Universidade Federal de Viçosa
Campus Rio Paranaíba

Diego William Lima Queiroz - 5222

Relatório de Análise sobre o algoritmo Insertion Sort

Rio Paranaíba - MG
2022

Universidade Federal de Viçosa
Campus Rio Paranaíba

Diego William Lima Queiroz - 5222

Relatório de Análise sobre o algoritmo Insertion Sort

Trabalho apresentado para obtenção de créditos na disciplina SIN213 Projeto de Algoritmos da Universidade Federal de Viçosa - Campus de Rio Paranaíba, ministrada pelo Professor: Pedro Moises de Souza.

Rio Paranaíba - MG
2022

RESUMO

Algoritmo de ordenação, em ciência da computação, é um algoritmo que coloca os elementos de uma determinada sequência ”desorganizada” em uma determinada ordem. Em outras palavras, ele realiza sua ordenação completa ou parcial. O objetivo da classificação é facilitar a recuperação de dados de uma lista, ou organizar e delimitar dados.

Neste primeiro trabalho da disciplina SIN213 Projeto de Algoritmos, ministrado pelo professor Pedro Moises de Souza faremos uma análise comparativa do algoritmo Insertion Sort, dentre três possíveis tipos de ordenações, aleatório(randômico), crescente e decrescente.

:::

Sumário

Introdução_____1
Insertion Sort_____1.1

Custo do Algoritmo_____2

Conclusão_____3

Referências_____4

Função usada no projeto_____5

1 INTRODUCÃO

Em várias situações do nosso dia-a-dia nos deparamos com a necessidade de trabalharmos com dados/informações devidamente ordenadas, como, por exemplo, ao procurar um contato na lista Telefônica, imagine como seria difícil se estes nomes não estivessem em ordem alfabética? Então não é difícil perceber que as atividades que envolvem algum método de ordenação estão muito presentes na computação.

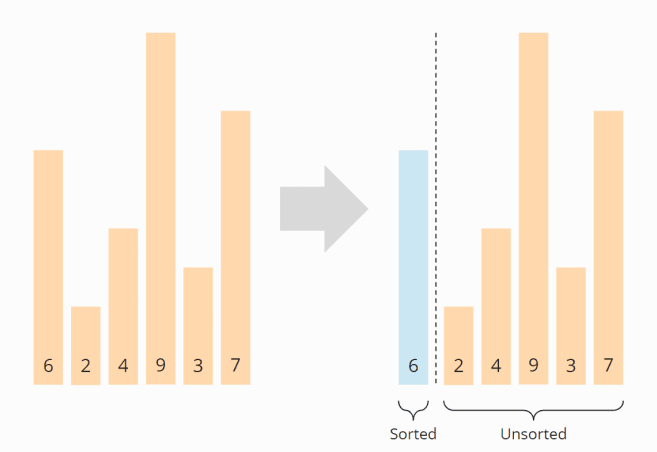
Um algoritmo que pode ser usado para a ordenação de tais problemas de dados é o algoritmo Insertion Sort. Podemos sumarizar seu funcionamento da seguinte maneira, imagine que você está ordenando um baralho de cartas, porém você só pode mover uma carta por vez a movendo ou para esquerda ou para a direita (dependendo da sua escolha sendo de ordenação crescente ou decrescente).

Fonte: https://www.happycoders.eu/wp-content/uploads/2020/05/Insertion_Sort_Playing_Card_Example

Este é o funcionamento do algoritmo, ele move cada nó de um a um garantindo que no final da ordenação a lista estará organizada.

1.1 Insertion Sort

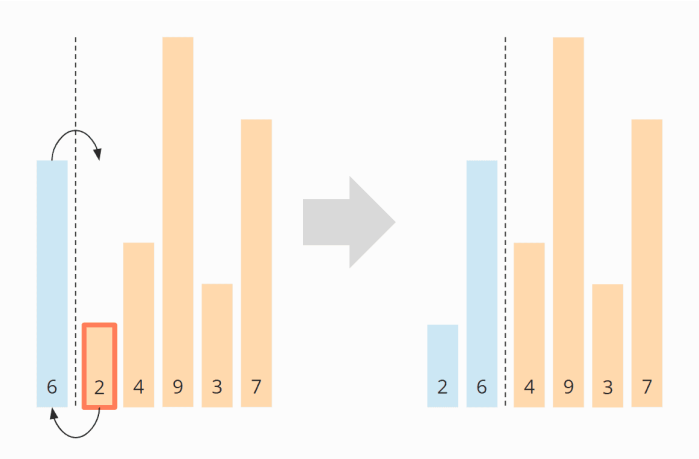
Podemos dividir o passo a passo do algoritmo em 6 partes diferentes, tais como:
1- Primeiro, dividimos o array em uma parte esquerda, ordenada, e uma parte direita, não ordenada. A parte ordenada já contém o primeiro elemento no início, pois um array com um único elemento sempre pode ser considerado ordenado.



Fonte: https://www.happycoders.eu/wp-content/uploads/2020/05/

2- Em seguida, olhamos para o primeiro elemento da área não classificada e verificamos onde, na área classificada, ele precisa ser inserido comparando-o com seu vizinho esquerdo.

No exemplo, o 2 é menor que o 6, então pertence à sua esquerda. Para abrir espaço, movemos o 6 uma posição para a direita e depois colocamos o 2 no campo vazio. Em seguida, movemos a borda entre a área classificada e não classificada um passo para a direita:

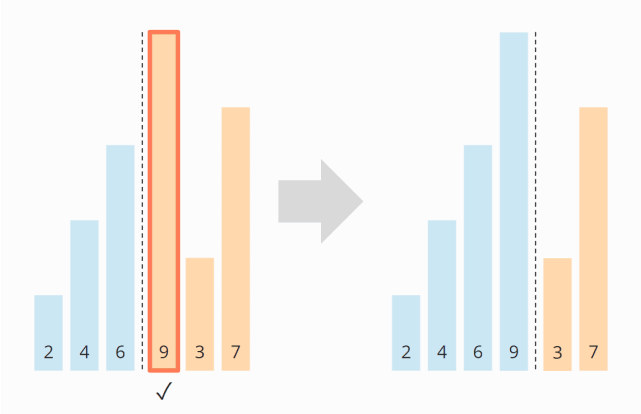


Fonte: https://www.happycoders.eu/wp-content/uploads/2020/05/

3- Olhando novamente para o primeiro elemento da área não classificada, o 4. Ele é menor que o 6, mas não menor que o 2 e, portanto, pertence entre o 2 e o 6. Então, movemos o 6 novamente uma posição para o direita e coloque o 4 no campo vago:

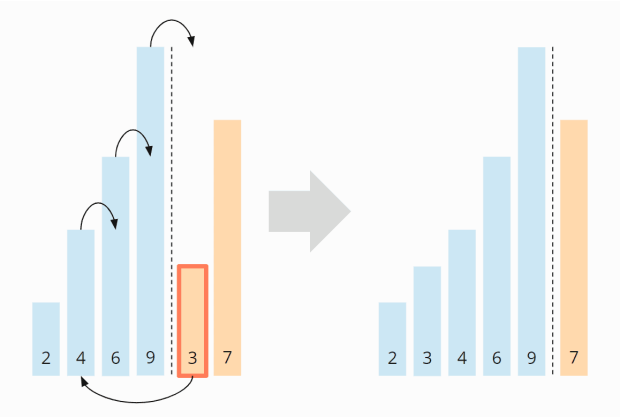
Fonte: https://www.happycoders.eu/wp-content/uploads/2020/05/

4- O próximo elemento a ser ordenado é o 9, que é maior que seu vizinho esquerdo 6 e, portanto, maior que todos os elementos na área ordenada. Portanto, ele já está na posição correta, portanto, não precisamos deslocar nenhum elemento nesta etapa:



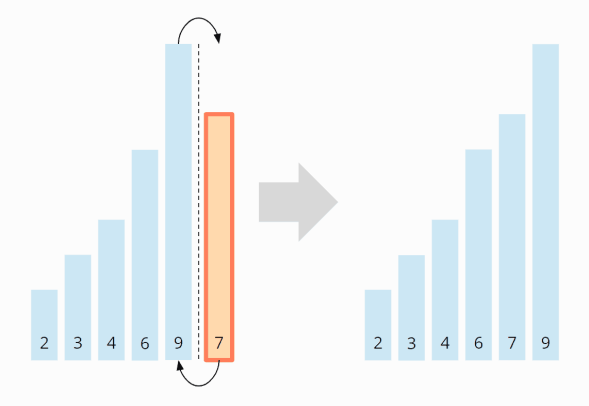
Fonte: https://www.happycoders.eu/wp-content/uploads/2020/05/

5- O próximo elemento é o 3, que é menor que o 9, o 6 e o 4, mas maior que o 2. Então movemos o 9, 6 e 4 uma posição para a direita e então colocamos o 3 onde o 4 estava antes :



Fonte: https://www.happycoders.eu/wp-content/uploads/2020/05/

6- Isso deixa o 7 – é menor que o 9, mas maior que o 6, então movemos o 9 um campo para a direita e colocamos o 7 na posição vaga:



Fonte: https://www.happycoders.eu/wp-content/uploads/2020/05/

2 Custo do algoritmo

No campo da ciência da computação, a notação Big O é uma estratégia para medir a complexidade do algoritmo. Big O é uma função que é definida em termos de entrada. A letra 'n' geralmente representa o tamanho da entrada para a função. Simplificando, n representa o número de elementos em uma lista. Em diferentes cenários, os profissionais se preocupam com o pior caso, o melhor caso ou a complexidade média de uma função.

A complexidade do pior caso (e do caso médio) do algoritmo de ordenação por inserção é $O(n^2)$. Ou seja, na pior das hipóteses, o tempo necessário para ordenar uma lista é proporcional ao quadrado do número de elementos da lista.

A complexidade de tempo do melhor caso do algoritmo de ordenação por inserção é a complexidade de tempo O(n). Significando que o tempo necessário para ordenar uma lista é proporcional ao número de elementos na lista; este é o caso quando a lista já está na ordem correta. Há apenas uma iteração neste caso, pois a operação do loop interno é trivial quando a lista já está em ordem.

Utilizando o Big O para calular o custo do algoritmo Insertion Sort temos que este é $O(n \log n)$*.

3 Conclusão

Como podemos observar nos cálculos e gráficos em questão o algoritmo insertion Sort apresenta dificuldade quando o número de células necessários na sua tarefa cresce, tendo um aumento em progressão geométrica em relação aos seus valores menores, como no exemplo:

Fonte: próprio Autor

Quando calculamos os valores de 10 a 100.000 o número de segundos aumenta em poucas unidades, porém quando o valor sobe acima disso o número de segundo necessários sobe vertiginosamente.

Isso é especialmente notado quando analisamos a função decrescente, que tem o pior desempenho dentre as demais, demorando cerca de 1 hora para o processo de ordenação ser finalizado.

Fonte: próprio Autor

:::

4 REFERÊNCIAS BIBLIOGRAFICAS

[1] BENDER, Michael A.; FARACH-COLTON, Martin; MOSTEIRO, Miguel A. Insertion sort is O (n log n). **Theory of Computing systems**, v. 39, n. 3, p. 391-397, 2006.

[2] MOHAMMED, Adnan Saher; AMRAHOV, Şahin Emrah; ÇELEBI, Fatih V. Bidirectional Conditional Insertion Sort algorithm; An efficient progress on the classical insertion sort. **Future Generation Computer Systems**, v. 71, p. 102-112, 2017.

[3] JADOON, Sultanullah; SOLEHRIA, Salman Faiz; QAYUM, Mubashir. Optimized selection sort algorithm is faster than insertion sort algorithm: a comparative study. **International Journal of Electrical & Computer Science IJECS-IJENS**, v. 11, n. 02, p. 19-24, 2011.

[4] M. A. Bender, E. D. Demaine, and M. Farach-Colton. Cache-oblivious B-trees. In Proc. 41st IEEE Ann. Symp. on Foundations of Computer Science, pages 399–409, 2000.

5 Função Usada no projeto

```
void InsertionSort (int vet[], lista *l, int tamanho){ //Algoritmo de ordenação insertion sort.

    int n = tamanho, i, j, key; // = l->tam
    time_t t_ini, t_fim;
    t_ini = time(NULL);

    for(j=1; j<=n; j++){
        key = vet[j];
        i = j - 1;
        while(i>=0 && vet[i]>key){
            vet[i+1] = vet[i];
            i = i - 1;
        }
        vet[i+1]=key;
    }
    t_fim = time(NULL);
    l->tempo = difftime(t_fim, t_ini);
}
```