparações
5	00:00:00:00	15	45
100	00:00:00:00	7323	19900
1.000	00:00:00:00	749940	1999000
10.000	00:00:00:07	75839778	199990000
50.000	00:00:23:09	1875046779	4999950000
100.000 (caso médio)	00:01:42:08	7512159393	19999900000
100.000 (pior caso)	00:02:07:02	14999351271	19999900000
100.000 (melhor caso)	00:00:05:06	0	19999900000
500.000	00:53:55:06	187519374846	499999500000

Tabela 3 – Bubble Sort

4 Combo Sort

É um algoritmo de ordenação que é uma espécie de Bubble Sort aprimorado. A ideia dele é fazer as trocas que o Bubble Sort faz de maneira espaçada, usando um passo de $h/1.3$. Onde h, no começo, é o tamanho do array. A cada repetição é refeita a conta do $h/1.3$ e portanto as espaçadas diminuem de tamanho até atingirem o valor de 1, onde ocorre, obviamente, um bubble sort comum para terminar de ordenar as partes que ainda não estão ordenados. Confira a figura 4: Sua implementação foi um pouco mais complexa. Como ele utiliza uma estratégia de divisão e conquista, as coisas nesse algoritmo se tornam mais abstratas. Ele, como o Bubble Sort, não realiza trocas em um array que já está

Figura 4 – Combo Sort



Fonte: os autores

Qtd. de Números	Tempo de Exec.	Movimentos	Comparações
5	00:00:00:00	9	30
100	00:00:00:00	777	2417
1.000	00:00:00:00	13266	43445
10.000	00:00:00:00	188601	653504
50.000	00:00:00:00	1121172	4366851
100.000 (caso médio)	00:00:00:00	2218323	8133526
100.000 (pior caso)	00:00:00:00	669249	7533529
100.000 (melhor caso)	00:00:00:00	0	7333530
500.000	00:00:00:04	11665287	48666894

Tabela 4 – Combo Sort

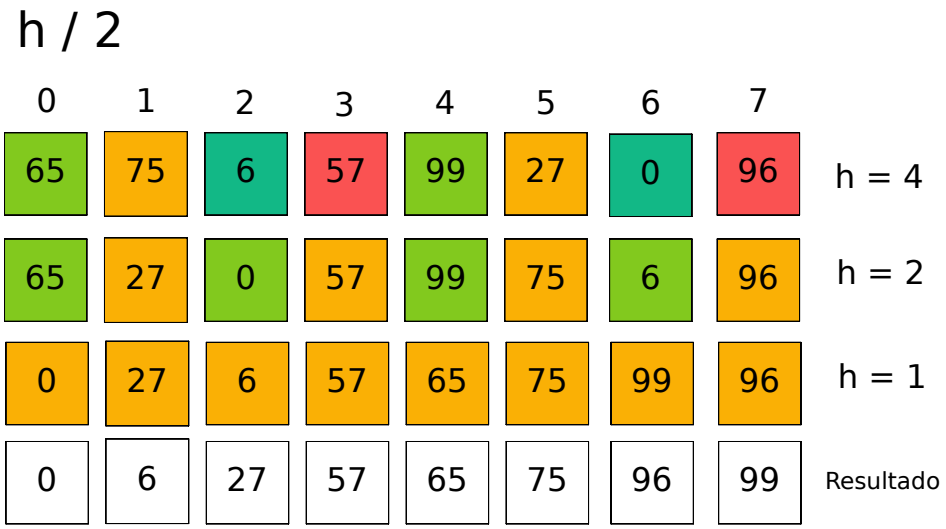
ordenado. Ele possui: no pior caso $O(n^2)$, no caso médio $O((n^2)/2^p)$ e no melhor caso $O(nlogn)$. Confira sua tabela de dados 4.

5 Shell Sort

6 Bogo Sort

É um algoritmo conhecido por sua estupidez. Ele consiste completamente na sorte. Você pega o array, verifica se está ordenado, caso não esteja você o embaralha e torce pelo

Figura 5 – Shell Sort



Fonte: os autores

Qtd. de Números	Tempo de Exec.	Movimentos	Comparações
5	00:00:00:00	11	27
100	00:00:00:00	921	1459
1.000	00:00:00:00	16326	24627
10.000	00:00:00:00	261466	381828
50.000	00:00:00:00	1762403	2453403
100.000 (caso médio)	00:00:00:00	4153791	5676702
100.000 (pior caso)	00:00:00:00	2005882	3528793
100.000 (melhor caso)	00:00:00:00	1522866	3045777
500.000	00:00:00:06	27340478	36309097

Tabela 5 – Shell Sort

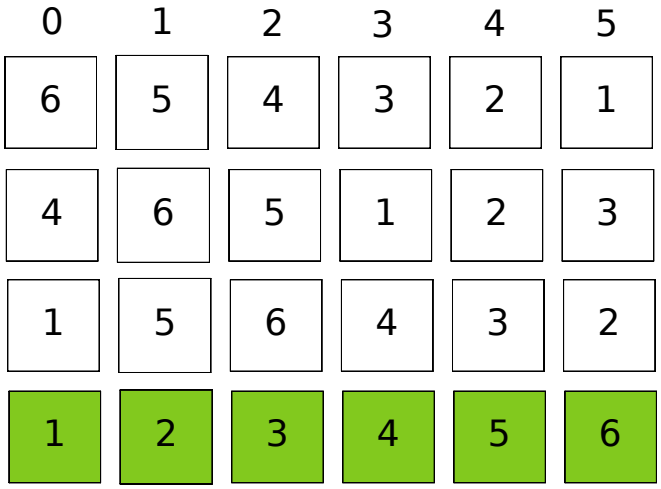
melhor. Ele é totalmente ruim e depende do acaso para o seu bom funcionamento. Confira a figura 6.

Sua implementação, dentre todos até agora, foi a mais trabalhosa. Isso é contra intuitivo, mas o problema era que o método de embaralhamento utilizado havia sido implementado de modo errado, por conta disso o algoritmo nunca funcionava. Ele possui: no pior caso $O((n+1)!)$, no caso médio $O((n+1)!)$ e no melhor caso $O(n)$. Confira sua tabela de dados 6.

7 Quick Sort

8 Merge Sort

Figura 6 – Bogo Sort



Fonte: os autores

Qtd. de Números	Tempo de Exec.	Movimentos	Comparações
5	00:00:00:00	1350	966
100	-	-	-
1.000	-	-	-
10.000	-	-	-
50.000	-	-	-
100.000 (caso médio)	-	-	-
100.000 (pior caso)	-	-	-
100.000 (melhor caso)	-	-	-
500.000	-	-	-

Tabela 6 – Bogo Sort

Qtd. de Números	Tempo de Exec.	Movimentos	Comparações
5	00:00:00:00	24	20
100	00:00:00:00	777	946
1.000	00:00:00:00	10140	14476
10.000	00:00:00:00	124275	198895
50.000	00:00:00:00	693462	1185042
100.000 (caso médio)	00:00:00:00	1438929	2588880
100.000 (pior caso)	00:00:00:00	659244	2270969
100.000 (melhor caso)	00:00:00:00	512859	2235539
500.000	00:00:00:02	7649553	16940870

Tabela 7 – Quick Sort