# PROGRAMAÇÃO E ALGORITMOS (LEII)

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

# Resumo

- Ordenação e Pesquisa
  - Pesquisa Linear
  - Pesquisa Binária
  - Inserção Ordenada
  - InsertSort
  - ShellSort

# Ordenação e Pesquisa

- Os sistemas computacionais são usados para guardar grandes quantidades de dados, a partir de onde registos indivisuais devem ser encontrados, de acordo com algum critério.
  - A única razão para guardar algum item de informação é a sua utilidade futura.
  - Nalgum momento vai-se especificar um critério de pesquisa, a partir do qual deverá ser devolvido um sub-conjunto dos items guardados.

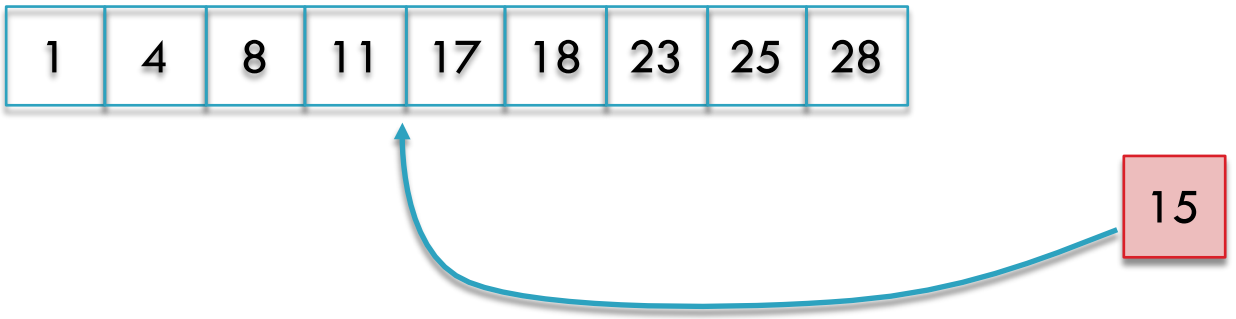
# Ordenação e Pesquisa

- Os algoritmos mais simples de pesquisa apenas são satisfatórios para quantidades de dados reduzidas.
  - "Toy problems" académicos podem criar ilusão de funcionalidade.
- Pesquisa Linear:

```
Pessoa* procura(Pessoa *v, int tot, int BI){
        int i;
        for (i=0;i<tot;i++)
                if (v[i].BI==BI)
                        return(&v[i]);
        return(NULL);
}
```

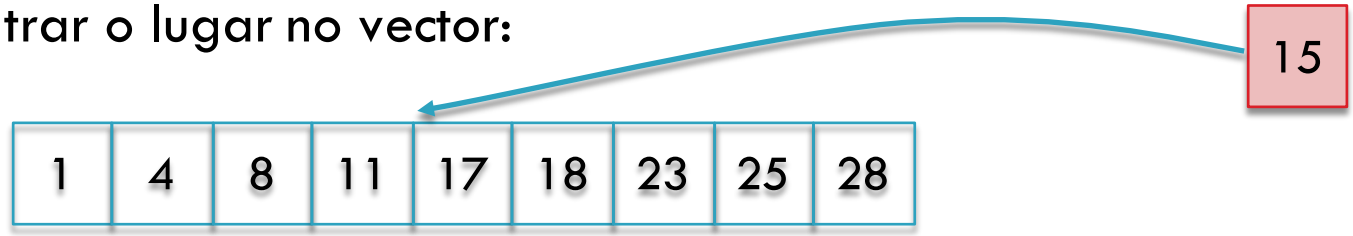
# Ordenação e Pesquisa

- Em dados ordenados, a pesquisa binária representa uma vantagem decisiva no desempenho do sistema.
- Os vectores são estruturas de dados particularmente difíceis de manter ordenadas.



# Inserção Ordenada

- É um dos algoritmos que permite manter um conjunto de dados sempre ordenado.
- Ao inserir um novo elemento, ele é guardado na posição do vector que mantém a estrutura ordenada:
  - Encontrar o lugar no vector:




  - Arranjar espaço e fazer shift de todos os elementos à direita:


  - Copiar o elemento a inserir para a respectiva posição:

# Inserção Ordenada

- Implemente uma função em linguagem C que insira um novo elemento num vector de forma ordenada:

  - Pessoa* insertSort(Pessoa *v, int* tot, Pessoa nv);

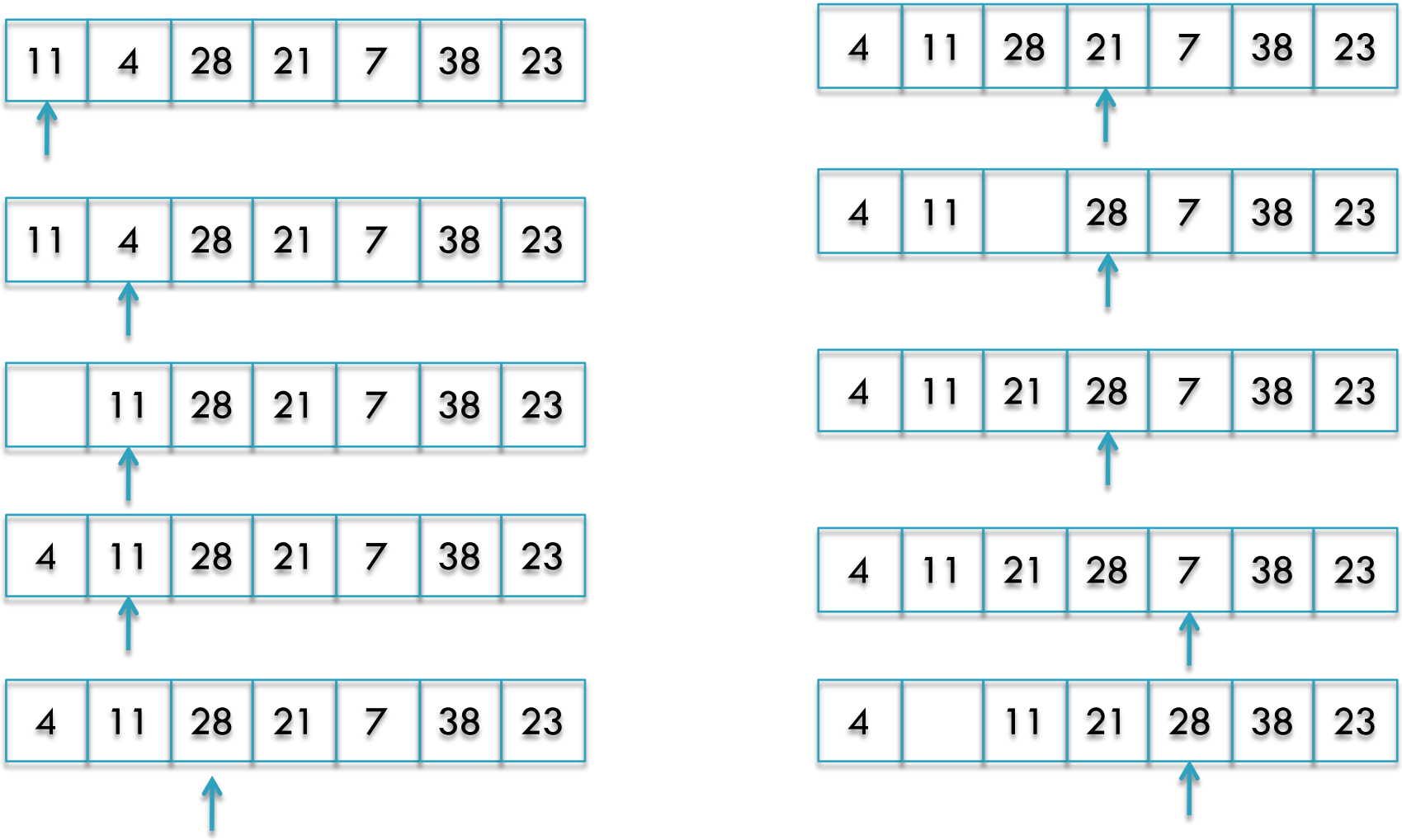
    //v=vector de pessoas; tot=total de elementos no vector

    //nv=elemento a inserir

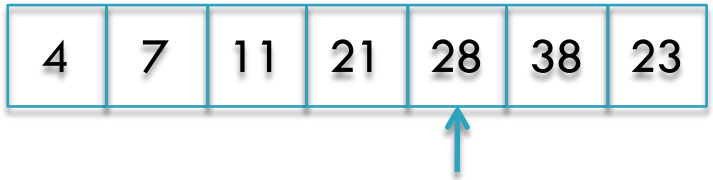
# InsertSort

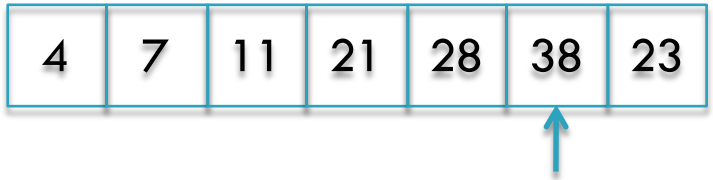
- Suponhamos agora que um conjunto de dados (vector) não está ordenado.
- O algoritmo "InsertSort" é um algoritmo "in-place" que permite a sua ordenação.
  - Os algoritmos in-place não necessitam de recursos adicionais significativos para executar.
  - Pseudo-código:
    - Para a posição i (0<n)
      - Chave = elemento da posição i
      - Para todas as posições à esquerda de "i" (decrescente)
      - Todos os elementos à esquerda de "i" maiores que Chave fazem shift à direita de 1 posição.
      - Coloca "chave" na posição livre.

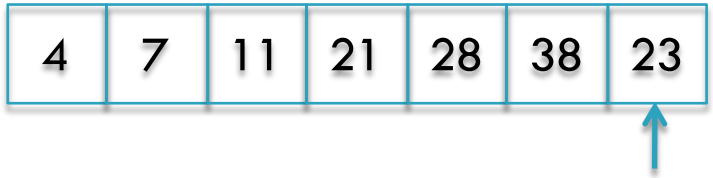
# InsertSort

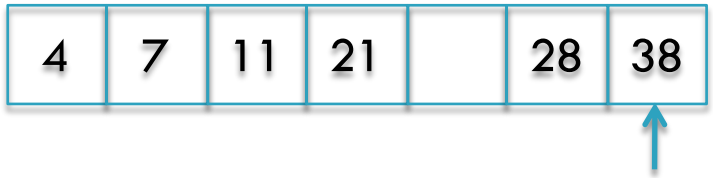

11 4 28 21 7 38 23
11 4 28 21 7 38 23
11 28 21 7 38 23
4 11 28 21 7 38 23
4 11 28 21 7 38 23
4 11 28 21 7 38 23
4 11 28 21 7 38 23
4 11 21 28 7 38 23
4 11 21 28 7 38 23
4 11 21 28 38 23

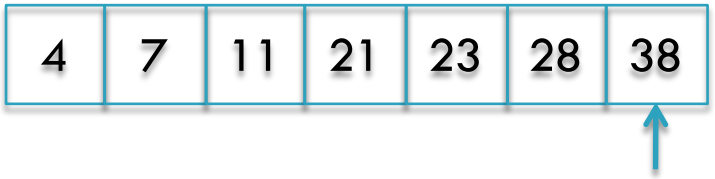
# InsertSort


4 7 11 21 28 38 23


4 7 11 21 28 38 23


4 7 11 21 28 38 23


4 7 11 21 28 38


4 7 11 21 23 28 38

# InsertSort

- Implemente uma função em C que ortdene um vector segundo o algoritmo"InsertSort"

- Pessoa* insertSort(Pessoa *v, int tot);

  //V=vector de pessoas, tot=total de elementos no vector

- Este algoritmo tem complexidade computacional $O(n^2)$, o que para grandes quantidades de dados se pode revelar problemático.
  - Melhor caso?
  - Pior caso?

# Shell Sort

- Este algoritmo é uma generalização do InsertSort
  - O InsertSort é eficiente quando o vector está "quase ordenado".
  - O InsertSort é ineficiente tipicamente devido a mover cada valor uma posição de cada vez.
- O algoritmo Shell sort optimiza o insertSort ao comparar elementos separados por intervalos (de tamanho variável monótono decrescente).
- Itso permite que um elemento dê maiores saltos no vector, em direcção à sua posição de ordenação.
- Quando o intervalo tem tamanho "1", é na realidade equiovalente ao "insertSort", mas nessa altura já o vector estará quase ordenado.

# Shell Sort

- Considere-se o vector [13 14 94 33 82 25 59 97 65 23 45 27 73 25 39 10].
- Começar (por exemplo) com um intervalo de tamanho 5, equivale a visualizar os dados numa forma matricial:
  - 13 14 94 33 82
  - 25 59 97 65 23
  - 45 27 73 25 39
  - 10
- Se ordenarmos os elementos por coluna, o valor 10 aparece na 1ª posição do vector
  - 10 14 73 25 23
  - 13 27 94 33 39
  - 25 59 97 65 82
  - 45
- Visualizando o vector em 1D, temos agora: [10 14 73 25 23 13 27 94 33 39 25 59 97 65 82 45]
- O processo repete-se, com um intervalo inferior, até que intervalo =1

# Shell Sort

- Implemente uma função em C que ortdene um vector segundo o algoritmo"Shell"
- Pessoa* shellSort(Pessoa *v, int tot);

  //V=vector de pessoas, tot=total de elementos no vector
- Algoritmo:
  - *incremento* ← round(*n*/2)
  - while *inc* > 0 do:
    - for *i* = *incremento* .. *n* − 1 do:
      - *temp* ← a[*i*]
      - *j* ← *i*
      - while *j* ≥ *incremento* and a[*j* − *incremento*] > *temp* do:
        - a[*j*] ← a[*j* − *incremento*]
        - *j* ← *j* − *incremento*
        - a[*j*] ← *temp*
    - *incremento* ← round(*incremento* / 2.2)