Algoritmia Aplicada

Ano lectivo 2011-2012

Aula 11 – Ordenação

Sumário

- Ordenação
  - Métodos elementares de ordenação
    - Selecção
    - Inserção
    - *ShellSort*
    - *BubbleSor*t
  - Método *QuickSort*
  - Método *HeapSort*
  - Operação de selecção: Métodos fundamentais
  - Operação de fusão. Ordenação por fusão (*MergeSort*)

Ordenação

- **Objectivo:**
  - Ordenação de registos de ficheiros com **chave**
  - Chaves (pequena parte dos registos) usadas para controlar a ordenação
  - Re-arranjar os registos de modo a deixá-los por uma ordem (numérica ou alfabética)

Ordenação

- **Algoritmos elementares** → **pequenos conjuntos de dados**

  N - nº elementos a ordenar (pequeno se N<500)

- **"Dividir para reinar"** → **grandes conjuntos de dados**

Ordenação

- **Tipos de ordenação:**
  - **Se ficheiro cabe em memória (array)**
    Ordenação interna – qualquer registo pode ser acedido
  - **Se ficheiro em disco**
    Ordenação externa – registos têm que ser acedidos sequencialmente ou em grandes blocos
- **Tempo de execução**

Algoritmos elementares	**ordem N^2**
Algoritmos elaborados	**ordem N log N**
• **Usando algumas propriedades digitais das chaves**	**ordem N**

Ordenação

- **Espaço de memória**
  - **Ordenam no lugar, sem usar memória adicional (apenas uma pequena tabela)**
  - **Ordenam com recurso a listas; utilizam N ponteiros extra**
  - **Precisam de duplicar a informação a ser ordenada**
- **Estabilidade**
  - **Método é estável se preserva a ordem relativa das chaves iguais do ficheiro**
  - **Pode ser forçada juntando um número de ordem à chave**
- **Exemplos: com inteiros ou caracteres**
- **Se registos são grandes → ordenação indirecta sobre array de apontadores ou índices**

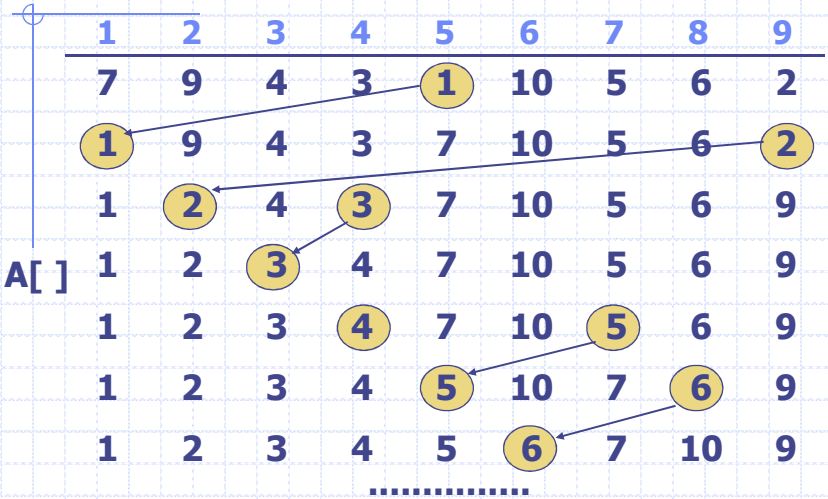
Ordenação – Selecção

- **Procedimento:**
  - **Encontrar o menor dos elementos do array e trocá-lo com o elemento na primeira posição**
  - **Procurar o segundo menor e trocá-lo com o elemento na segunda posição**

  .....
  - **Procurar o penúltimo menor e trocá-lo com o elemento que está na penúltima posição**

Ordenação – Selecção

- **Exemplo:** **N=9**



Ordenação – Selecção

- **Algoritmo:**

N-1 passagens

```
Procedure OrdenaSeleccao  (A, N)
  DO FOR I=1 TO N-1
        MIN ← I
        DO FOR J=I+1 TO N
              IF A[J] < A[MIN]
              THEN MIN ← J
        IF MIN <> I
        THEN  T ← A[MIN]
              A[MIN] ← A[I]
              A[I] ← T
```

SWAP
A[MIN] <=> A[I]

Ordenação de um vector A[] de N elementos por ordem crescente

Ordenação – Selecção

- **Conclusões:**
  - **O algoritmo de selecção é de ordem N^2**
  - **Fazem-se em média $N^2/2$ comparações**

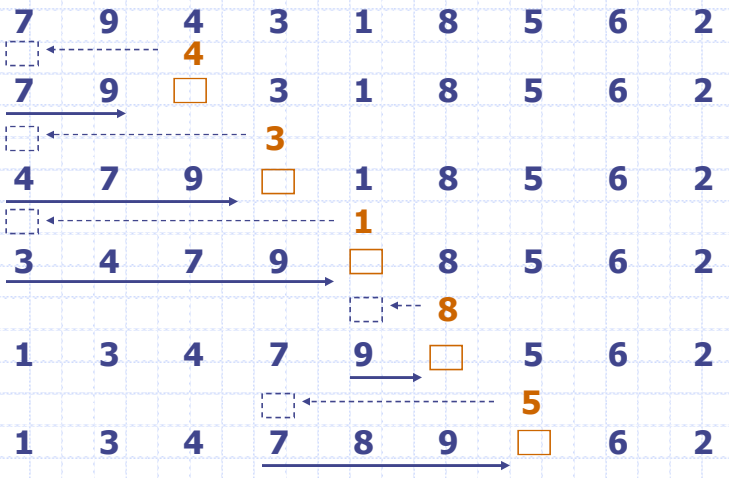
Ordenação – Inserção

- **Procedimento:**
  - **Considera-se um elemento de cada vez que é colocado no seu lugar entre os que já estão ordenados**
  - **Seleccionar um elemento do array como se o seu lugar ficasse em aberto – procura-se o seu lugar e desloca-se para a direita os elementos maiores (um a um)**
  - **Se a pesquisa do lugar entre os já ordenados for feita da direita para a esquerda e, se o elemento a colocar for o menor, podemos ultrapassar o limite inferior do array**

    SOLUÇÃO: Usar uma sentinela em A[0] – evita a inclusão de um teste

Ordenação – Inserção

- **Exemplo:**



Ordenação – Inserção

- **Algoritmo:**

N-1 passagens

```
Procedure OrdenaInsercao  (A, N)
  DO FOR I=2 TO N
        V ← A[I]
        J ← I
        DO WHILE V<A[J-1]
              A[J] ← A[J-1]
              J ← J-1
        A[J] ← V
```

Assume-se que em A[0] está uma sentinela

Ordenação de um vector A[] de N elementos por ordem crescente

Ordenação – Inserção

- **Conclusões:**
  - **O algoritmo inserção é de ordem $N^2/2$**
  - **O ciclo interno é executado cerca de $N^2/2$ vezes. A "Inserção" é feita em média a meio caminho de um subconjunto de tamanho N/2**
    - **É de ordem N para elementos iniciais já ordenados**
    - **É de ordem N^2 para ordem inicial inversa**

Ordenação – ShellSort

- ✓ **INSERÇÃO – método lento porque troca apenas elementos adjacentes**
- ✓ **Se o elemento menor do array está na última posição, são necessários N passos para o colocar no sítio**

➢ **O ShellSort é uma extensão simples ao método de Inserção, permitindo trocas entre elementos distantes**

Ordenação – ShellSort

➢ **Procedimento:**

- **Compara-se o 1º elemento com o de ordem N e arranja-se-os, depois o 2º com o de ordem N+1 e arranja-se-os, e assim sucessivamente, começando por todos os elementos**
- **Resulta num ficheiro re-arranjado de tal modo que, considerados cada N-ésimo elemento a começar de qualquer ponto, os elementos estão ordenados.**
- **O ficheiro obtido deste modo é considerado de ordem h**

Ordenação – ShellSort

- **Um ficheiro de ordem h é um grupo de ficheiros independentes e ordenados, co-existindo entremeados**
- **Ao fazer-se esta ordenação à distância h, movemos elementos de mais longe o que facilita a ordenação para h menores**
- **Usando este procedimento para qualquer sequência de h's que termine em 1, obtemos um ficheiro ordenado**
- **Sequência de h's que empiricamente se provou funcionar bem:**

  ... 1093, 364, 121, 40, 13, 4, 1

Ordenação – ShellSort

- **Valores do parâmetro h (incrementos)**

h= 1, 4, 13, 40, 121, 364, 1093 ...

Por esta ordem
h ← h div 3

Por esta ordem
h ← 3 h+1

```
h=1
DO
   h=3*h+1
UNTIL h>N
```

←

Para gerar o último valor de h, considerando um vector de N elementos a ordenar

Ordenação – ShellSort

- **Exemplo:**

Incrementos

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
h=13	1-14	A													L	
	2-15		E													S
		A	E	O	R	T	I	N	G	E	X	A	M	P	L	S
	1-5-9-13															
	A-T-E-P	A				E				P				T		
	2-6-10-14															
h=4	E-I-X-L		E				I				L				X	
	3-7-11-15															
	O-N-A-S			A				N				O				S
	4-8-12															
	R-G-M				G				M				R			
		A	E	A	G	E	I	N	M	P	L	O	R	T	X	S
h=1	1-2-3-4-...-15	A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Ordenação – ShellSort

- **O último passo é uma inserção, mas os elementos só se deslocam uma posição**
- **A forma funcional depende dos incrementos → difícil comparação analítica**
  Algumas sequências de valores de h funcionam melhor que outras
- **É dos métodos mais utilizados**

Ordenação – BubbleSort

- **Procedimento:**
  - Percorre-se todo o ficheiro e trocam-se elementos adjacentes, se necessário.
  - Quando não houver mais trocas, o ficheiro está ordenado

7	5	4	3	6	8	9	1	2
7	5	4	3	6	8	1	9	2
7	5	4	3	6	1	8	9	2
		.....						
1	7	5	4	3	6	8	9	2
1	7	5	4	3	6	8	2	9
		.....						
1	2	7	5	4	3	6	8	9
		.....						

Ordenação – QuickSort

- **Características**
  - Ordena no lugar
  - Ordem: N log N
  - Ciclo interno simples
    - Recursivo
    - Pior caso é de ordem N^2 (ficheiro ordenado)
    - Frágil (pode funcionar mal em certos casos)
- **Técnica – "DIVIDIR PARA REINAR"**
  - Parte o ficheiro em dois e ordena-os separadamente:

Ordenação – QuickSort

- **Técnica – "DIVIDIR PARA REINAR"**
  - Parte o ficheiro em dois e ordena-os separadamente
  - O local da partição depende do ficheiro

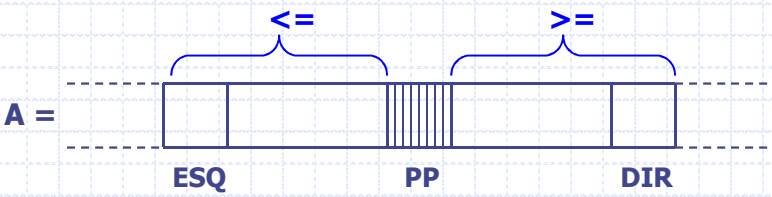
1 N ← Início

```
ORDENAR (ESQ, DIR)
    PONTOPARTIÇÃO = PARTIÇÃOENTRE (ESQ, DIR)
    ORDENAR (ESQ, PONTOPARTIÇÃO-1);
    ORDENAR (PONTOPARTIÇÃO+1, DIR);
```

- Os parâmetros **ESQ** e **DIR** delimitam o subficheiro a ordenar dentro do ficheiro original
- Supõe-se: **ESQ<DIR**

Ordenação – QuickSort

- **Observações: Ponto de Partição (PP)**
  - O elemento A [PP] está na sua posição final no ficheiro, e será ignorado das vezes seguintes
  - Todos A[ESQ], ..., A[PP-1] <= A[PP]
  - Todos A[PP+1], ..., A[DIR] >= A[PP]



Ordenação – QuickSort

- **Implementação**
  - Escolhe-se A[DIR] arbitrário, que irá para a sua posição final
  - Varre-se a partir da esquerda até se encontrar elemento >= A[DIR]
  - Varre-se a partir da direita até se encontrar elemento <= A[DIR]
  - Estes dois elementos que provocaram a paragem estão fora do seu lugar e devem ser trocados
  - Continua-se até os apontadores de varrimento se cruzarem
  - Troca-se A[DIR] com elemento à esquerda do subficheiro da direita

Ordenação – QuickSort

- **Exemplo**

A[DIR]=A[15]=E → Vai para a sua posição final

Ordenação – QuickSort

- **Implementação (cont.)**
  - **O algoritmo continua ordenando os dois subficheiros à esquerda e direita recursivamente**
  - **O ciclo interno só incrementa os ponteiros e compara os elementos do ficheiro com uma constante -> rapidez do algoritmo**
  - **Cada elemento é colocado no seu lugar ao ser usado como elemento de partição**
  - **Quando há chaves iguais param-se os apontadores**

Ordenação – QuickSort

- **Algoritmo**

```
Procedimento QuickSort  (L, R)
 IF R > L
 THEN I ← PARTICAO (L, R)
 CALL QuickSort (L, I-1)
 CALL QuickSort (I+1, L)
```

- **Notas**
  - **Para um vector com N elementos, o procedimento é chamado inicialmente assim: CALL QuickSort (1,N)**
  - **PARTICAO (L, R) é uma função que determina exactamente qual a posição final (definitiva!) para o elemento que inicialmente está em R (elemento A[R])**

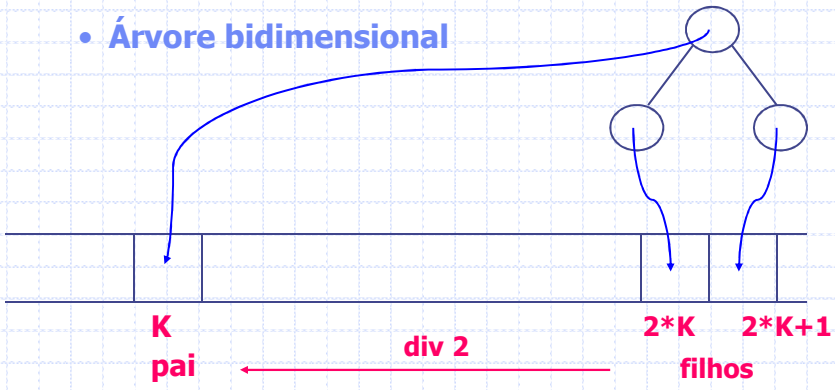
Filas de prioridade

- **Operações**
  - **Inserção de um novo elemento**
  - **Remoção do elemento de maior valor**

  Difere de uma fila normal, onde o elemento a remover é o mais antigo (o que está há mais tempo na fila)
- **Aplicações**
  - **Sistema de gestão de tarefas – tarefa a ser executada é a de mais alta prioridade**
  - **Cálculos numéricos – números de maior valor têm de ser processados primeiro**

Heap

- **Estrutura usada para suportar operações com filas de prioridade**
- **Representação**
  - **Array**
  - **Árvore bidimensional**



Heap

- **Condições do *heap***
  - **A chave em cada nó tem de ser superior (ou igual) às chaves dos seus filhos (se existirem)**
  - **$\Rightarrow$ chave maior se situe na raiz**

- **Todas as operações sobre filas de prioridade (excepto a fusão) podem ser executadas em tempo logarítmico usando *heaps***

Heap

- **Operação de inserção**
  - **Incrementa-se o tamanho N do *heap* de uma unidade**
  - **Registo novo é colocado em A[N], o que pode violar a condição de *heap***
  - **Se ocorrer violação, i.e., registo novo é superior ao pai, esta é corrigida por troca do registo novo com o seu pai**
    - **Isto pode provocar outra violação, que pode ser resolvida do mesmo modo**

Heap

- **Operação de eliminação**
  - **Decrementar o tamanho N do *heap* de uma unidade, deixando de haver lugar para o elemento que estava na última posição do array**
  - **Maior elemento (A[1]) é retirado através de troca com o elemento que estava em A[N]**
  - **Restaurar o *heap*, se necessário (se houver violação da condição de *heap*)**

Ordenação – HeapSort

- **Ideia: Construir um *heap* com os elementos a ordenar e removê-los por ordem**
  - **Não usa memória adicional**
  - **Tempo de execução: M log M para qualquer entrada (M é o nº elementos a ordenar)**
  - **Ciclo interno mais longo que QuickSort (demora em média o dobro)**

Ordenação – HeapSort

- **Ideia:** Construir um *heap* com os elementos a ordenar e removê-los por ordem

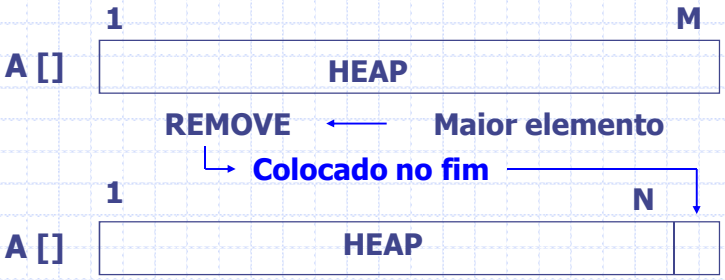
  N → tamanho do *heap*
  M → nº elementos a ordenar

- **Implementação**
  - Construir o *heap* com M operações de *inserção*
  - Efectuar depois M operações de *remoção*, colocando o elemento removido no lugar deixado vago pelo *heap* que vai encolhendo

```
N ← 0
DO FOR K=1 to M
   CALL INSERT (A[K])
DO FOR K=M to 1 STEP -1
    A[K]=REMOVE
```


Ordenação – HeapSort

- **Depois da construção do HEAP**




```
N ← 0
 DO FOR K=1 to M
   CALL INSERT (A[K])
DO FOR K=M to 1 STEP -1
    A[K] ← REMOVE
```

Ordenação – HeapSort

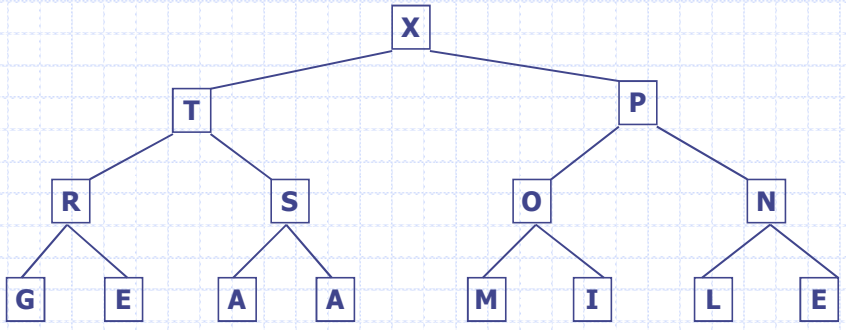
- **Na prática é melhor construir-se o heap, depois de lidos os valores a ordenar, do fim para o princípio do array**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
						N							L	E
					P						M	I		
				X					T	A				
			R				G	E						
		P			O	N								
	X		R	T										
X	T	P	R	S	O	N	G	E	A	A	M	I	L	E

Ordenação – HeapSort

- **Exemplo:**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
X	T	P	R	S	O	N	G	E	A	A	M	I	L	E



***Heap* depois das 15 inserções**

Ordenação – HeapSort

- **HEAPS indirectos**
  - Em vez de se moverem os registos em A[ ], cria-se um array **HEAP [ ]** de índices de A[ ]
  - **A[HEAP[K]]** é a **chave do elemento K** do HEAP
  - É conveniente outro array **INV[K]** que nos dê a posição no HEAP para o elemento de ordem K

A[K]	A	S	O	R	T	I	N	G	E	X	A	M	P	L
HEAP[K]	10	5	13	4	2	3	7	8	9	1	11	12	6	14
A[HEAP[K]]	X	T	P	R	S	O	N	G	E	A	A	M	I	L
INV[K]	10	5	6	4	2	13	7	8	9	1	11	12	3	14
A[K]	A	S	O	R	T	I	N	G	E	X	A	M	P	L

Selecção e Fusão

- **Selecção - Encontrar o k elemento menor (ou os k menores elementos)**
- **Fusão - Combinar dois ficheiros ordenados num único, também ordenado**
- **Exemplo de Selecção:**
  - Encontrar a mediana de uma série de valores , por exemplo, as notas dos alunos
    (o elemento que é maior que metade dos elementos e menor que a outra metade)
- **Exemplo de Fusão:**
  - Obter uma lista ordenada com todos os alunos a partir das listas das turmas

Selecção e fusão são complementares no sentido em que uma separa e outra junta ficheiros

Operação de Selecção: Métodos fundamentais

- **Poucos elementos –** Ordenação por Selecção
- **Mais elementos –** Filas de prioridade
  - Inserem-se K elementos
  - Trocam-se os N-K maiores usando os restantes, deixando os K menores na fila
- **No QuickSort**
  - a partição modifica A[ ], devolvendo I tal que A[1]... A[I-1] são <= A[I] e A[I+1] ... A[N] são > A[I]

Operação de Selecção: Méts. fundamentais

- **No QuickSort**
  - Se procurarmos os K menores e K=I, o problema está resolvido
  - Se K>I, procuramos à direita

FORMULAÇÃO RECURSIVA

SELECT (1, N, (N+1) div 2)

Parte o array no seu valor mediano

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	E	T	I	N	G	O	X	S	M	P	L	R
				L	I	N	G	O	P	M	R	X	T	S
				L	I	G	M	O	P	N				
							M							

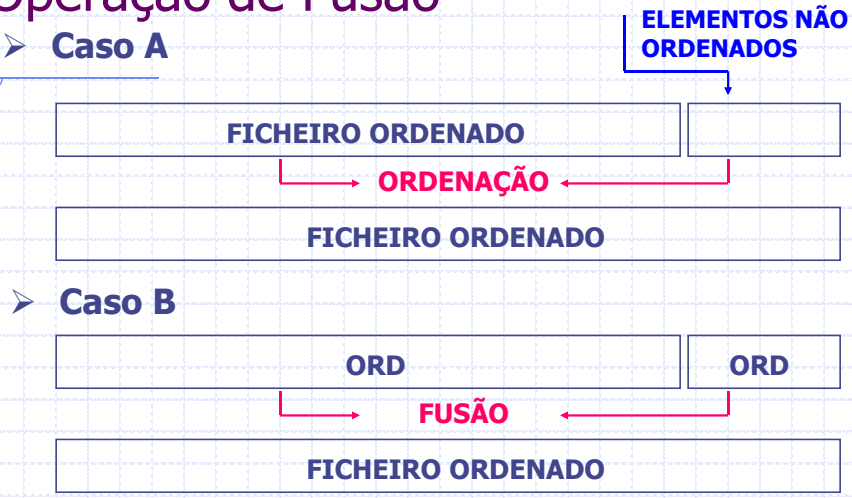
Todos menores à esquerda (N.ORD.)

Todos maiores à direita (N. ORD.)

Mediana no lugar

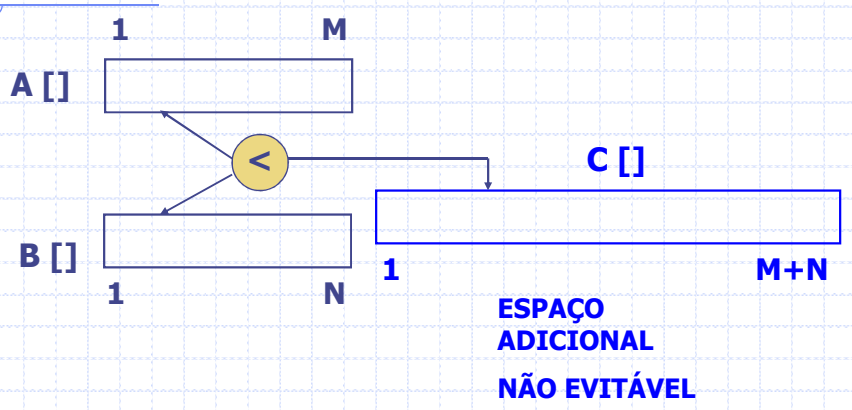
Operação de Fusão

- **Caso A**



- **Caso B**

Operação de Fusão



Ordenação por Fusão

➢ **Procedimento:**

- **Divide-se o ficheiro ao meio, ordena-se recursivamente cada uma das partes e faz-se a sua fusão**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
	O	S												
A	O	S												
			R	T										
					I	N								
			I	N	R	T								
A	I	N	O	R	S	T								
							E	G						
									A	X				
							A	E	G	X				
											M	P		
													E	L
											E	L	M	P
							A	E	E	G	L	M	P	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Ordenação por Fusão

➢ **Outra versão:**

Bottom-up mergesort

- **Percorre ficheiro e faz fusão de listas de dimensão 1, produzindo sublistas ordenadas de tamanho 2**
- **Percorre de novo e faz fusão das de 2 em 4**
- **E assim sucessivamente**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	S	O	R	I	T	G	N	E	X	A	M	L	P	E
A	O	R	S	G	I	N	T	A	E	M	X	E	L	P
A	G	I	N	O	R	S	T	A	E	E	L	M	P	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Ordenação por Fusão

- **Conclusões**
  - **São necessários log N passos para ordenar N elementos, uma vez que cada passo duplica o tamanho dos subficheiros ordenados**
  - **Tempo de execução: N log N**
  - **A principal vantagem do MERGESORT é a estabilidade**
  - **A principal desvantagem é o espaço necessário**

Próxima aula

- Pesquisa