Algoritmos de ordenação

Fabrício J. Barth

BandTec - Faculdade de Tecnologia Bandeirantes

Outubro de 2011

Tópicos

- Introdução e Justificativa.
- Algoritmo de ordenação bolha.
  - ⋆ Iterativo
  - ⋆ Recursivo
- *QuickSort*.

Introdução e Justificativa

- A **entrada** é um vetor cujos elementos precisam ser ordenados.
- A **saída** é o mesmo vetor com seus elementos na ordem especificada.

Algoritmo de ordenação bolha

- *Os elementos maiores são mais leves e sobem como bolhas até suas posições corretas.*
- Exemplo: ordenar $\{25, 48, 37, 12, 57, 86, 33, 92\}$

Algoritmo de ordenação bolha (iterativo)

```
1   public static int[] bolha(int [] v){
2       for(int i=v.length-1; i>=1; i--){
3           for(int j=0; j<i; j++){
4               if(v[j] > v[j+1]){
5                   int temp = v[j]; //troca
6                   v[j] = v[j+1];
7                   v[j+1] = temp;
8               }
9           }
10      }
11      return v;
12  }
```

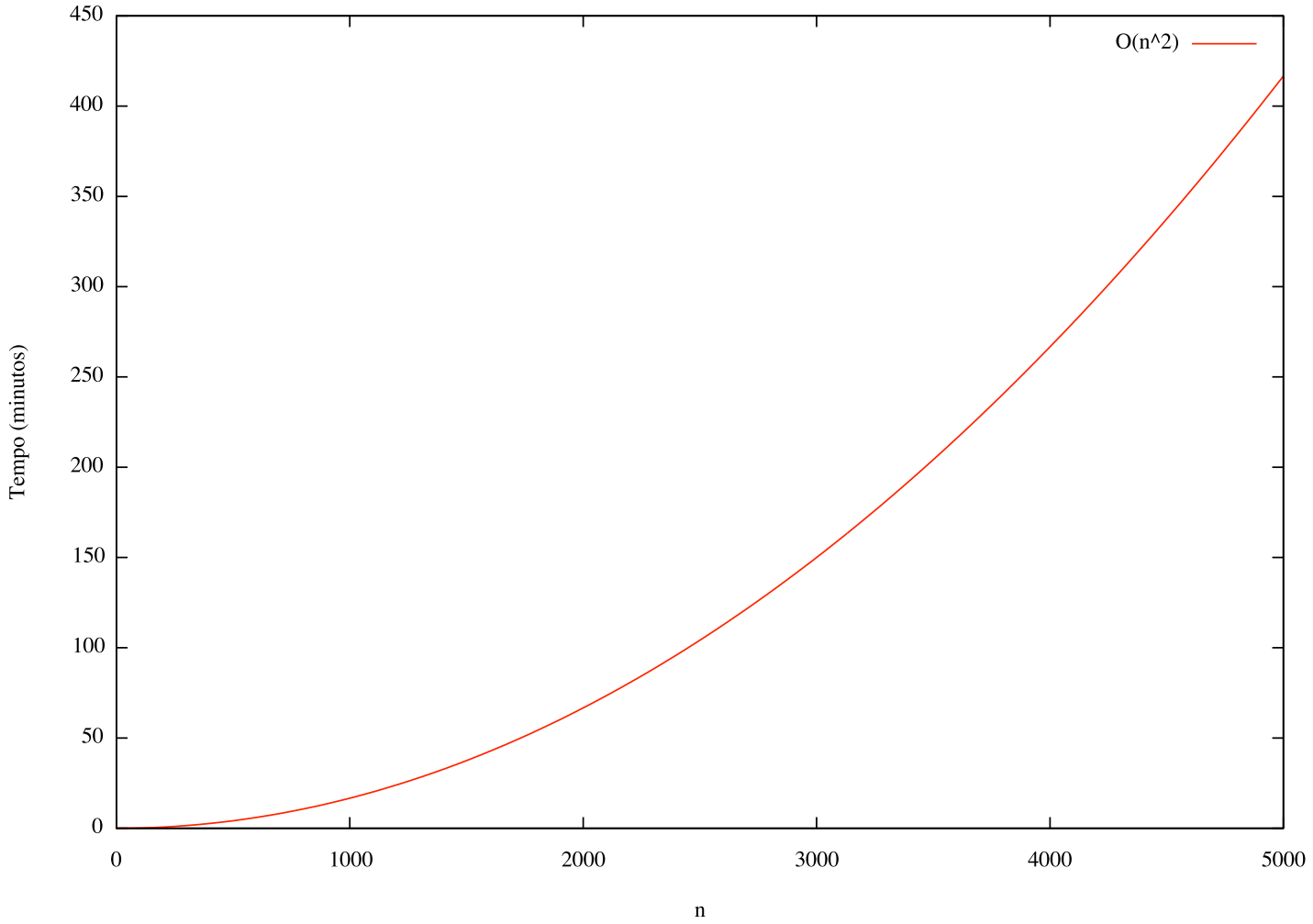
Algoritmo de ordenação bolha (c/ parada)

```
1   public static int[] bolhaCurto(int [] v){
2       for(int i=v.length-1; i>=1; i--){
3           boolean troca = false;
4           for(int j=0; j<i; j++){
5               if(v[j] > v[j+1]){
6                   int temp = v[j]; v[j] = v[j+1];
7                   v[j+1] = temp; troca = true;
8               }
9           }
10          if(!troca) return v;
11      }
12      return v;}
```

Considerações sobre o desempenho do método bolha

- Para ordenar um vetor, fazemos na primeira rodada $(n-1)$ comparações. Na segunda rodada fazemos $(n-2)$, até chegarmos em 1 comparação.
- Tempo total gasto pelo algoritmo:
  $(n-1)+(n-2)+\cdots+2+1 \simeq O(n^2)$

Digamos que cada comparação dura 1ms (milissegundo). Sendo assim, temos:



Algoritmo de ordenação bolha (recursivo)

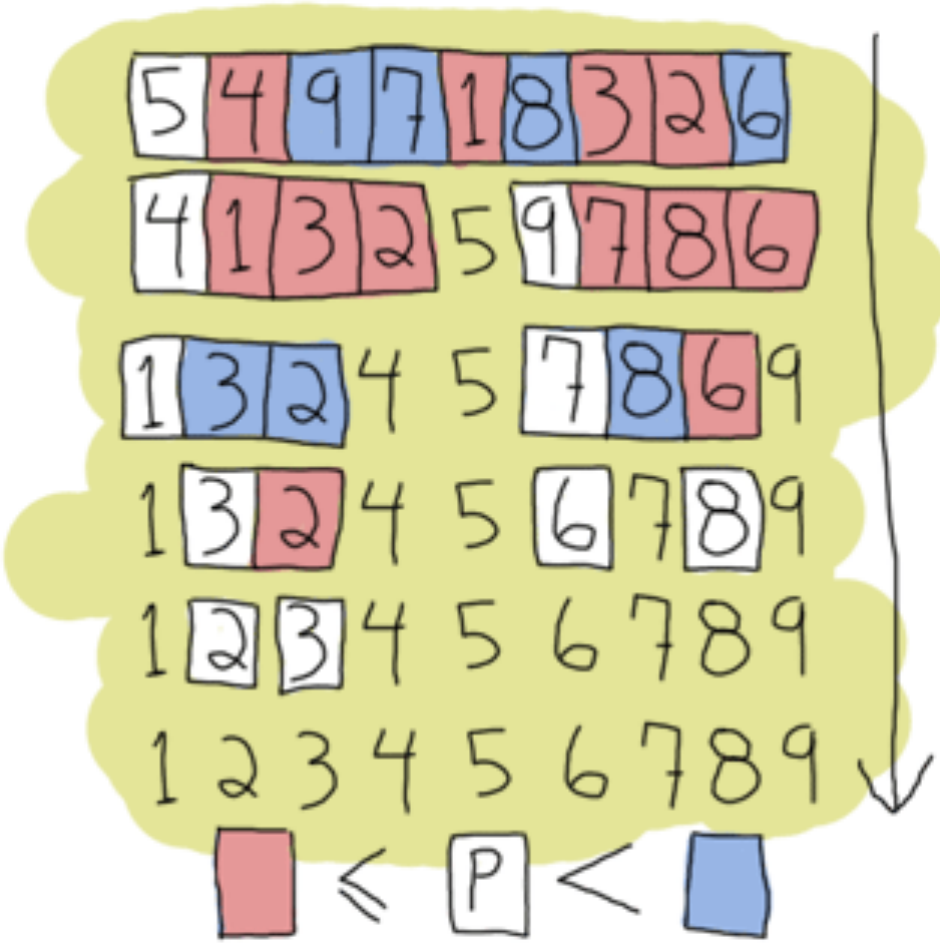
O algoritmo recursivo de ordenação bolha posiciona o elemento de maior valor e chama, recursivamente, o algoritmo para ordenar o vetor restante, com $n - 1$ elementos.

```
1   public static int[] bolhaRecursivo(int n, int [] v){
2       for(int i=n-1; i>=1; i--){
3           boolean troca = false;
4           for(int j=0; j<i; j++){
5               if(v[j] > v[j+1]){
6                   int temp = v[j]; v[j] = v[j+1];
7                   v[j+1] = temp; troca = true;
8               }
9           }
10          if(!troca) return bolhaRecursivo(n-1,v);
11      }
12      return v;}
```

Exemplo de utilização

```
1  public static void main(String[] args) {
2      int a[] = {1,7,8,5,6,4,2,3,9,63,67,3,11,25,13,14};
3      int v[] = Bolha.bolha(a);
4      for(int i=0; i<v.length; i++)
5          System.out.print(v[i]+" , ");
6  }
```

QuickSort - *Dividir para conquistar, com pivô!*



QuickSort

```
QuickSort (A, p, r)
if p < r then
    q = Partition (A, p, r)
    QuickSort (A, p, q-1)
    QuickSort (A, q+1, r)
end if
```

Para ordenar um vetor A, a chamada inicial deve ser QuickSort (A, 1, A.length).

QuickSort - Partition

```
PARTITION (A, P, R)
x = A[r]
i = p - 1
for j = p to r - 1 do
    if A[j] ≤ x then
        i = i + 1
        exchange A[i] with A[j]
    end if
end for
exchange A[i + 1] with A[r]
return  i + 1
```

QuickSort - implementação (1/3)

```
1  public static void quickSort(int A[], int p, int r){
2      if(p<r){
3          int q = partition(A,p,r);
4          quickSort(A,p,q-1);
5          quickSort(A,q+1,r);
6      }
7  }
```

QuickSort - implementação (2/3)

```
1  public static int partition(int A[], int p, int r){
2      int x = A[r]; int i = p - 1; int temp;
3      for(int j=p; j <= r-1; j++){
4          if(A[j]<=x){
5              i = i + 1; temp = A[i];
6              A[i] = A[j]; A[j] = temp;
7          }
8      }
9      temp = A[i+1]; A[i+1] = A[r];
10     A[r] = temp; return i + 1;
11 }
```

QuickSort - implementação (3/3)

```
1  public static void main(String[] args) {
2      int a[] = {1,7,8,5,6,4,2,3,9,11,25,13,14};
3      QuickSort.quickSort(a,0,a.length-1);
4      for(int i=0; i<a.length; i++)
5          System.out.print(a[i]+" , ");
6  }
```

QuickSort - escolha do pivô

- O pivô ideal é aquele que produz segmentos P e R com tamanhos aproximadamente iguais: **chave de valor mediado**.
- A identificação do **pivô ideal** requer a varredura de todo o vetor (o benefício não justifica o custo).
- Deseja-se um critério de escolha **simples** e **rápido**.
- Sem conhecimento prévio sobre a distribuição de valores das chaves, supõe-se que qualquer um possa ser o pivô e arbitra-se, por exemplo, a **primeira chave**.

Material de **consulta** e **referência**

- Capítulo 16 do livro: "Introdução a Estruturas de Dados" do Waldemar Celes, Renato Cerqueira e José Lucas Rangel.
- Capítulo 7 do livro: "Introduction to Algorithms" do Cormen, Leiserson, Rivest e Stein.
- Algumas imagens foram obtidas no site http://learnyousomeerlang.com/recursion.