# PROGRAMAÇÃO E ALGORITMOS (LEII)

Universidade da Beira Interior, Departamento de Informática
Hugo Pedro Proença, 2016/2017

# Resumo

- Complexidade Computacional
  - Notação “Big-O”
  - Exemplos
  - Exercícios

# Complexidade Computacional

- Exercício: Considre a seguinte estrutura de dados:
  - typedef struct{

    int BI;

    char nome[80];

    float salario;

    }Pessoa;
- Suponha que um vector guarda informação sobre um conjunto de pessoas, ordenadas pelo seu nº de BI.
- Implemente uma função que devolva o salário de uma determinada pessoa (especificada pelo nome).

# Complexidade Computacional

- Protótipo:

  - float procuraNome(Pessoa *v, int total, char *nome);

# Complexidade Computacional

```
float procuraNome(Pessoa *v, int total, char *nome){
    int i;                                          //1
    for (i=0;i<total;i++)                           //2
       if (strcmp(v[i].nome,nome)==0)               //3
                return(v[i].salario);               //4
    return(-1);                                     //5
}
```

# Complexidade Computacional

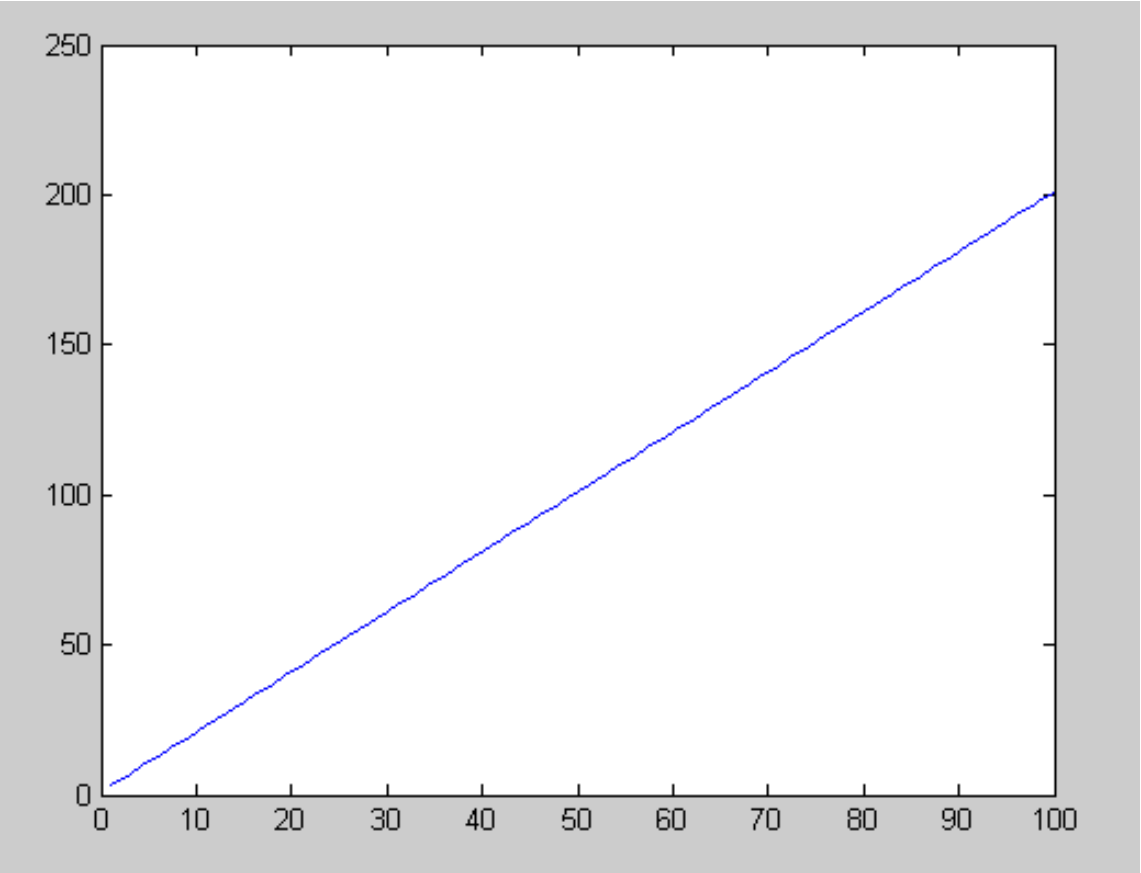
- Qual a complexidade computacional da função implementada?
  - Pior caso: O maior número de passos executados ocorrerá quando o nome da pessoa a procurar não aparece no vector.
    - //1, apenas declaração estática. Não se conta como "passo".
    - //2, será executada "total" vezes
    - //3, será executada "total" vezes
    - //4, não é executada
    - //5, será executada 1 vez

# Pior caso, Limite assintótico

- Para um vector com tamanho 10, no pior caso a função vai executar 2*10 + 1 passos=21.
- Para um vector com tamanho 1000, no pior caso vão ser executados 2*1000+1=2001 passos.
- No caso geral vão ser executados 2*n+1 passos, sendo "n" o tamanho do vector.
- Sendo f(n)=2*n+1, para calcular a complexidade computacional da função, é necessário encontrar uma função g(n) que, multiplicada por uma constante positiva, funcione como limite assintótico para f(n):
  - **f(n)=O(c g(n))**

# Pior caso, Limite assintótico

f(n)

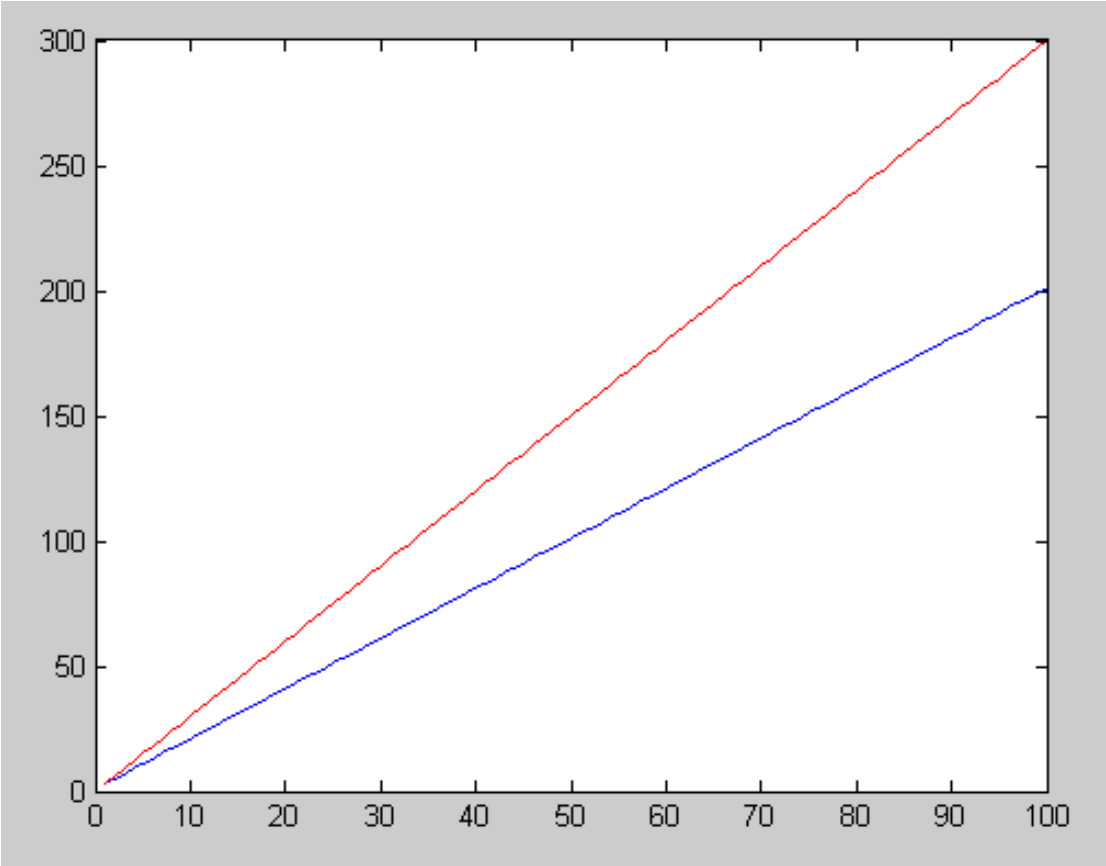



n

# Notação “Big-O”

- A função f(n) tem claramente um comportamento linear. Assin, sendo sendo g(n)=n, para c=3 (ou superior), temos:

f(n)=2n+1
g(n)=3n



# Pior caso, Limite assintótico

- Como f(n) <= 3 n, concluímos que f(n) é uma função com complexidade computacional linear, isto é:
  - F(n)=O(n)
- Exercício: Repita a procura feita anteriormente, mas utilizando como elemento de procura o nº de bilhete de identidade, critério de ordenação no vector:

  - float procuraBI(Pessoa *v, int total, int BI);

# Pesquisa Binária:

- Uma solução que apenas substitua a comparação entre nomes pela comparação entre BIs sería pouco satisfatória.
  - Suponhamos que andamos à procura do BI=12 000 000 e o 1° elemento do vector é o 4 321 000. Será provável que o elemento a procurar esteja na posição seguinte?
  - Se formos procurar na lista telefónia por alguém chamado "Rebelo", fará sentido irmos às primeiras páginas da lista?
    - Abreu, Alexandre,...

# Pesquisa Binária:

- O algoritmo de pequisa binária é bastante eficiente e simples de implementar. Consiste em ir dividindo ao meio o espaço em que é plausível procurar.
  - Temos um espaço de tamanho "n" a procurar.
  - Vamos ao meio do espaço (n/2) e verificamos se o elemento que procuramos está para trás ou para a frente desse elemento.
    - Se estiver para atrás continuamos a procurar entre 1→n/2.
    - Se estiver para a frente procuramos entre n/2→n.
  - Repete-se este procedimento até encontrar o elemento.

# Pesquisa Binária:

```
float procuraBI(Pessoa *v, int total, int BI){
int inicio=1, fim=total, actual;          //1
while (inicio<total){                     //2
    actual=(inicio+fim)/2;                //3
    if (v[actual].BI==BI)                 //4
        return(v[actual].salario);        //5
    if (v[actual].BI>BI)                  //6
        fim=actual;                       //7
    else
        inicio=actual;                    //8
}
return(-1);                               //9
```

# Pesquisa Binária:

- Exercício: Determine a complexidade computacional da função de pesquisa binária.
  - Identifique o pior caso.
  - Simule o número de passos, para n=10, 100, 1000, ...
  - Encontre uma função f(n) que aproxima o n° de passos, no caso geral.
  - Encontre uma função g(n) tal que:
    - f(n) <= c g(n)
  - Conclua que f(n)=O(g(n)).

# Pesquisa Binária:

- Complexidade logarítmica: