CEFET – RJ / *Campus* Maria da Graça
Centro Federal de Educação Tecnológica
Celso Suckow da Fonseca – Rio de Janeiro

Prof. Cristiano Fuschilo

cristiano.fuschilo@cefet-rj.br

Estruturas de Dados

Bacharelado em Sistemas de Informação

COORDENAÇÃO DE Automação Industrial
Ensino de Qualidade

Sort

Ordenação dos Elementos de um Vetor

Algoritmo Bubble Sort

- Ordenar é o ato de se colocar os elementos de uma sequência, em uma dada relação de ordem entre si, de acordo com um critério pré-estabelecido.
- Os elementos do vetor devem ser trocados entre si para que fiquem na ordem desejada.Vamos iniciar com um exemplo simples que apresenta a troca do conteúdo de duas variáveis inteira (sem pensar em vetor).

Exemplo

```c
#include

void trocar (int *px, int *py) {

    int aux = *px;
    *px = *py;
    *py = aux;

}

void main(){

    int x, y;

    printf("\nValor para x? ");
    scanf("%d",&x);

    printf("\nValor para y? ");
    scanf("%d",&y);

    // mostrando os conteudos de x e y
    printf("\nX = %d e Y = %d\n", x, y);

    printf("\nTrocando...\n");
    trocar (&x, &y);

    printf("\nX = %d e Y = %d\n", x, y);

}
```

Método de Ordenação Bolha (Bubble Sort)

- Algoritmo: Percorre várias vezes o vetor de maneira sequencial (passos).
- Em cada passo, compara 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.

Observe

- O elemento da posição 0 (valor 50) é comparado com o elemento da posição 1 (valor 30).

0	1	2	3	4
50	30	40	20	10

- Como o objetivo é ordenar crescentemente, os conteúdos dos elemetos da posições 0 e 1 devem ser trocados.

0	1	2	3	4
30	50	40	20	10

Bubble Sort

- Em seguida serão comparados os conteúdos dos elementos das posições 1 e 2:

0	1	2	3	4
30	50	40	20	10

- Troca:

0	1	2	3	4
30	40	50	20	10

Bubble Sort

- Elementos das posições 2 e 3:

0	1	2	3	4
30	40	50	20	10

- Troca

0	1	2	3	4
30	40	20	50	10

Bubble Sort

- Elementos das posições 3 e 4:

0	1	2	3	4
30	40	20	50	10

- Troca

0	1	2	3	4
30	40	20	10	50

Bubble Sort

- Apesar do vetor não estar ordenado ainda, observe que o maior elemento ficou na última posição:

0	1	2	3	4
30	40	20	10	50

Bubble Sort

- O processo recomeça, porém ocorrerá entre as posições 0 e 3 (o elemento da posição 4 já está ordenado).Elementos das posições 0 e 1.:

0	1	2	3	4
30	40	20	10	50

- Não Troca

0	1	2	3	4
30	40	20	10	50

Bubble Sort

- Elementos das posições 1 e 2.:

0	1	2	3	4
30	40	20	10	50

- Troca

0	1	2	3	4
30	20	40	10	50

Bubble Sort

- Elementos das posições 2 e 3.:

0	1	2	3	4
30	20	40	10	50

- Troca

0	1	2	3	4
30	20	10	40	50

Bubble Sort

- E o processo recomeça, compara as posições o com 1:

0	1	2	3	4
30	20	10	40	50

- Troca

0	1	2	3	4
20	30	10	40	50

Bubble Sort

- Elementos das posições 1 e 2:

0	1	2	3	4
20	30	10	40	50

- Troca

0	1	2	3	4
20	10	30	40	50

Bubble Sort

- Recomeça... Elementos das posições 0 e 1:

0	1	2	3	4
20	10	30	40	50

- Troca

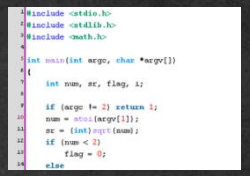
0	1	2	3	4
10	20	30	40	50

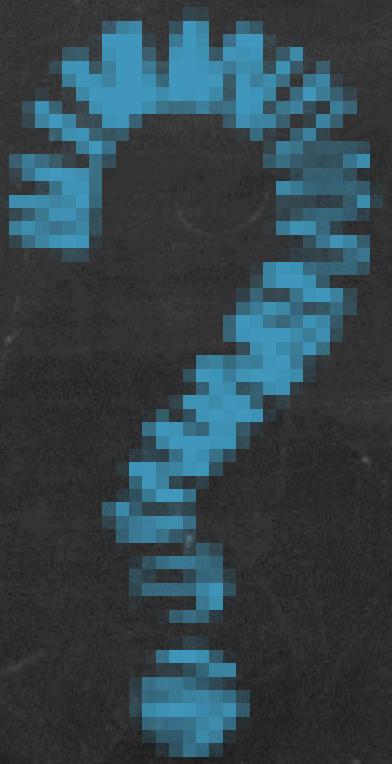
Bubble Sort

- Vetor Ordenado!

0	1	2	3	4
10	20	30	40	50

Bubble Sort - Algoritmo





Algoritmo Insertion Sort

- A ideia por trás da **ordenação por inserção**:
  - Percorrer as posições do vetor, começando com o índice 1.
  - Cada nova posição é como um novo numero, e precisa inserir no lugar correto no subarray ordenado à esquerda daquela posição.

Algoritmo Insertion Sort

- 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 posição seguinte do vetor.
- Mais interessante que o Bubble Sort para popular um vetor.

Observe

- O elemento da posição 1 – novo valor – (valor 30) é comparado com o elemento da posição 0 (valor 50).

0	1	2	3	4
50	30	40	20	10

- Como o objetivo é ordenar crescentemente, os conteúdos dos elemetos da posições 0 e 1 devem ser trocados entre si.

0	1	2	3	4
30	50	40	20	10

Insertion Sort

- Em seguida serão comparados os conteúdos dos elementos das posições 2 e 1:

0	1	2	3	4
30	50	40	20	10

- Troca

0	1	2	3	4
30	40	50	20	10

- Compara os elementos das posições 1 e 0, não troca

0	1	2	3	4
30	40	50	20	10

Insertion Sort

- Elementos das posições 3 e 2:

0	1	2	3	4
30	40	50	20	10

- Troca

0	1	2	3	4
30	40	20	50	10

- Compara os elementos das posições 2 e 1, troca

0	1	2	3	4
30	40	20	50	10

0	1	2	3	4
30	20	40	50	10

- Compara os elementos das posições 1 e 0, troca

0	1	2	3	4
30	20	40	50	10

0	1	2	3	4
20	30	40	50	10

Insertion Sort

- Elementos das posições 4 e 3; troca

0	1	2	3	4
30	40	50	20	10

0	1	2	3	4
30	40	20	10	50

- Compara os elementos das posições 3 e 2; troca

0	1	2	3	4
30	20	40	10	50

0	1	2	3	4
30	20	10	40	50

- Compara os elementos das posições 2 e 1, troca

0	1	2	3	4
20	30	10	40	50

0	1	2	3	4
20	10	30	40	50

- Compara os elementos das posições 1 e 0, troca

0	1	2	3	4
20	10	30	40	50

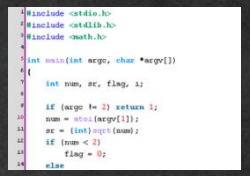
0	1	2	3	4
10	20	30	40	50

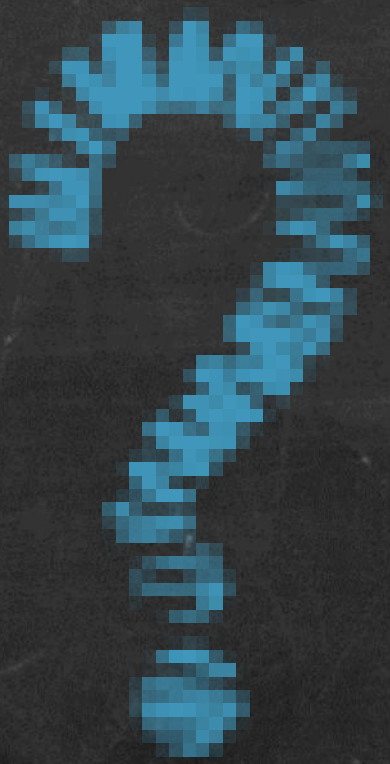
Insertion Sort

- Vetor Ordenado!

0	1	2	3	4
10	20	30	40	50

Insertion Sort - Algoritmo





Merge Sort

- A idéia básica do Merge Sort é criar uma sequência ordenada a partir de duas outras também ordenadas.
- Para isso, o algoritmo Merge Sort divide a sequência original em pares de dados, agrupa estes pares na ordem desejada; depois as agrupa as sequências de pares já ordenados, formando uma nova sequência ordenada de quatro elementos, e assim por diante, até ter toda a sequência.

Merge Sort

- Os três passos úteis dos algoritmos dividir-para-conquistar, que se aplicam ao Merge Sort são:
- Dividir: Dividir os dados em subsequências pequenas; Este passo é realizado recursivamente, iniciando com a divisão do vetor de n elementos em duas metades, cada uma das metades é novamente dividida em duas novas metades e assim por diante, até que não seja mais possível a divisão (ou seja, sobrem n vetores com um elemento cada).
- Conquistar: Classificar as duas metades recursivamente aplicando o merge sort;
- Combinar: Juntar as duas metades em um único conjunto já classificado.

Merge Sort

- Para completar a ordenação do vetor original de n elementos, faz-se o merge ou a fusão dos sub-vetores já ordenados.
- A desvantagem do Merge Sort é que requer o dobro de memória, ou seja, precisa de um vetor com as mesmas dimensões do vetor que está sendo classificado.

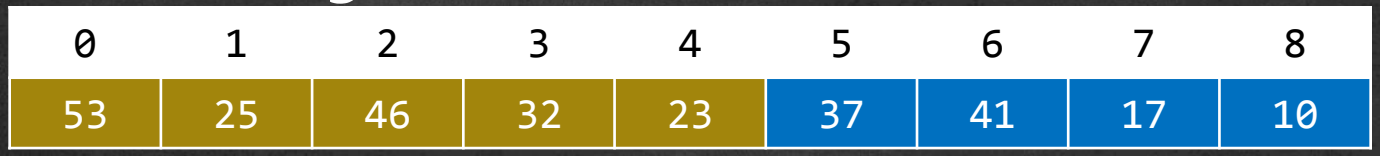
Observe

- Vetor original com elemento desordenados.

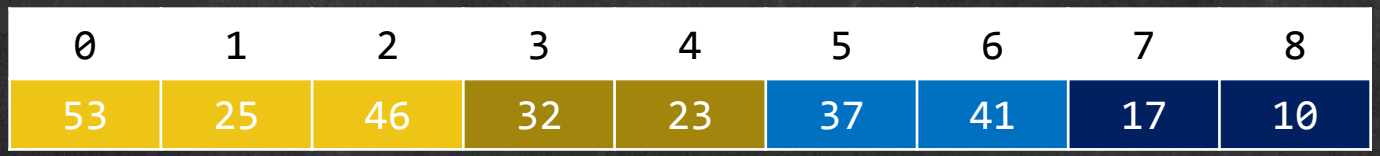
0	1	2	3	4	5	6	7	8
53	25	46	32	23	37	41	17	10

Merge Sort

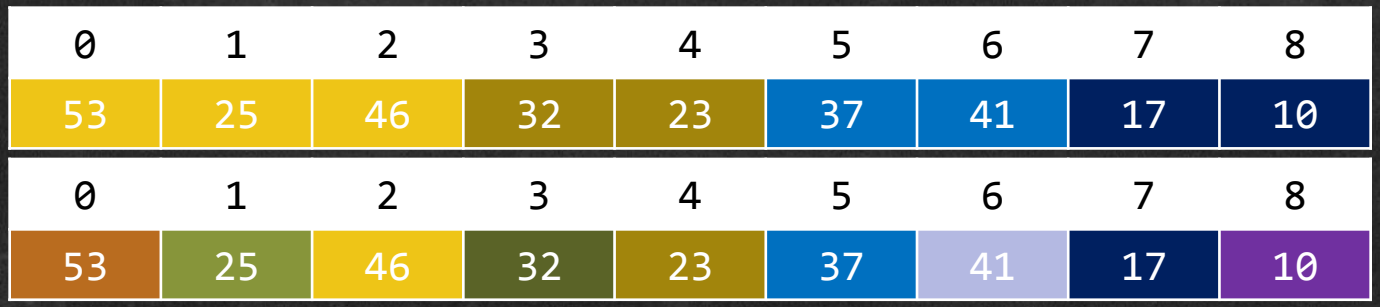
- O vetor original é subdividido em dois vetores.



- Cada um dos subvetores é novamente dividido.



- E assim por diante.



Merge Sort

- Após todo o processo de divisão, ocorre o processo da fusão ordenada dos subvetores.
- O subvetor (53) com o subvetor (25). Ordenando os dois.

0	1	2	3	4	5	6	7	8
25	53	46	32	23	37	41	17	10

Merge Sort

- O subvetor (25, 53) com o subvetor (46). Ordenando os dois.

0	1	2	3	4	5	6	7	8
25	53	46	32	23	37	41	17	10

0	1	2	3	4	5	6	7	8
25	46	53	32	23	37	41	17	10

Merge Sort

- O subvetor (32) com o subvetor (23).

0	1	2	3	4	5	6	7	8
25	53	46	32	23	37	41	17	10

0	1	2	3	4	5	6	7	8
25	46	53	23	32	37	41	17	10

Merge Sort

- O subvetor (25,46,53) com o subvetor (23,32).
- Ordenando os dois.

0	1	2	3	4	5	6	7	8
25	53	46	32	23	37	41	17	10

0	1	2	3	4	5	6	7	8
23	25	32	46	53	37	41	17	10

Merge Sort

- O mesmo processo se repete no subvetor (37, 41, 17, 10).

0	1	2	3	4	5	6	7	8
23	25	32	46	53	37	41	17	10

0	1	2	3	4	5	6	7	8
23	25	32	46	53	37	41	10	17

0	1	2	3	4	5	6	7	8
23	25	32	46	53	10	17	37	41

Merge Sort

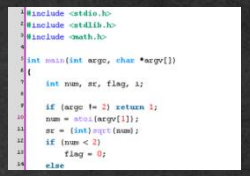
- Os subvetores resultantes (23,25, 32, 36,53) e (10, 17, 37, 41) são fundidos ordenandos durante a fusão.

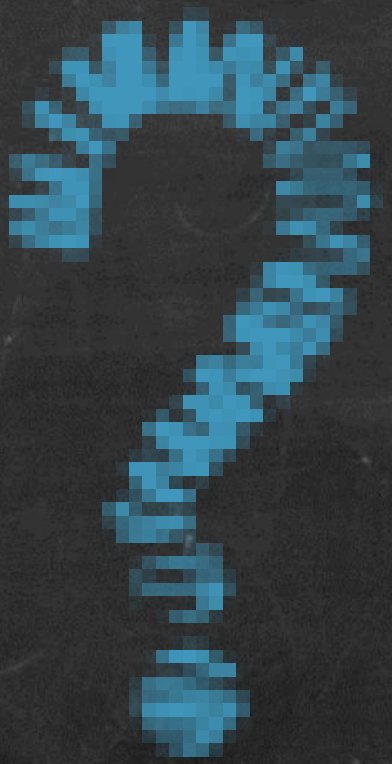
0	1	2	3	4	5	6	7	8
23	25	32	46	53	10	17	37	41

- O processo de ordenação termina!

0	1	2	3	4	5	6	7	8
10	17	23	25	32	37	41	46	53

Merge Sort- Algoritmo





Quick Sort

- Determina-se um elemento pivô.
- O pivô é posicionado dentro do vetor de tal forma que, todos à esquerda do pivô são menores que ele e, todos à direita do pivô são maiores.
- O pivô "divide" o vetor em dois subvetores.
- Recursivamente o quick sort é realizado na primeira metade do vetor e na segunda metade.

Quick Sort

- Algoritmo:
  - Seja x o vetor a ser ordenado e n o número de elementos de x.
  - Seja a um elemento de x escolhido ao acaso (por exemplo, a=x[0]).
  - Suponha que os elementos de x estão divididos de tal forma que a é colocado na posição j e as seguintes condições são verdadeiras:
    - Todos os elementos nas posições de 0 a j-1 são menores que a.
    - Todos os elementos nas posições de j+1 a n-1 são maiores ou iguais a a.
- Então a está na posição correta no vetor.
- Se este processo for repetido para os sub-vetores x[0] a x[j-1] e x[j+1] a x[n-1], o resultado é o vetor ordenado.

Observe

- Vetor original com elemento desordenados.

0	1	2	3	4	5	6	7
25	57	86	48	37	24	92	12

Quick Sort

- O pivô (primeiro elemento do vetor) é escolhido.

0	1	2	3	4	5	6	7
25	57	86	48	37	24	92	12
pivô							

Quick Sort

- A busca por elementos maiores que o pivô se inicia à esquerda do trecho do vetor que está sendo ordenado, e a busca por elementos menores que o pivô se inicia à direita.

0	1	2	3	4	5	6	7
25	57	86	48	37	24	92	12
pivô							dir

Quick Sort

- 25 não é maior que o pivô (25). A busca por um elemento maior que o pivô continua à esquerda.

0	1	2	3	4	5	6	7
25	57	86	48	37	24	92	12
pivô	esq						dir

Quick Sort

- 57 é encontrado. A busca por um elemento menor que o pivô à direita encontra o 12.
- Troca o elemento da esquerda com o da direita.

0	1	2	3	4	5	6	7
25	57	86	48	37	24	92	12
pivô	esq						dir

0	1	2	3	4	5	6	7
25	12	86	48	37	24	92	57
pivô	esq						dir

Quick Sort

- Continua a busca por elementos maiores que o pivô (25) a partir da esquerda e por elementos menores a partir da direita.
- Troca

0	1	2	3	4	5	6	7
25	12	86	48	37	24	92	57
pivô		esq			dir		

0	1	2	3	4	5	6	7
25	12	24	48	37	86	92	57
pivô		esq			dir		

Quick Sort

- O processo continua.

0	1	2	3	4	5	6	7
25	12	24	48	37	86	92	57
pivô		dir	esq				

- Ops! Ocorreu um cruzamento da posição esquerda com a posição direita. Troca o pivô com o elemento da posição direita.

0	1	2	3	4	5	6	7
24	12	25	48	37	86	92	57

Quick Sort

- O subvetor à esquerda do pivô contém os elementos menores que o pivô e o subvetor à direita do pivô contém os elementos maiores que o pivô. O processo se repete para o subvetor da esquerda do vetor e para o subvetor da direita.

0	1	2	3	4	5	6	7
24	12	25	48	37	86	92	57

- O 24 é pivô no subvetor à esquerda.

0	1	2	3	4	5	6	7
24	12	25	48	37	86	92	57
esq	dir						

Quick Sort

- E o processa continua no subvetor à esquerda.

0	1	2	3	4	5	6	7
12	24	25	48	37	86	92	57
esq	dir						

- E a direita, onde o pivô é o 48.

0	1	2	3	4	5	6	7
24	12	25	48	37	86	92	57
			esq				dir

Quick Sort

- Continua...Ops! Cruzamanto da posiçães direita e esquerda. Troca o pivô com o elemento da posição direita.

0	1	2	3	4	5	6	7
24	12	25	48	37	86	92	57
				dir	esq		

0	1	2	3	4	5	6	7
24	12	25	37	48	86	92	57
					esq		dir

Quick Sort

- Continua

0	1	2	3	4	5	6	7
24	12	25	37	48	86	92	57
						esq	dir

0	1	2	3	4	5	6	7
24	12	25	37	48	86	57	92
						esq	dir

0	1	2	3	4	5	6	7
24	12	25	37	48	86	57	92
						dir	esq

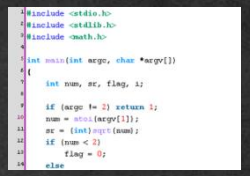
0	1	2	3	4	5	6	7
24	12	25	37	48	57	86	92
							pivô

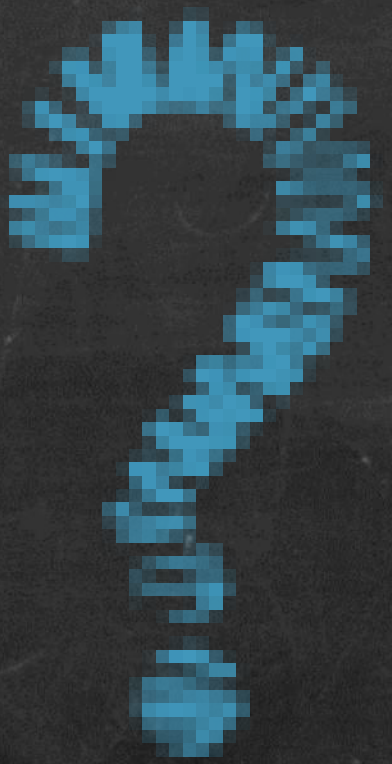
Quick Sort

- O processo de ordenação termina!

0	1	2	3	4	5	6	7
24	12	25	37	48	57	86	92

Quick Sort- Algoritmo






That's all Folks