UNIVERSIDADE
FEDERAL DO CEARÁ

Algoritmos Fundamentais para Ordenação (quadráticos)

TÓPICOS ESPECIAIS EM COMPUTAÇÃO II e ESTUDOS ESPECIAIS

UFC – Engenharia da Computação – 2023-2

Prof. Fischer Jônatas Ferreira

Trabalho esquenta - busca sequencial

```java
public static int buscaSequencial_v1(int x, int v[]) {

    int indice=-1;
    for (int i = 0; i < v.length; i++) {
        if (v[i] == x)
            indice = i;
    }
    return indice;
}
```

```java
public static int buscaSequencial_v2(int x, int v[]) {

    for (int i = 0; i < v.length; i++) {
        if (v[i] == x)
            return i;
    }
    return -1;
}
```

Trabalho esquenta - quadrática e cúbica

```
for (int i = 0; i < vet.length; i++) {
    for (int j = i; j < vet.length; j++) {
        if (vet[i] == numeroProcurado) {
            if (!entrou) {
                posicao = i;
                if (vet[j] == numeroProcurado) {
                    contador++;
                }
            }
        }
    }
    if (contador > 0)
        entrou = true;
}
if (entrou)
    System.out.println("Posicao: " + posicao + " - contador de repeticao: " + contador);
```

```
for (int i = 0; i < vet.length; i++) {
    for (int j = 0; j < vet.length; j++) {
        for (int l = 0; l < vet.length; l++) {
            if (vet[i] == numeroProcurado &&
                    vet[j] == numeroProcurado &&
                        vet[l] == numeroProcurado)
                posicao = i;
        }
    }
}
if (posicao != -1)
    System.out.println("Posicao: " + posicao);
```

Trabalho esquenta - busca binária

```
public static int PesquisaBinaria(int x, int v[], int e, int d) {
    int meio = (e + d) / 2;
    if (v[meio] == x)
        return meio;
    if (e >= d)
        return -1;
    else if (v[meio] < x)
        return PesquisaBinaria(x, v, meio + 1, d);
    else
        return PesquisaBinaria(x, v, e, meio - 1);
}
```

Qual é a complexidade?

```
1  Seja M[0..n][0..W] uma matriz
2  para X = 0 até W, incrementando faça
3      M[0][X] = 0
4  para j = 0 até n, incrementando faça
5      M[j][0] = 0
6  para j = 1 até n, incrementando faça
7      para X = 0 até W, incrementando faça
8          se w_j > X então
9              M[j][X] = M[j − 1][X]
10         senão
11             usa = v_j + M[j − 1][X − w_j]
12             naousa = M[j − 1][X]
13             M[j][X] = max{usa, naousa}
14 devolve M[n][W]
```

Qual é a complexidade?

n = 4, W = 7,

w1 = 1, v1 = 10,
w2 = 3, v2 = 40,
w3 = 4, v3 = 50,
w4 = 5 e v4 = 70

Algoritmo 22.13: MochilaInteira-BottomUp(n, W)

```
1  Seja M[0..n][0..W] uma matriz
2  para X = 0 até W, incrementando faça
3      M[0][X] = 0
4  para j = 0 até n, incrementando faça
5      M[j][0] = 0
6  para j = 1 até n, incrementando faça
7      para X = 0 até W, incrementando faça
8          se w_j > X então
9              M[j][X] = M[j − 1][X]
10         senão
11             usa = v_j + M[j − 1][X − w_j]
12             naousa = M[j − 1][X]
13             M[j][X] = max{usa, naousa}
14 devolve M[n][W]
```

item ↓ \ capacidade →	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1, $v_1 = 10$, $w_1 = 1$	0	10	10	10	10	10	10	10
2, $v_2 = 40$, $w_2 = 3$	0	10	10	40	50	50	50	50
3, $v_3 = 50$, $w_3 = 4$	0	10	10	40	50	60	60	90
4, $v_4 = 70$, $w_4 = 5$	0	10	10	40	50	70	80	90

Ordenação

A ordenação tem por objetivo rearrumar as chaves do vetor (ou outra estrutura de dados qualquer) de forma que os elementos obedeçam a uma regra (ex.: ordem numérica).

Entrada: $\langle A, n \rangle$, onde $A = (a1, a2, \ldots, an)$ é um vetor com n números.
Saída: Permutação $(a'1, a'2, \ldots, a'n)$ dos números de A de modo que $a'1 \leq a'2 \leq \cdots \leq a'n$

Ordenação

- Existem diversos algoritmos que resolvem o problema de ordenação.
- A entrada geralmente é um vetor com n elementos,
- Existem diversas razões para a ordenação ser considerada um dos problema mais fundamental no estudo de algoritmos:
  - É dos problemas mais básicos, mas é dos mais estudados
  - A necessidade de ordenar é inerente a muitos problemas
  - A ordenação frequentemente é utilizada como sub-rotina chave
  - Muitas heurísticas utilizam a ordenação para fornecer boas soluções
  - Existe uma ampla variedade de algoritmos de ordenação que empregam um rico conjunto de técnicas
- É possível demonstrar um limite inferior para a ordenação.

Ordenação

Tipos de algoritmos de ordenação segundo a sua classe de complexidade:

Classe	Algoritmo	Complexiade
Quadrática	Insertion sort	Melhor caso: On Pior caso: On^2
Quadrática	Selection sort	Melhor caso: On^2 Pior caso: On^2
Logarítmica	Merge sort	Melhor caso: $O(nlogn)$ Pior caso: $O(nlogn)$
Logarítmica	Quicksort	Melhor caso: $O(nlogn)$ Pior caso: On^2

Insertion Sort

- O Insertion sort é eficiente para ordenar:
  - Um número pequeno de elementos
  - Se a entrada tiver muitos elementos ou todos elementos já ordenados
- Complexidade:
  - **Melhor Caso**: O(n) quando a entrada já esteja ordenada
  - **Pior Caso**: O(n^2) quando a entrada não ordenada

- Algoritmo estável: números com o mesmo valor aparecem no arranjo de saída na mesma ordem em que se encontram no arranjo de entrada
- Algoritmo in-place: ordena o vetor de entrada sem usar um vetor auxiliar

Insertion Sort - algoritmo

- O Insertion sort é eficiente para ordenar um número pequeno de elementos
- Considera que o primeiro elemento está ordenado (ou seja, na posição correta)
- A partir do segundo elemento, insere os demais elementos na posição apropriada entre aqueles já ordenados
- O elemento é inserido na posição adequada movendo-se todos os elementos maiores para a posição seguinte do vetor

Insertion Sort

- Dado um vetor A[1..n] com n números, a idéia da ordenação por inserção é executar n rodadas de instruções onde:
  - a cada rodada temos um subvetor de A ordenado
  - subvetor contém um elemento a mais do que o subvetor ordenado da rodada anterior
- Sabendo que o subvetor A[1..i − 1] está ordenado, é fácil “encaixar” o elemento A[i] na posição correta para deixar o subvetor A[1..i] ordenado:
  - Compare A[i] com A[i − 1], com A[i − 2], e assim por diante, até encontrar um índice j tal que A[j] ≤ A[i]

Insertion Sort

```
INSERTION-SORT(A)
1 for j ← 2 to comprimento[A]
2       do chave ← A[j]
3          ▷ Inserir A[j] na seqüência ordenada A[1..j – 1].
4          i ← j – 1
5          while i > 0 e A[i] > chave
6             do A[i + 1] ← A[i]
7                i ← i – 1
8          A[i + 1] ← chave
```

Como otimizar o Insertion Sort?

Exercício:

Apresenta uma melhoria para o algoritmo Insertion Sort?

Como otimizar o Insertion Sort?

Exercício:

Apresenta uma melhoria para o algoritmo Insertion Sort?

Possibilidade: Para procurar o lugar do elemento no subvetor ordenado em vez de fazer uma busca linear fazer uma busca binária.

Bubble Sort

- O Bubble sort é um algoritmo de ordenação que percorre o vetor diversas vezes, e a cada passagem faz flutuar para o topo o maior elemento da sequência.
  - Essa movimentação lembra a forma como as bolhas em um tanque de água flutuam para a superfície.
  -
- O Bubble sort é um algoritmo estável e é in-place

Bubble Sort

- O Bubble sort é um algoritmo de ordenação que percorre o vetor diversas vezes, e a cada passagem faz flutuar para o topo o maior elemento da sequência.
- Em cada passo, é comparado cada elemento no vetor com o seu sucessor (p[i] com p[i+1]) e troca o conteúdo das posições em análise, caso não estejam na ordem desejada.
- A ideia é colocar o maior elemento em sua posição e continuar fazendo o mesmo para todos os outros elementos.
- Assim, ao fim da primeira iteração do algoritmo o maior elemento estará na posição n do vetor.
- No final da segunda iteração o segundo maior elemento estará na posição n-1.
- No final da terceira iteração o terceiro maior elemento do vetor estará na posição n-2, e assim sucessivamente.

Bubble Sort

```
BUBBLESORT(A)
1  for i = 1 to A·comprimento
2      for j = A·comprimento downto i + 1
3          if A[j] < A[j − 1]
4              then trocar A[j] com A[j − 1]
```

Bubble Sort

- Se o vetor estiver ordenado, então nenhuma troca será realizada.
- Melhor caso O(n)
- Pior caso O(n2)
- A condição de pior caso ocorre quando os elementos do vetor são organizados em ordem decrescente.

Selection Sort

- O Selection sort é um algoritmo que mantém o vetor de entrada A[1..n] dividido em dois subvetores contíguos separados por uma posição i, um à direita e outro à esquerda, estando um deles ordenado.

- O subvetor da esquerda, A[1..i], contém os menores elementos da entrada ainda não ordenados e o subvetor da direita, A[i+1..n], contém os maiores elementos da entrada já ordenados.

- A cada iteração, o maior elemento do subvetor A[1..i] é encontrado e colocado na posição i, aumentando o subvetor ordenado em uma unidade

Selection Sort

Algoritmo 17.1: SELECTIONSORT(A, n)

```
1 para i = n até 2, decrementando faça
2     iMax = i
3     para j = 1 até i − 1, incrementando faça
4         se A[j] > A[iMax] então
5             iMax = j
6     troca A[iMax] com A[i]
7 devolve A
```

Shell Sort

- O Shellsort é uma variação do Insertion sort que faz comparação de elementos mais distantes e não apenas vizinhos.
- Dizemos que um vetor está h-ordenado se, a partir de qualquer posição, considerar todo elemento a cada h posições leva a uma sequência ordenada. Por exemplo:
  - O vetor A = (1, 3, 5, 8, 4, 15, 20, 7, 9, 6)
  - Está 5-ordenado, pois as sequências de elementos, estão ordenadas
  - (1, 15),
  - (3, 20),
  - (5, 7),
  - (8, 9) e
  - (4, 6).

Shell Sort

- Já o vetor A = (1, 3, 5, 6, 4, 9, 8, 7, 15, 20) está 3-ordenado, pois são sequências ordenadas de elementos que estão à distância 3 entre si.
- (1, 6, 8, 20),
- (3, 4, 7),
- (5, 9, 15),
- (6, 8, 20),
- (4, 7),
- (9, 15) e
- (8, 20)

Shell Sort

Algoritmo 15.2: SHELLSORT(A, n, H, m)

```
1 para t = 1 até m, incrementando faça
2     para i = H[t] + 1 até n, incrementando faça
3         atual = A[i]
4         j = i − 1
5         enquanto j ≥ H[t] e A[j − H[t] + 1] > atual faça
6             A[j + 1] = A[j − H[t] + 1]
7             j = j − H[t]
8         A[j + 1] = atual
```

Vantagens dos algoritmos quadráticos para ordenação

- Implementação fácil
- Funciona muito bem para instâncias pequenas
- Pode ter um bom desempenho se a entrada estiver muitos elementos ordenados

Desvantagens dos algoritmos quadráticos para ordenação

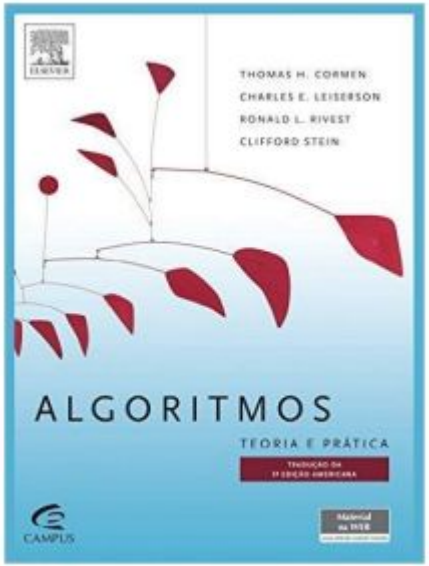
- Implementação não trivial
- Indicado para instâncias grandes

Exercício

Crie seu algoritmo quadrático para o problema da ordenação.

Bibliografia principal

Cormen, Thomas H., et al. "Algoritmos: teoria e prática." Editora Campus 2 (2002).



Bibliografia alternativa

- Dasgupta, Sanjoy, Christos Papadimitriou, and Umesh Vazirani. Algoritmos. AMGH Editora, 2009.

- Ziviani, Nivio. Projeto de algoritmos: com implementações em Pascal e C. Vol. 2. Thomson, 2004.

- Lintzmayer, Carla. Análise de Algoritmos e de Estruturas de Dados http://professor.ufabc.edu.br/~carla.negri/cursos/materiais/Livro-Analise.de.Algoritmos.pdf