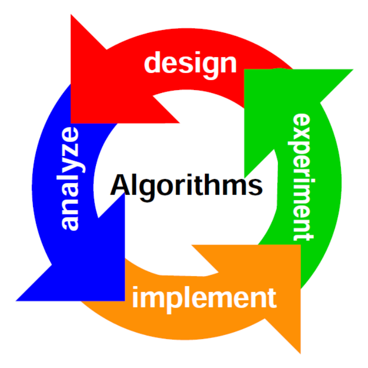
Finitude, Corretude & Complexidade de Algoritmos

Oseas Andre

25/01/2025

Table of contents

Introdução


design
experiment
Algorithms
analyze
implement

Selection Sort

Selection Sort

O Selection Sort é um dos algoritmos clássicos para ordenação de vetores. A ideia fundamental do Selection Sort é ir selecionado o menor(supondo ordem crescente) elemento do sub vetor. E colocar na posição correta.

Selection Sort - Algoritmo

Algorithm 1: Selection Sort, Adaptação do algoritmo “Selection Sort” do livro [1, p. 95].

```
selectionsort(n, A)
    var i, j, menor : integer;
    for i ←1 to n do
        menor ← i;
        for j ← i + 1 to n do
            if  A[j] < A[menor] then
                menor ← j;
            end
        end
        swap(A[menor], A[i]);
    end
end
```

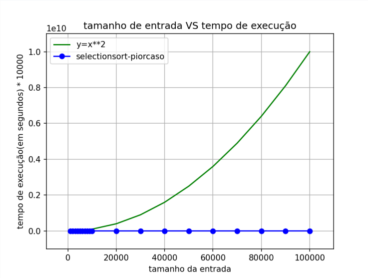
Selection Sort, Finitude e Corretude

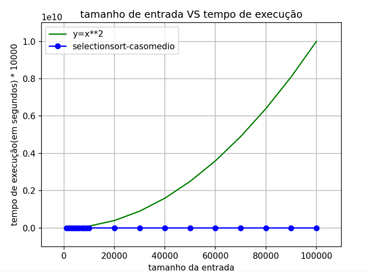
- Finitude
  1. o laço externo executa exatamente n vezes;
  2. o laço interno executa exatamente $n-1$ vezes;
  3. como todos os laços executam um número finito de vezes, esse algoritmo é finito.
- Corretude
  1. - Invariante: Invariante: No final da i-ésima iteração do laço externo o sub vetor A[1...i] está ordenado. E o i-ésimo elemento está na posição correta da permutação A[1...n] ordenada.
  2. O algoritmo é finito; o laço executa progressivamente de 1 até n iterações do laço externo.

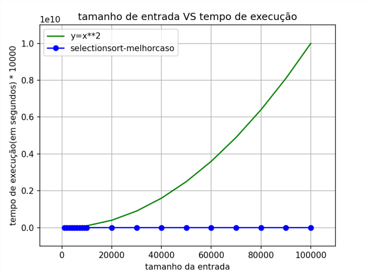
Selection Sort, Complexidade

...	Complexidade
Pior caso	$O(n^2)$
Caso Médio	$O(n^2)$
Melhor Caso	$O(n^2)$

Selection Sort, resultado dos testes







Bubble Sort

Bubble Sort

O Bubble Sort é outro dos algoritmos clássicos para ordenação de vetores. Ele é baseado em repetidas comparações e trocas de elementos adjacentes. A ideia principal é que os elementos menores flutuem para o final da lista a cada passagem, como bolhas subindo na água.

Bubble Sort - Algoritmo

Algorithm 2: Bubble Sort, *Adaptação do algoritmo "Bubble Sort" do livro [1, p. 106].

```
bubblesort(n, A)
    var j, swapped : integer;
    repeat
        swapped ← 0;
        for j ← 2 to n do
            if A[j − 1] > A[j] then
                swap(A[j − 1], A[j]);
                swapped ← 1;
            end
        end
    until swapped = 0;
end
```

Bubble Sort, Finitude

Invariante: No final da k-ésima iteração do laço externo, linhas 3 a 11, o k-ésimo maior elemento da permutação ordenada está na posição correta. Portanto no final da k-ésima iteraçao o sub vetor $A[(n-k)+1...n]$ está ordenado.

Bubble Sort, Finitude

1. No inicio de cada iteração do laço externo *swapped* $= 0$,
2. A cada iteração do laço externo, o laço interno, linhas 5 a 10, percorre subvetor $A[2...n]$. E compara os valores dos índices $j-1$ e j. Se $A[j-1]$ for maior que $A[j]$ o algoritmo troca o valor desses índices, colocando o valor de um no outro. Assim, "empurrando" o valor do k-ésimo maior elemento do vetor para a direita,
3. Quando $A[j-1]$ é maior que $A[j]$ o algoritmo também atribui 1 para a variável de controle *swapped*, indicando que houve alguma troca,
4. A condição de parada do laço externo é *swapped* $= 0$, quado em uma iteração do laço externo não forem feitas nenhuma troca,
5. Sabemos que no final da n-ésima iteração o n-ésimo maior elemento do vetor, vai estar na posição correta da permutação ordenada,
6. Portanto na $n+1$ iteração do laço externo, nenhuma troca vai ser feita. E no fim dessa iteração *swapped* $= 0$. De 3) sabemos que o laço externo termina.
7. De 1), 2), 3), 4), 5) e 6) concluímos que o "bubble sort" é finito, pois todos os seus laços executam um número finito de vezes.

Bubble Sort, Corretude

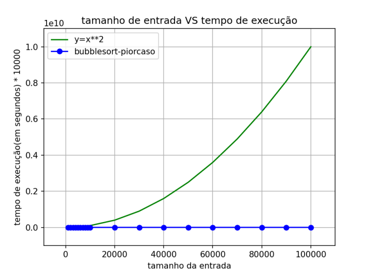
Prova por indução

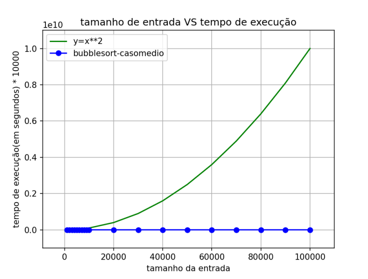
1. caso base, primeira iteração do laço externo: de 2) sabemos que o sub vetor $A[n...n]$ está ordenado.
2. Hipótese de indução: supondo que o invariante vale para uma iteração i qualquer.
3. Passo indutivo, mostrar que vale para i + 1: No início da $i+1$-ésima iteração o sub vetor $A[(n-i)+1...n]$ está ordenado. De 3.1.2) sabemos que o $i+1$-ésimo elemento do vetor vai estar na posição correta da permutação ordenada. Portando no final da $i+1$-ésima iteração o sub vetor $A[(n-i...n)]$ está ordenado.
4. Conclusão: De 3.1 sabemos que o laço externo tem um número de iterações progressiva de 1 até $n+1$. Portanto de 1), 2) e 3) temos que na n-ésima iteração o vetor $A[1...n]$ está ordenado. Então concluímos que o "Bubble sort" é correto.

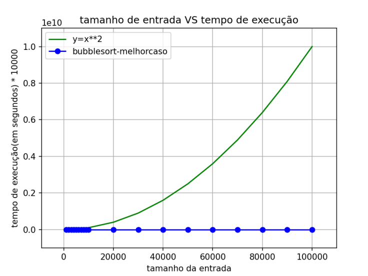
Bubble Sort, Complexidade

...	Complexidade
Pior caso	$O(n^2)$
Caso Médio	$O(n^2)$
Melhor Caso	$O(n^2)$

Bubble Sort, resultado dos testes







Insertion Sort

Insertion Sort

A ideia por trás do algoritmo Insertion Sort é simples. Consiste em organizar os elementos de uma lista ou vetor de maneira semelhante a como as pessoas ordenam cartas em suas mãos. O algoritmo constrói a ordenação gradualmente, inserindo cada elemento no lugar correto numa parte da lista que já está ordenada.

Insertion Sort - Algoritmo

Algorithm 3: Insertion Sort

insertionsort(n, A)
    `var` i`,` j`,` v `:  integer`;
    for $i \leftarrow 2$ **to** n **do**
        $v \leftarrow A[i]$;
        $j \leftarrow i$;
        while $v < A[j-1]$ **do**
            $A[j] \leftarrow A[j-1]$;
            $j \leftarrow j-1$;
        end
        $A[j] \leftarrow v$;
    end
end

Insertion Sort, Finitude

- Finitude
  1. O laço externo executa exatamente n-1 vezes.
  2. O número de execuçoes do laço interno depende do conteúdo do vetor. Esse laço é responsável por empurrar os elementos do vetor para a direita.
  3. Suponha um índice j qualquer se os elementos do sub vetor $A[1...j-1]$ são todos maiores que v o laço interno é executado j vezes. Caso contrario, não são todos maiores que v o laço interno é executado k ¡ j vezes também um número finito.
  4. De 1), 2) e 3) concluímos que esse algoritmo é finito, pois todos os laços executam um número finito de vezes.

Insertion Sort, Corretude

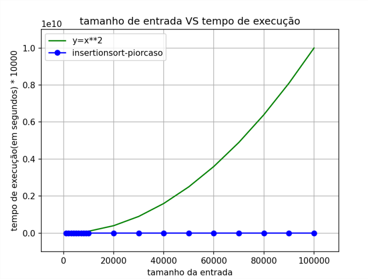
Invariante: No final da k-ésima iteração do laço externo, o sub vetor $A[1...k+1]$ está ordenado. **Prova por indução**

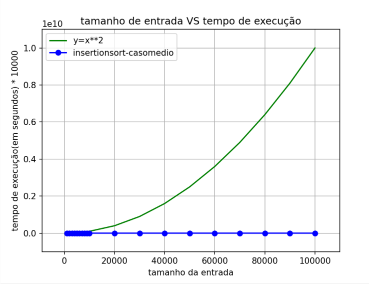
1. caso base: lista com um elemento, trivial, já ordenada. Lista com dois elementos(primeira iteração):
    1.1 O algoritmo considera o sub vetor $A[1...1]$ já ordenado e faz a inserção do valor no índice 2.
    1.2 As linhas de 7, 8 e 9 garantem que no final da iteração o sub vetor $A[1...2]$ está ordenado.
2. Hipótese de indução: supondo que o invariante vale para a $i-1$-ésima iteração qualquer, i ¿ 1.
3. Passo indutivo, mostrar que vale para (i-1)+1: Se vale para $i-1$-ésima iteração então no final dela o sub vetor $A[1...i]$ está ordenado. Na i-ésima iteração (inserção de um elemento no sub vetor ordenado $A[1...i]$) as linhas 7 a 9 garantem que no final do sub vetor $A[1...i+1]$ está ordenado.
4. Com exatamente $n-1$ iterações, no final dessa o sub vetor $A[1...n]$ está ordenado, concluímos então que esse é um algoritmo correto.

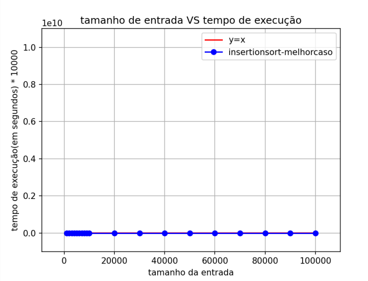
Insertion Sort, Complexidade

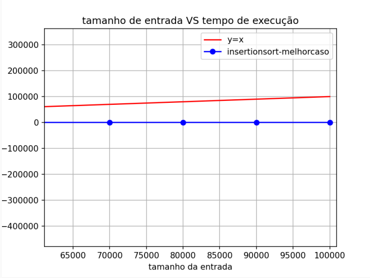
...	Complexidade
Pior caso	$O(n^2)$
Caso Médio	$O(n^2)$
Melhor Caso	$O(n)$

Insertion Sort, resultado dos testes









Conclusão

Dos gráficos fica evidente a confirmação dos resultados teóricos. O limite superior de todos os algoritmos analisados foi satisfeito nos testes, considerando o tempo "real" de execução da implementação do algoritmo.

Dentre os três, o "selection sort" foi o que se mostrou mais estável entre os cenários de pior, melhor e caso médio. O "bubble sort" demonstrou com portamento exponencial no pior caso e, no caso médio mas no melhor caso mostrou comportamento relativamente "linear".

O "Insertion sort" mostrou comportamento exponencial no pior caso, no caso médio também, mas no no melhor caso mostrou comportamento relativamente linear.

References

[1] Sedgewick Robert. *ALGORITHMS*. Addison-Wesley Publishing Company Inc, 1983.