UNINASSAU
Campus Boa Viagem

Estrutura de dados

PROFESSOR:

Adilson da Silva

CURSO (2023.2)

Apresentação baseada em apresentação do professor Leopoldo França

MA UNG UNINORTE UNESC UNIFAEL UNI7
Grupo Ser Educacional

UNINASSAU
Campus Graças

Algoritmos Recursivos

Recursão

UNINASSAU
Campus Graças

"Uma função ou procedimento é dito *recursivo* (ou apresenta *recursividade*) se for definido em termos de si próprio."

Recursão

UNINASSAU
Campus Graças

- Recursão é o processo pelo qual uma função chama a si mesma, repetidamente, um numero finito de vezes ;
- A recursividade é um dos elementos da programação que pode causar um certo impacto inicialmente ;
- Porém, se bem utilizado, torna os programas mais simples e elegantes;

Recursão

- Recursão direta: Uma função A chama a própria função A.

- Recursão indireta: Uma função A chama uma função B, que por sua vez, chama a função A.

Exemplo

- Um dos exemplos mais populares de recursividade direta vem da matemática, com o cálculo do fatorial de um número. O fatorial de N é definido por:

0! = 1

N! = N x (N-1)! - Para N>0

Exemplo

UNINASSAU
Campus Graças

- Fatorial
  - A definição de fatorial é:
    - F(n) = 1 se n = 0;
    - F(n) = n * F(n-1), se n>0.
    - onde n é um numero inteiro positivo. Uma propriedade (facilmente verificável) dos fatoriais é que: $n! = n * (n-1)!$

Exemplo

- Esta propriedade é chamada de propriedade recursiva: o fatorial de um numero pode ser calculado através do fatorial de seu antecessor.

    - F(4) = 4 * F(4-1)
    - F(3) = 3 * F(3-1)
    - F(2) = 2 * F(2-1)
    - F(1) = 1 * F(1-1)
    - F(0) = 1

Recursão

UNINASSAU
Campus Graças

- Algoritmo recursivo para o cálculo do fatorial.

```
funcao Fatorial (N: inteiro): inteiro
inicio
        se N = 0 entao
                retorne 1
        senao
                retorne N * Fatorial(N - 1)
        fimse
fimfuncao
```

Recursão aqui!

Caso base ou Condição de Parada

UNINASSAU
Campus Graças

Como uma função recursiva pode chamar a si mesma indefinidamente, é essencial a existência do caso base, ou condição de parada.

Recursão

- É o processo de resolução de um problema, reduzindo-o em um ou mais sub-problemas com as seguintes características:

  - idênticos ao problema original.
  - resolução mais simples e conhecida.

Forma geral

UNINASSAU
Campus Graças

- Esquematicamente, os algoritmos recursivos têm a seguinte forma:

se "**condição para o caso de base**" então
  resolução direta para o caso de base
senão
  uma ou mais chamadas recursivas
fimse

Caso Base

UNINASSAU
Campus Graças

- Um algoritmo recursivo pode ter um ou mais casos de base e um ou mais casos gerais.
- E para que o algoritmo termine, as chamadas recursivas devem convergir em direção ao caso de base, senão o algoritmo não terminará jamais.
- Convergir significa ter uma parte menor do problema para ser resolvido.

Caso Base - exemplo

UNINASSAU
Campus Graças

F(4) = 4.F(4-1)

F(3) = 3.F(3-1)

F(2) = 2.F(2-1)

F(1) = 1.F(1-1)

F(0) = 1 --- Caso Base

F(1) = 1.1

F(2) = 2.1

F(3) = 3.2

F(4) = 4.6

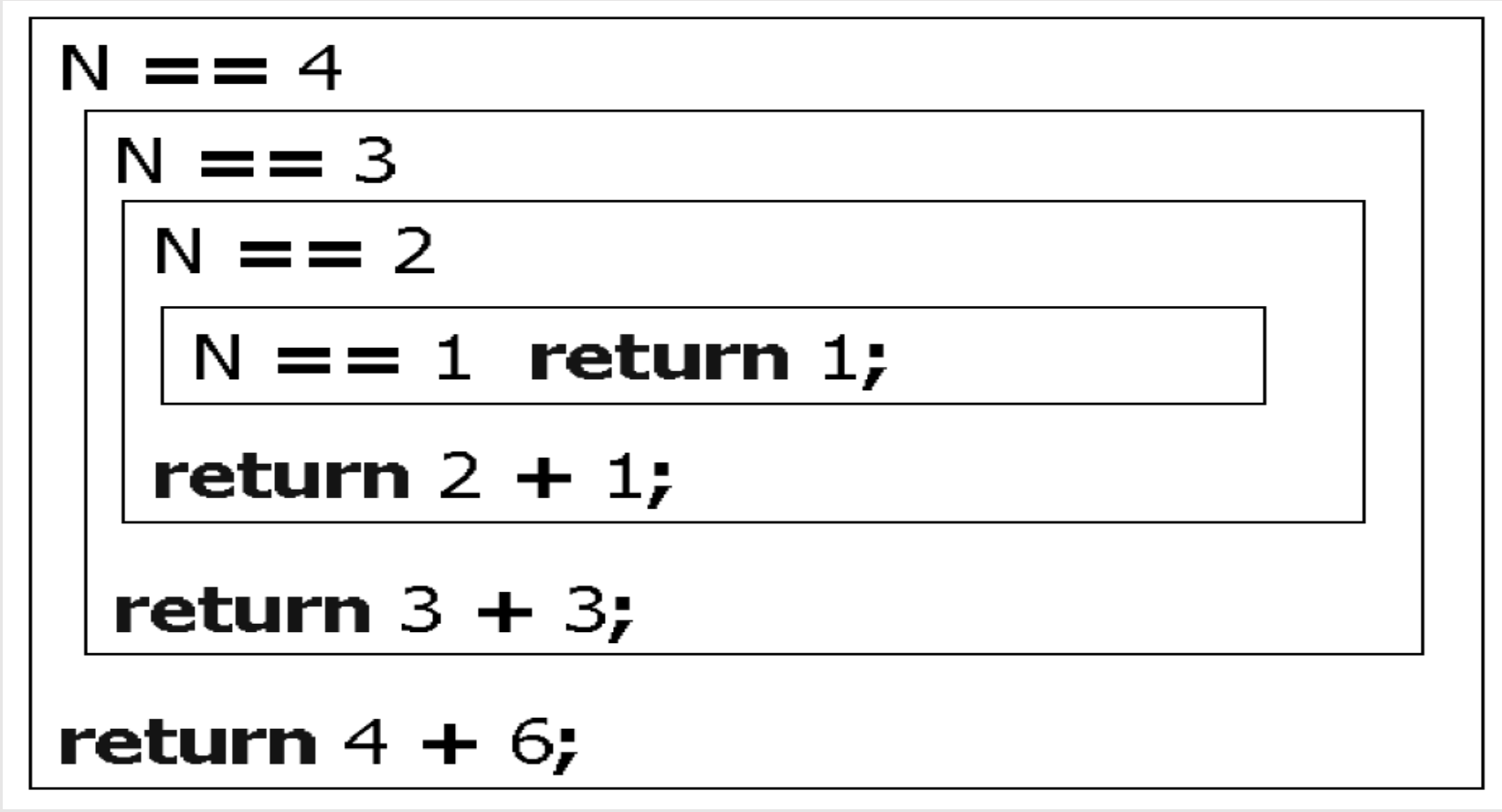
Exemplo: Somatório

UNINASSAU
Campus Graças

```
funcao somatorio (n : inteiro): inteiro
inicio
    se n=1 entao
      retorne 1
    senao
      retorne n + somatorio(n-1)
    fimse
fimfuncao
```

Exemplo: Somatório

Para N == 4, temos:



Algoritmos Recursivos x Iterativos

UNINASSAU
Campus Graças

- Todo algoritmo recursivo possui um algoritmo iterativo equivalente, mudando apenas a sua complexidade de construção.
- **Vantagens**
  - Simplifica a solução de alguns problemas
  - Algoritmos recursivos são mais **compactos** para alguns tipos de algoritmo, mais **legíveis** e mais **fáceis** de ser **compreendidos** e **implementados**.

Algoritmos Recursivos x Iterativos

UNINASSAU
Campus Graças

- **Desvantagens**
  - Por usarem intensivamente a pilha de execução, os algoritmos recursivos tendem a ser mais lentos e a consumir mais memória que os iterativos, porém pode valer a pena sacrificar a eficiência em benefício da clareza.

  - Erros de implementação podem levar a estouro de pilha. Isto é, caso não seja indicada uma condição de parada, ou se esta condição nunca for satisfeita, entre outros.

Algoritmos Recursivos x Iterativos

UNINASSAU
Campus Graças

Quando não usar Recursão:

- **Repetição do processamento:**

  Em geral, cada chamada recursiva é independente uma da outra. Caso ocorram os mesmos cálculos para duas chamadas recursivas independentes, esses cálculos serão repetidos para cada chamada.

- **Gasto de memória excessivo:**

  Cada chamada recursiva aloca memória para as variáveis locais e para os parâmetros, sendo que na forma interativa isso ocorre apenas uma vez.

Recursão – Exercício 1

UNINASSAU
Campus Graças

- Faça um algoritmo recursivo para elevar um número a uma potência inteira não negativa. Faça também uma versão interativa dessa função.

$$x^n = \begin{cases} 1 & \text{se } n = 0 \\ x \cdot x^{n-1} & \text{se } n > 0 \end{cases}$$

Recursão – Exercício 2

UNINASSAU
Campus Graças

- Escreva uma função recursiva MDC(n,m) que retorne o maior divisor comum de dois números inteiros n e m, de acordo com as seguintes definições:

$$MDC(n,m) = \begin{cases} m & \text{se } m<=n \text{ e } n \bmod m = 0 \\ MDC(m,n) & \text{se } n<m \\ MDC(m,\ n \bmod m) & \text{caso contrário} \end{cases}$$

UNINASSAU
Campus Graças

Algoritmos de ordenação

Ordenação de dados

UNINASSAU
Campus Graças

- A medida que trabalhamos com uma base de dados, surge a necessidade de exibir os dados ordenados sob diferentes critérios:
  - Ex: Podemos ordenar os clientes por ordem alfabética, os alunos pela nota, moradores pelo CEP, imóveis pelo preço, etc.

- A ordenação também pode ser usada como passo preliminar para a realização de uma série de buscas.

Ordenação de dados

UNINASSAU
Campus Graças

- Abordaremos aqui os seguintes algoritmos:
  - Bubble Sort;
  - Selection Sort;

Análise de algoritmos

UNINASSAU
Campus Graças

- Ao analisar diferentes algoritmos que resolvem um mesmo problema, surge a questão de como compará-los, de forma a saber quanto um algoritmo é “melhor” ou “pior” do que outro;
- Solução: medir o tempo de execução de cada algoritmo para uma mesma quantidade de trabalho e comparar os resultados.
- Essa solução entretanto tem alguns problemas:
  - Não leva em consideração o que ocorre quando o algoritmo é executado em plataformas com diferente poder computacional;
  - Não leva em consideração o que ocorre quando a quantidade de trabalho a ser feito varia.

Análise de algoritmos

UNINASSAU
Campus Graças

- Seja **A** um algoritmo para um problema **P**. A quantidade de tempo que **A** consome para processar uma quantidade de trabalho **W** depende da máquina usada para executar esse algoritmo **A**.
- Mas o efeito da máquina se resume a uma constante multiplicativa: se **A** consome tempo **t** numa determinada máquina, consumirá tempo **2*t** numa máquina duas vezes mais lenta e **t/2** numa máquina duas vezes mais rápida.
- Para eliminar o efeito da máquina, basta discutir o consumo de tempo de A sem considerar o efeito de constantes multiplicativas. A notação assintótica é ideal para fazer isso.

Exemplo

UNINASSAU
Campus Graças

- Informe usando a notação assintótica quanto tempo devem levar as seguintes operações:

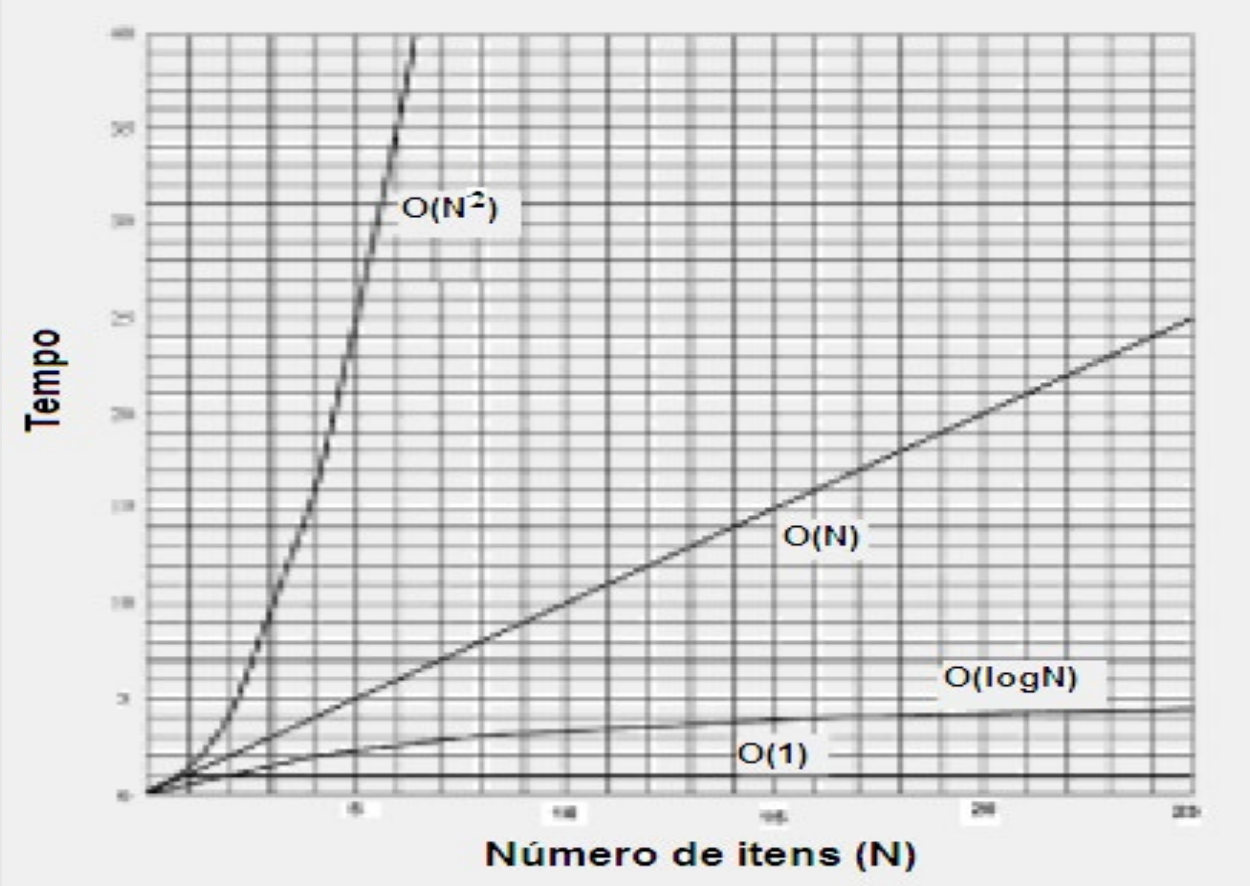
1) Uma busca de uma informação em um vetor:
   - Uma busca linear possui tempo de execução O(N).

2) Inserção de um elemento em um vetor não ordenado:
   - Inserir um elemento leva O(1) (tempo constante).

3) Inserção de um elemento em um vetor ordenado:
   - Inserir um elemento em um vetor ordenado leva O(N).

Notação Assintótica O(x)

UNINASSAU
Campus Graças



Notação Assintótica O(x)

UNINASSAU
Campus Graças

- De acordo com o gráfico, podemos classificar os valores utilizando a notação O para análise dos algoritmos da seguinte forma:

  - O(1) ➡ excelente;
  - O(log N) ➡ bom;
  - O(N) ➡ ruim;
  - O(N^2) ➡ péssimo.

Ordenação de dados

UNINASSAU
Campus Graças

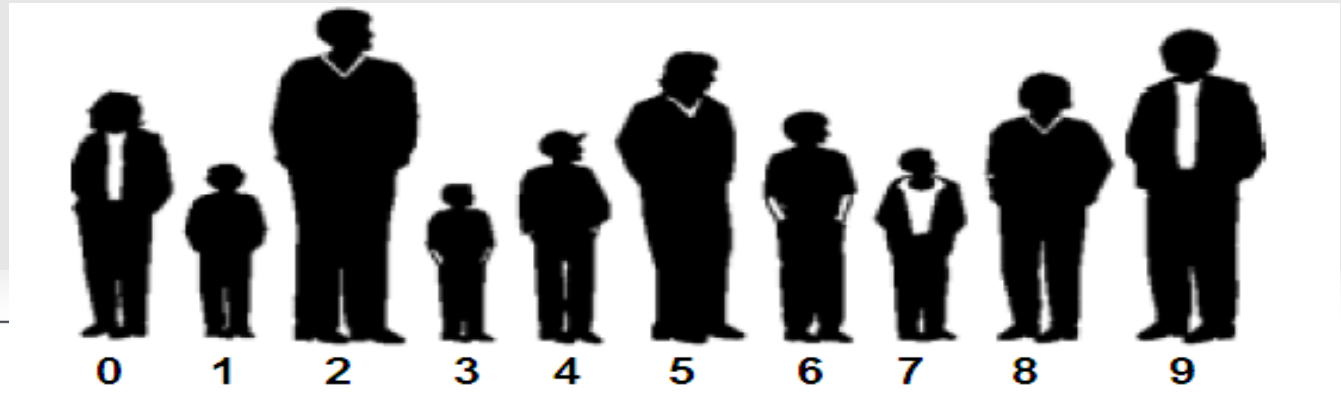
- De forma geral, os algoritmos de ordenação ***Bubble Sort*** e ***Selection Sort*** envolvem dois passos, executados repetidamente até que os dados estejam ordenados:
  - 1. Comparar dois itens.
  - 2. Trocar de posição dois itens ou copiar um item.
- A forma que cada algoritmo realiza a ordenação entretanto irá variar.

Funcionamento do Bubble Sort

UNINASSAU
Campus Graças

- Embora seja lento, é o mais simples dos algoritmos de ordenação;

- Ex: Vamos ordenar por tamanho (menor para o maior) os garotos que irão participar de uma competição esportiva. Vamos assumir que existam N atletas, cada um ocupando uma posição que vai de **0** na esquerda até **N – 1** na direita.



Funcionamento do Bubble Sort

UNINASSAU
Campus Graças

- Começaremos do início da fila (a esquerda) e comparamos os dois garotos nas posições 0 e 1. Se o garoto na posição zero for maior, nós trocamos eles de posição, do contrário, deixamos do jeito que está . Em seguida, fazemos o mesmo com os garotos na posição 1 e 2.

- Em resumo, aqui estão as regras:
  - 1. Comparar os dois atletas.
  - 2. Se o da esquerda for maior, troque os dois de posição.
  - 3. Mova uma posição para a direita e volte ao passo 1.

Funcionamento do Bubble Sort

UNINASSAU
Campus Graças

- Ao chegar ao final da linha (a direita) sabemos que a última posição contém o garoto mais alto. Em seguida, repetimos o processo, só que agora sem considerar a última posição a direita, pois essa já está ordenada.

- Repetiremos o processo, sempre diminuindo em 1 unidade os elementos a serem comparados na próxima passagem, até que todos estejam ordenados.

Exercício

UNINASSAU
Campus Graças

- Faça uma função Ordenacao_Bubble que recebe um vetor de números inteiros e ordena o mesmo de acordo com o método **Bubble Sort**.

Solução

UNINASSAU
Campus Graças

```
public static void ordenacao_bubbleSort(int[ ] vetor)   {
        int out, in;
        int nElems = vetor.length;
        int temp = 0;
        for(out=nElems-1; out>0; out--) // loop externo (trás para frente)
            for(in=0; in<out; in++) // loop interno (em frente)
              if( vetor[in] > vetor[in+1] ) { // fora de ordem?
                temp = vetor[in];
                vetor[in] = vetor[in+1];
                vetor[in+1] = temp;
              }
} // fim bubbleSort()
```

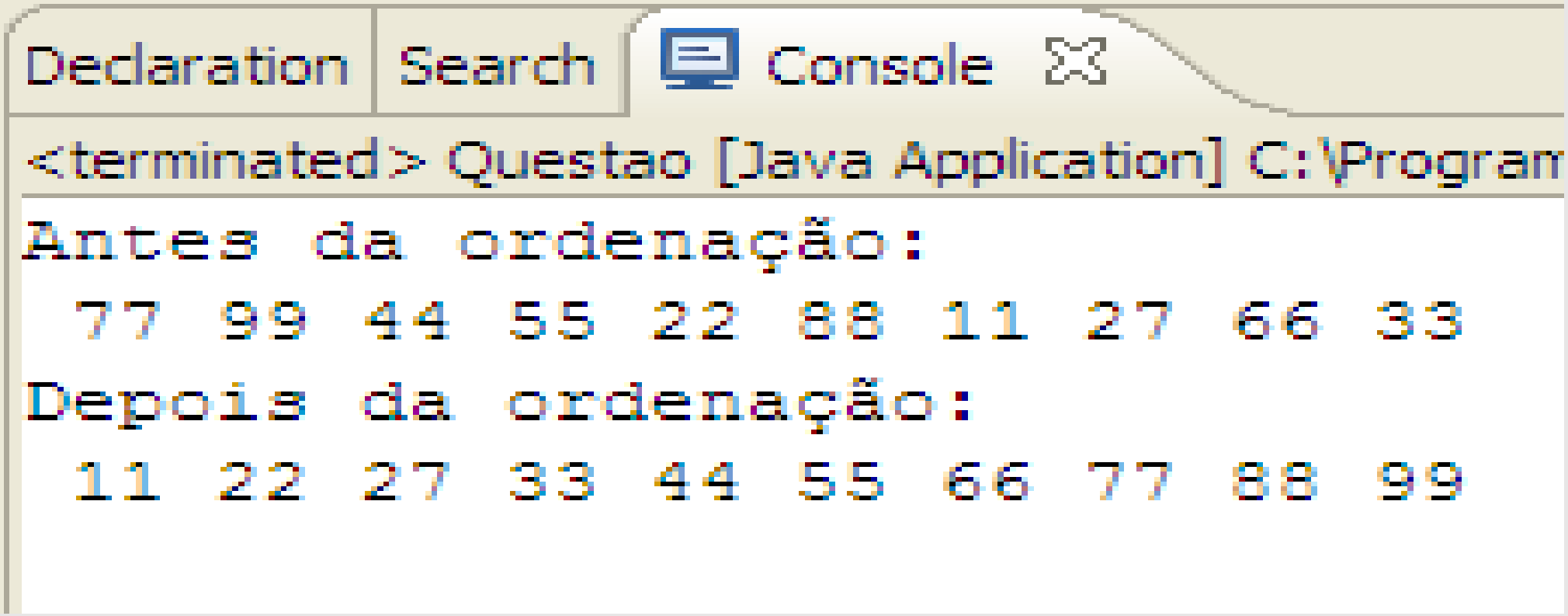
Solução

```
        public static void main(String argv[])   {

        int[ ] vetor1 = new int[ ] {77,99,44,55,22,88,11,27,66,33};
        System.out.println("Antes da ordenação: ");
        for (int i=0;i<vetor1.length;i++)
                System.out.print(" " + vetor1[i]);
        System.out.println();
        ordenacao_bubbleSort(vetor1);
        System.out.println("Depois da ordenação: ");
        for (int i=0;i<vetor1.length;i++)
                System.out.print(" " + vetor1[i]);
}
```

Teste do código Bubble Sort

UNINASSAU
Campus Graças



Selection Sort

UNINASSAU
Campus Graças

- O algoritmo Selection sort é um avanço em relação ao Bubble sort por reduzir o número de trocas de $O(N^2)$ para $O(N)$. Infelizmente, o número de comparações continua $O(N^2)$.

- Apesar disso, o selection sort pode representar uma grande melhoria no caso de registros grandes que precisam ser movidos fisicamente na memória, quando o tempo de troca pode ser bem maior que o tempo de comparação.

- Em Java isso não acontece, já que são as referências que são movidas de lugar, e não os objetos.

Descrição do Selection Sort

UNINASSAU
Campus Graças

- O Selection Sort faz uma passagem por todos os elementos e seleciona (daí o nome) o menor de todos. Esse menor elemento é então trocado com o primeiro elemento a esquerda (posição 0). Agora o primeiro elemento está ordenado, e não precisa ser movido de novo.

- Repetimos o processo, mas agora começando da posição 1, e após localizar o menor elemento, trocamos ele com o da posição 1. Esse processo continua até que todos os elementos estejam ordenados.

- Perceba que no Selection Sort os elementos ordenados se acumulam do lado esquerdo (índices menores) enquanto no Bubble sort eles se acumulam no lado direito.

Exercício

UNINASSAU
Campus Graças

- Faça uma função Ordenacao_Selection que recebe um vetor de números inteiros e ordena o mesmo de acordo com o método **Selection Sort**.

Solução Selection Sort

UNINASSAU
Campus Graças

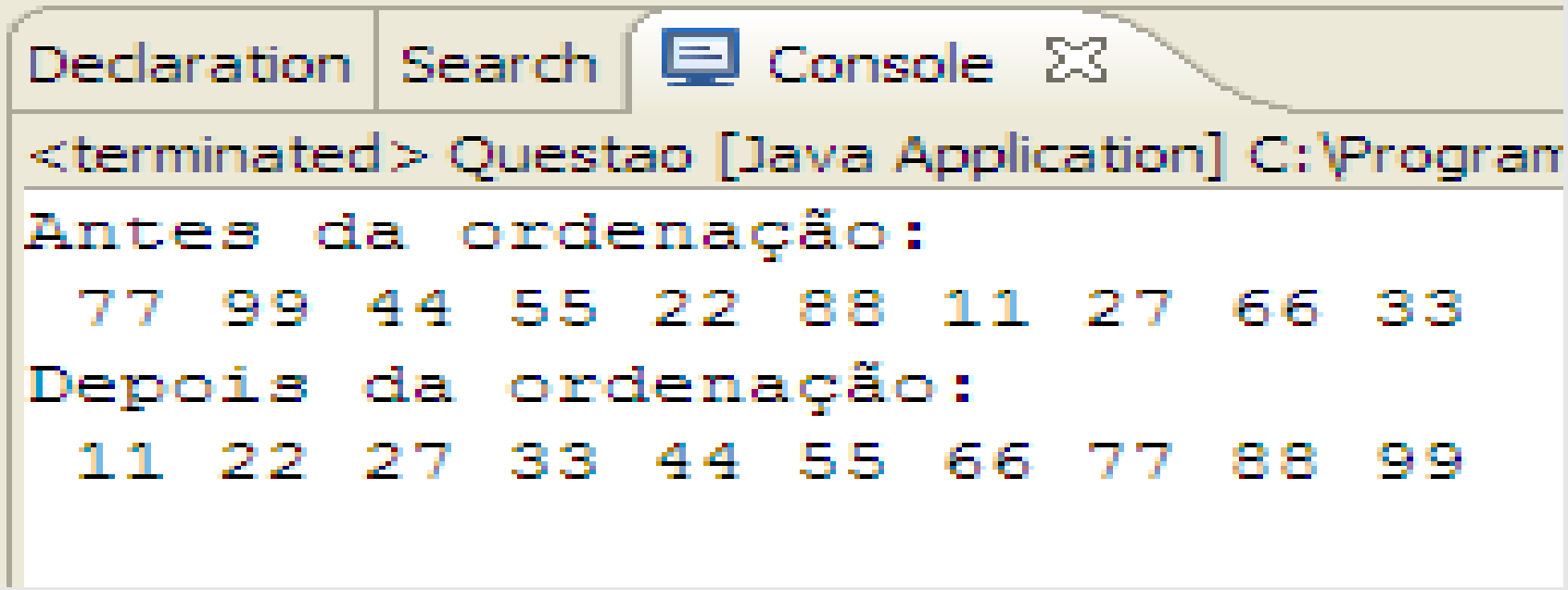
```
public static void ordenacao_Selection_Sort(int[ ] vetor)   {
 int out, in, min, temp;
 int nElems = vetor.length;
 for ( out=0; out<nElems-1; out++) { // loop externo
            min = out; // minimum
            for ( in=out+1; in<nElems; in++) // loop interno
                        if ( vetor[in] < vetor[min] ) // se  min é maior,
                                    min = in; // temos um novo mínimo
            // Coloca o novo mínimo no seu lugar correto no vetor
            temp = vetor[out];
            vetor[out] = vetor[min];
            vetor[min] = temp;
 } // fim do laço externo
} // fim do selectionSort()
```

Teste do Selection Sort


```
public static void main(String argv[])   {
      int[ ] vetor1 = new int[ ] {77,99,44,55,22,88,11,27,66,33};
      System.out.println("Antes da ordenação: ");
      for (int i=0;i<vetor1.length;i++)
            System.out.print(" " + vetor1[i]);
      System.out.println();
      ordenacao_Selection_Sort(vetor1);
      System.out.println("Depois da ordenação: ");
      for (int i=0;i<vetor1.length;i++)
            System.out.print(" " + vetor1[i]);
}
```

Teste do Selection Sort

UNINASSAU
Campus Graças



Eficiência do Selection Sort

UNINASSAU
Campus Graças

- Para valores grandes de N, o tempo de comparação irá dominar, então podemos dizer que o Selection Sort tem tempo de execução $O(N^2)$, da mesma forma que o Bubble sort.

- Apesar disso, é mais rápido devido ao menor número de trocas. Para valores pequenos de N, a diferença de performance será maior, especialmente se o tempo de troca for maior que o tempo de comparação.

Comparando os algoritmos

UNINASSAU
Campus Graças

- Exercício:
  - Vamos verificar o tempo de execução dos 2 métodos de ordenação. Para isso, execute os 2 algoritmos passando como parâmetro um mesmo vetor de 100000 elementos inteiros criados aleatoriamente (lembre-se de copiar os dados para dois outros vetores antes de rodar os algoritmos).
  - Entre cada execução de cada algoritmo meça o tempo decorrido para verificar quanto tempo o algoritmo precisou para ordenar os dados.
  - Varie o número de elementos para verificar como o tempo de execução varia com a quantidade de elementos.

Comparando os algoritmos

UNINASSAU
Campus Graças

```
public static void main(String argv[])   {
        String fx;
        NumberFormat nf = NumberFormat.getNumberInstance();
        nf.setMaximumFractionDigits(9);
        nf.setMinimumIntegerDigits(1);
        double r=0;

        int[] vetor5 = criaVetor(100000);
         int[] vetor6 = new int[100000];

        // copia os elementos do vetor 5 para o outro vetor.
        for (int i=0;i<vetor5.length;i++)  {
                vetor6[i] = vetor5[i];
        }
```

Comparando os algoritmos

UNINASSAU
Campus Graças

```
// medicao algoritmo Bubble Sort.

    long t10 = System.nanoTime();
    ordenacao_bubbleSort(vetor5);
    long t11 = System.nanoTime();
    r = ((double)(t11 - t10))/1000000000;
    fx = nf.format(r);
    System.out.println("Tempo decorrido Bubble Sort: 100000 elementos " +        fx + "
    segundos");
```

Comparando os algoritmos

UNINASSAU
Campus Graças

```
        // medicao algoritmo Selection Sort.

        long t12 = System.nanoTime();
        ordenacao_Selection_Sort(vetor6);
        long t13 = System.nanoTime();
        r = ((double)(t13 - t12))/1000000000;
        fx = nf.format(r);
        System.out.println("Tempo decorrido Selection Sort: 100000
elementos "      + fx + " segundos");
 }
```

Comparando os algoritmos

UNINASSAU
Campus Graças

```
public static int[] criaVetor(int tamanho)  {
        int[] v = new int[tamanho];
        for (int i = 0;i< tamanho; i++)  {
                v[i] = (int) ( 100*Math.random() + 1);
        }
        return v;
}
```

Comparando os algoritmos

UNINASSAU
Campus Graças

- O uso do algoritmo Bubble Sort só é viável se a quantidade de dados for pequena;
- O algoritmo selection sort minimiza o número de trocas, mas o número de comparações continua alto. Ele pode ser útil se a quantidade de dado's for pequena e o tempo de troca é grande em relação ao tempo de comparação;
- Visite o site:
  - http://www.sorting-algorithms.com/

Obrigado

E- mail:adilson.silva@sereducacional.com

@prof.Adilson.silva