PANC: Projeto e Análise de Algoritmos

Aula 04: Paradigma de Divisão e Conquista e Ordenação utilizando MergeSort (Lógica e Complexidade)

Breno Lisi Romano

03 de Março de 2020

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus São João da Boa Vista

Sumário

- Revisão de Conteúdo
- Divisão e Conquista
- Ordenação por Intercalação (*Merge Sort*)
  - Proposta de Divisão e Conquista Adotada
  - Lógica
  - Exemplo Prático
  - Pseudocódigo
  - Implementação
- Análise da Complexidade do *Merge Sort*

Recapitulando...

- **T(n)** é a função de complexidade que representa a **medida de custo da execução** de um **algoritmo** para uma instância de **tamanho n**:
  - A função de **complexidade de tempo T(n)** mede o tempo necessário para executar um algoritmo **(número de instruções) → Quantidade de Operações Executadas**

- **Ordenação por Inserção (*Insertion Sort*):**
  - Caracterizada pelo **princípio** no qual se **divide** o **array** em **dois segmentos**: um já **ordenado** e o outro **não ordenado**
    - O **progresso** se desenvolve em **n-1 interações**
    - Em cada interação: um **elemento do segmento não ordenado** é **transferido** para o **primeiro segmento**, e **inserido** na **posição correta** em relação aos demais elementos já existentes
  - **Análise da Complexidade – T(n):**
    - **Melhor Caso:** T(n) é Linear – O(n)
    - **Pior Caso:** T(n) é Quadrático – $O(n^2)$

Divisão e Conquista (1)

- "*Divide-and-Conquer is perhaps the most commonly used algorithm design technique in computer science.*
- *Faced with a big problem P, divide it into smaller subproblems, solve these sub-problems, and combine their solutions into a solution for P.*
- *But how do you solve the smaller problems?*
- *Simply divide each of the small problems into smaller problems, and keep doing this until the problems become so small that it is trivial to solve them.*
- *Sound like recursion? Not surprisingly, a recursive procedure is usually the easiest way of implementing divide-and-conquer*"
- - **Ian Parberry, *Problems on Algorithms***

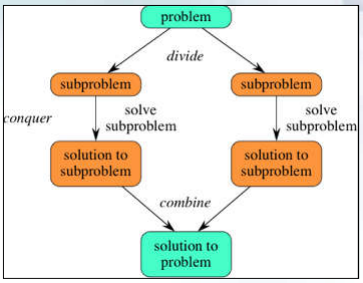
Divisão e Conquista: Definição (2)

- Divisão e Conquista (ou Dividir e Conquistar, ou ainda, D&C) é um **paradigma de solução de problemas** no qual tentamos **simplificar a solução do problema** original **dividindo-o em subproblemas** menores e **resolvendo-os** (ou "conquistando-os") separadamente
- O processo:
  - **Dividir** o problema original em **subproblemas** – normalmente com a metade (ou algo próximo disto) do tamanho do problema original, porém com a mesma estrutura
  - **Conquistar**, ou determinar a solução dos subproblemas, comumente, de maneira recursiva – que agora se tornam mais "fáceis"
  - Se necessário, **combinar** as soluções dos subproblemas para produzir a **solução completa** para o problema original

5

Divisão e Conquista: Definição (3)

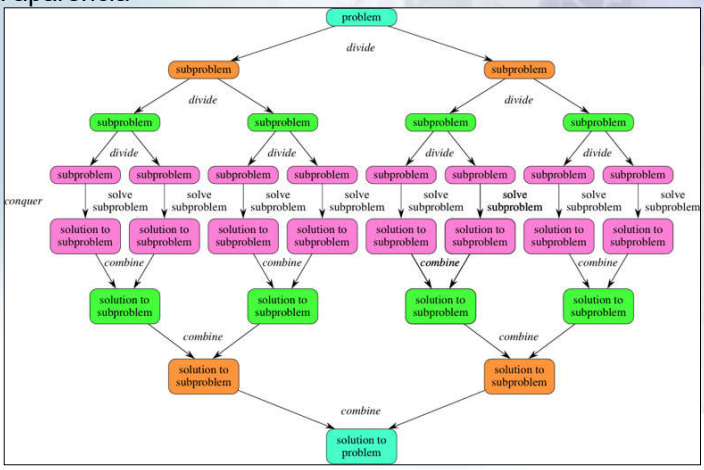
- **Passos** de um algoritmo de Divisão e Conquista: **Dividir**, **Conquistar**, **Combinar**
- **Ilustração**: Visualizar um passo, assumindo que cada passo de dividir cria dois subproblemas
  - Alguns algoritmos de dividir-e-conquistar criam mais de dois



6

Divisão e Conquista: Definição (4)

- **Ilustração**: Se expandirmos mais duas etapas recursivas, ele terá esta aparência



Como se cria pelo menos dois subproblemas, um algoritmo de Divisão e Conquista faz múltiplas chamadas recursivas

Divisão e Conquista: Observação (5)

- Conforme descrito anteriormente, pode-se resolver subproblemas de estrutura igual de maneira recursiva
- Eventualmente, é necessário resolver alguns subproblemas diferentes do problema original em termos de estrutura
- Consideram-se estes subproblemas como parte do processo de combinar as soluções

Divisão e Conquista: Utilização (6)

- Existem **quatro condições** que indicam se o paradigma Divisão e Conquista pode ser aplicado com sucesso:
  - Deve ser possível **dividir o problema** em **subproblemas**
  - A **combinação** de **resultados** deve ser **eficiente**
  - Os **subproblemas** devem possuir **tamanhos parecidos** dentro de um mesmo nível
  - A **solução** dos **subproblemas** são **operações repetidas** ou correlacionadas

Divisão e Conquista: Solução Genérica (7)

```
D&CGenerico(x)
Entrada: (sub)problema x
se x é o caso base então
    retorna resolve(x);
senão
    Divida x em n subproblemas x_0, x_1, ..., x_{n-1};
    para i ← 0 até n − 1 faça
        y_i ← D&CGenerico(x_i);
    fim
    Combine y_0, y_1, ..., y_{n-1} em y;
    retorna y;
fim
```

Divisão e Conquista: Definição (2)

- Divisão e Conquista (ou Dividir e Conquistar, ou ainda, D&C) é um **paradigma de solução de problemas** no qual tentamos **simplificar a solução do problema** original **dividindo-o em subproblemas** menores e **resolvendo-os** (ou "conquistando-os") separadamente
- O processo:
  - **Dividir** o problema original em **subproblemas** – normalmente com a metade (ou algo próximo disto) do tamanho do problema original, porém com a mesma estrutura
  - **Conquistar**, ou determinar a solução dos subproblemas, comumente, de maneira recursiva – que agora se tornam mais "fáceis"
  - Se necessário, **combinar** as soluções dos subproblemas para produzir a **solução completa** para o problema original

11

Aplicação do paradigma de Divisão e Consquisa.....

ORDENAÇÃO POR INTERCALAÇÃO (*MERGE SORT*)

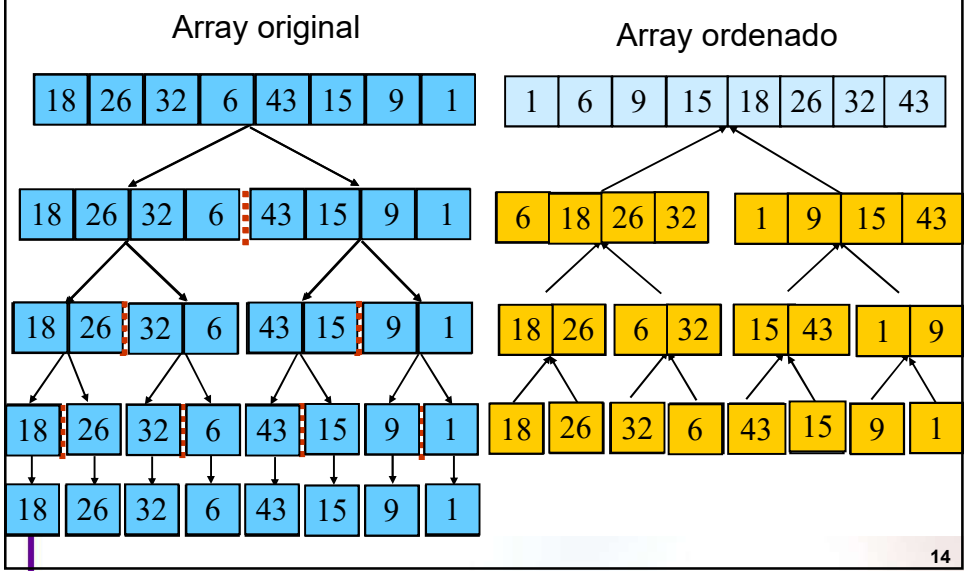
12

Merge Sort: Divisão e Conquista

- *Mergesort* é um **algoritmo** para resolver o problema de **ordenação de arrays** e um exemplo clássico do **uso do paradigma de Divisão e Conquista** (*to merge* = intercalar)
  - Duas abordagens: **Top-Down (Recursiva)** e Bottom-Up (Iterativa)
- **Descrição** do ***Mergesort*** em alto nível (**Top-Down**):
  - **Divisão**: Divide a sequência de n elementos que deve ser ordenada em duas subsequências de n/2 elementos cada uma
  - **Conquista**: Ordena as duas subsequências recursivamente, utilizando a ordenação por intercalação (Algoritmo Mergesort)
  - **Combinação**: Intercala as duas subsequências ordenadas para produzir a resposta ordenada (Algoritmo Intercala)
  - **Condição de parada da Recursão**: quando for ordenar apenas um elemento, este caso será a sub-solução elementar

13

Merge Sort: Ilustração para um Array (n=8)



14

Merge Sort: Algoritmo

- ***Mergesort***: O objetivo é rearranjar um array A[p...r], com p ≤ *r, em* ordem crescente

```
MERGESORT(A, p, r)
1  se p < r                                   Divisão
2     então q ← ⌊(p + r)/2⌋
3           MERGESORT(A, p, q)                Conquista
4           MERGESORT(A, q + 1, r)
5           INTERCALA(A, p, q, r)             Combinação
```

	p			q				r
A	66	33	55	44	99	11	77	22	88

15

Merge Sort: Ilustração do Algoritmo (1)

- ***Mergesort***: O objetivo é rearranjar um array A[p...r], com p ≤ *r, em* ordem crescente

```
MERGESORT(A, p, r)
1  se p < r
2     então q ← ⌊(p + r)/2⌋
3           MERGESORT(A, p, q)
4           MERGESORT(A, q + 1, r)
5           INTERCALA(A, p, q, r)
```

	p			q				r
A	33	44	55	66	99	11	77	22	88

16

Merge Sort: Ilustração do Algoritmo (2)

- ***Mergesort***: O objetivo é rearranjar um array A[p...r], com p ≤ *r, em* ordem crescente

```
MERGESORT(A, p, r)
1  se p < r
2     então q ← ⌊(p + r)/2⌋
3           MERGESORT(A, p, q)
4           MERGESORT(A, q + 1, r)
5           INTERCALA(A, p, q, r)
```

	p				q				r
A	33	44	55	66	99	11	22	77	88

Merge Sort: Ilustração do Algoritmo (3)

- ***Mergesort***: O objetivo é rearranjar um array A[p...r], com p ≤ *r, em* ordem crescente

```
MERGESORT(A, p, r)
1  se p < r
2     então q ← ⌊(p + r)/2⌋
3           MERGESORT(A, p, q)
4           MERGESORT(A, q + 1, r)
5           INTERCALA(A, p, q, r)
```

	p				q				r
A	11	22	33	44	55	66	77	88	99

Merge Sort: Algoritmo Intercala()

- O que significa intercalar dois (sub)arrays ordenados?
- **Problema:** Dados A[p...q] e A[q+1...r] crescentes, rearranjar A[p...r] de modo que ele fique em ordem crescente

Entrada:

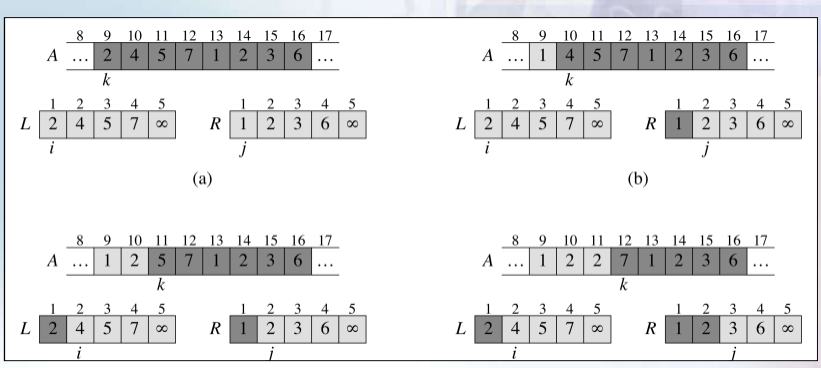
	p				q				r
A	22	33	55	77	99	11	44	66	88

Saída:

	p				q				r
A	11	22	33	44	55	66	77	88	99

19

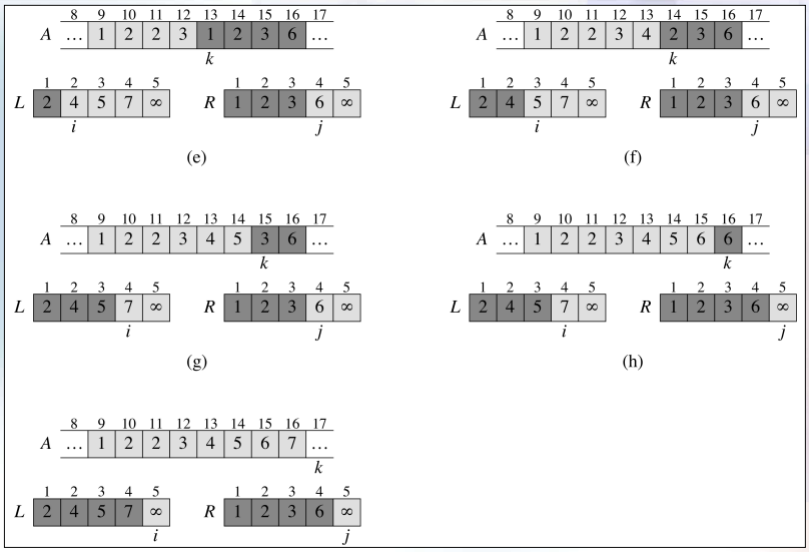
Merge Sort: Algoritmo Intercala() – Com Sentinela (1)



Ref.: Algoritmos – Cormen (2012)

20

Merge Sort: Algoritmo Intercala() – Com Sentinela (2)



Ref.: Algoritmos – Cormen (2012)

Merge Sort: Algoritmo Intercala() – Com Sentinela (3)

```
INTERCALA(A, p, q, r)
 1: n1 ← q − p + 1
 2: n2 ← r − q
 3: sejam L[1..n1 + 1] e R[1..n2 + 1] novos vetores
 4: para i ← 1 até n1 faça
 5:     L[i] ← A[p + i − 1]
 6: para j ← 1 até n2 faça
 7:     R[j] ← A[q + j]
 8: L[n1 + 1] ← ∞
 9: R[n2 + 1] ← ∞
10: i ← 1
11: j ← 1
12: para k ← p até r faça
13:     se L[i] ≤ R[j] então
14:         A[k] ← L[i]
15:         i ← i + 1
16:     senão
17:         A[k] = R[j]
18:         j ← j + 1
```

Merge Sort: Algoritmo Intercala() – Sem Sentinela (1)

A	11	22	33	44	55	66	77	88	99
					j	*i*			
B	22	33	55	77	99	88	66	44	11

23

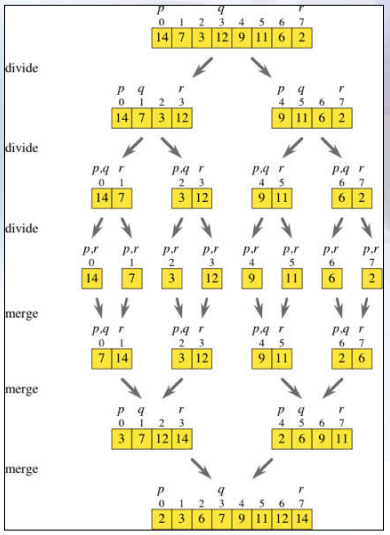
Merge Sort: Algoritmo Intercala() – Sem Sentinela (2)

```
INTERCALA(A, p, q, r)
 1  para i ← p até q faça
 2      B[i] ← A[i]
 3  para j ← q + 1 até r faça
 4      B[r + q + 1 − j] ← A[j]
 5  i ← p
 6  j ← r
 7  para k ← p até r faça
 8      se B[i] ≤ B[j]
 9          então A[k] ← B[i]
10                i ← i + 1
11          senão A[k] ← B[j]
12                j ← j − 1
```

24

Merge Sort: Ilustração do Algoritmo Exemplo Completo

- **A[] = [14, 7, 3, 12, 9, 11, 6, 2]**



25

Merge Sort: Análise da Complexidade Algoritmo Intercala

- **Pior Caso – T(n)**:

Entrada:

	p				q				r
A	22	33	55	77	99	11	44	66	88

Saída:

	p				q				r
A	11	22	33	44	55	66	77	88	99

```
INTERCALA(A, p, q, r)
1   para i ← p até q faça
2       B[i] ← A[i]
3   para j ← q + 1 até r faça
4       B[r + q + 1 − j] ← A[j]
5   i ← p
6   j ← r
7   para k ← p até r faça
8       se B[i] ≤ B[j]
9           então A[k] ← B[i]
10                i ← i + 1
11          senão A[k] ← B[j]
12                j ← j − 1
```

- **Tamanho da Entrada:** n = r – p + 1
- **Complexidade T(n):** O(n) → Linear

26

Merge Sort: Análise da Complexidade Algoritmo Mergesort (1)

- **Pior Caso – T(n):** Assumir que o Tamanho do Array é Par

```
MERGESORT(A, p, r)
1   se p < r
2       então q ← ⌊(p + r)/2⌋
3             MERGESORT(A, p, q)
4             MERGESORT(A, q + 1, r)
5             INTERCALA(A, p, q, r)
```

Linha	Consumo de Tempo	
1	1	
2	1	
3	T(n/2)	$T(\lceil n/2 \rceil)$
4	T(n/2)	$T(\lfloor n/2 \rfloor)$
5	n: Complexidade do Intercala	
T(n)	**T(n/2) + T(n/2) + O(n) + 2**	

Se não fosse par!

27

Merge Sort: Análise da Complexidade Algoritmo Mergesort (2)

- **Fórmula de Recorrência** (ou seja, uma fórmula definida em termos de si mesma)**:**
  - **T(1) = O(1)** se n=1
  - **T(n) = T(n/2)+ T(n/2)+ O(n)** se n>1

- Em geral, ao utilizar-se do **paradigma de Divisão e Conquista**, adota-se **recursividade** → **Complexidade** T(n) é uma **Fórmula de Recorrência**
  - Precisamos aprender a **resolver recorrência** → Encontrar uma **"Fórmula Fechada"** para T(n)
  - Veremos em outras aulas como resolver recorrência
  - Mas queremos calcular a Complexidade T(n) do *MergeSort*

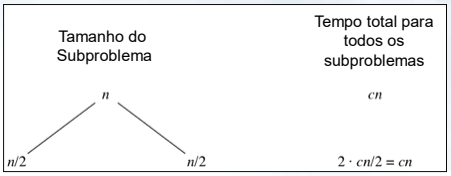
28

Merge Sort: Análise da Complexidade Algoritmo Mergesort (3)

- **Fórmula de Recorrência:**
  - **T(1) = c** se n=1
  - **T(n) = 2.T(n/2) + c.n** se n>1

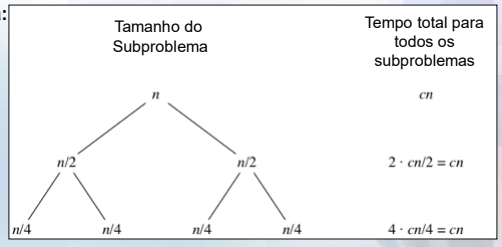
  onde c é um a constante para o tempo exigido em resolver problemas de tamanho 1, bem como o tempo por elemento do (sub)array para as etapas de dividir e combinar

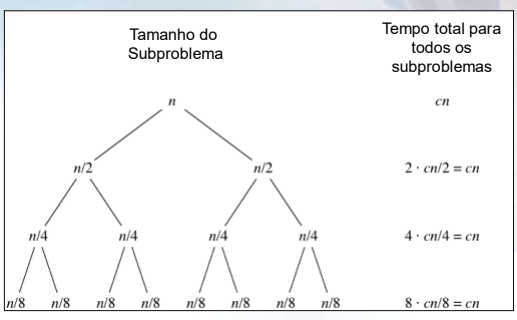
- Vamos resolver com uma "Árvore de Recorrência":



Merge Sort: Análise da Complexidade Algoritmo Mergesort (4)

- **Fórmula de Recorrência:**
  - **T(1) = c**
  - **T(n) = 2.T(n/2) + c.n**

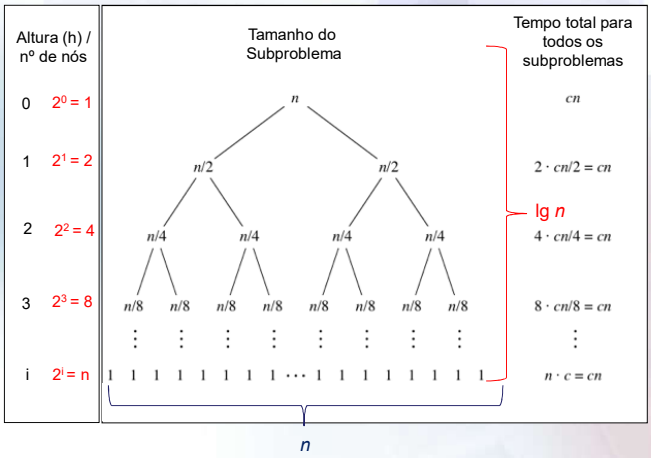




Instituto Federal de São Paulo – IFSP São João da Boa Vista

Merge Sort: Análise da Complexidade Algoritmo Mergesort (5)

- **Fórmula de Recorrência:**
  - **T(1) = c**
  - **T(n) = 2.T(n/2) + c.n**
- **Quantos nós tem na árvore com a altura i?**
  - n elementos
- **E qual a relação com a altura da árvore?**
  - $2^i = n$
  - $\lg 2^i = \lg n$
  - $i = \lg n$ ($\lg n = \log_2 n$)
- **Quantos níveis tem a árvore?**
  - lg n + 1 (Obs.: +1 → h = 0)



- **Complexidade T(n):** c.n . (lg n + 1) → T(n) = c.n lg n + c.n → **T(n) = O(n.lg n)**

31

Instituto Federal de São Paulo – IFSP São João da Boa Vista

Merge Sort: Conclusão

- ***MergeSort*** necessita de espaço O(n) para **armazenar o array temporário**
- **Complexidade** T(n) do *MergeSort*:
  - **Pior Caso** - T(n): O (n.log n)
  - **Melhor Caso** - T(n): O (n.log n)
  - **Caso Médio** - T(n): O (n.log n)
- Pode-se dizer que o **UP** de tempo para a **ordenação** é **O(n.log n)**
- Como o **UP** e o **LB** da **ordenação** são de **mesma ordem**:
  - A **ordenação** é um problema **computacionalmente resolvido**
  - Qualquer **algoritmo** que ordena n números em tempo **O(n.log n) é ótimo**

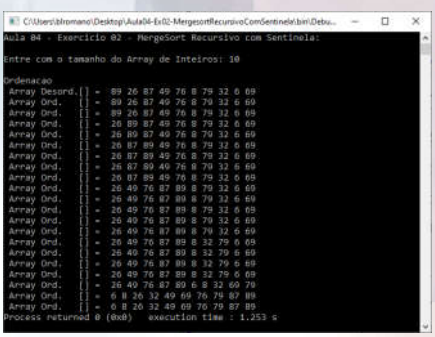
32

Trabalhos para Casa (1)

- **Exercício 01 – Ilustração Prática do *Mergesort***:
  - Utilizando a Figura do Slide 25 (Ilustração do Algoritmo Exemplo Completo) como modelo, realize uma ilustração completa da operação de Ordenação por Intercalação (*Mergesort*) para o array A = ⟨3, 41, 52, 26, 38, 57, 9, 49⟩
    - Deixar claro as 03 etapas do algoritmo: Divisão, Conquista e Combinação
    - Salvar em um arquivo .doc e submeter

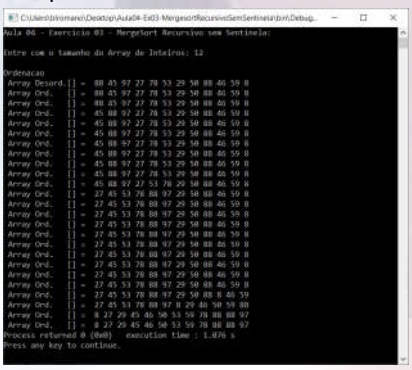
Trabalhos para Casa (2)

- **Exercício 02 – *MergeSort* Recursivo com Sentinela**:
  - Implementar o MergeSort em C (Top-Down / Recursivo) utilizando-se o algoritmo apresentado nos slides como base e também desenvolver o algoritmo Intercala adotando-se Sentinela para controle
    - Deve-se criar um algoritmo que gere um Array de números inteiros aleatórios de tamanho N, fornecido pelo usuário
    - Deve-se criar duas funções para realizar a operação de ordenação por intercalação: MergeSortRecursivo() e IntercalaComSentinela()
    - Deve-se imprimir cada etapa da ordenação na Console



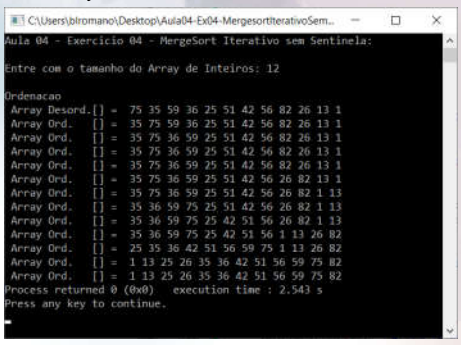
Trabalhos para Casa (3)

- **Exercício 03 – *MergeSort* Recursivo sem Sentinela**:
  - Implementar o MergeSort em C (Top-Down / Recursivo) utilizando-se o algoritmo apresentado nos slides como base e também desenvolver o algoritmo Intercala adotando-se Sentinela para controle
    - Deve-se criar um algoritmo que gere um Array de números inteiros aleatórios de tamanho N, fornecido pelo usuário
    - Deve-se criar duas funções para realizar a operação de ordenação por intercalação: MergeSortRecursivo() e IntercalaComSentinela()
    - Deve-se imprimir cada etapa da ordenação na Console



Trabalhos para Casa (4)

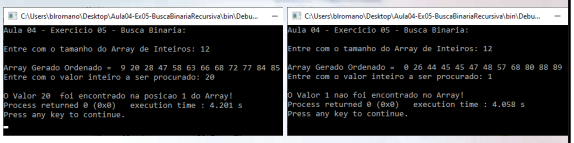
- **Exercício 04 – *MergeSort* Iterativo sem Sentinela**:
  - Implementar o MergeSort em C (Bottom-Up / Iterativo) e utilizar o algoritmo Intercala Sem Sentinela
    - Deve-se criar um algoritmo que gere um Array de números inteiros aleatórios de tamanho N, fornecido pelo usuário
    - Deve-se criar duas funções para realizar a operação de ordenação por intercalação: MergeSortIterativo() e IntercalaSemSentinela()
    - Deve-se imprimir cada etapa da ordenação na Console



Instituto Federal de São Paulo – IFSP São João da Boa Vista

Trabalhos para Casa (5)

- **Exercício 05 – Busca Binária com o Paradigma de Divisão e Conquista**:
  - O usuário deve fornecer um **array de tamanho N** e um **valor x a ser procurado** no array
    - Deve-se partir do pressuposto que o **array** de números inteiros encontra-se **ordenado**
  - Deve-se criar uma função BuscaBinariaRecursiva() que encontre o índice i do elemento x a ser encontrado no array, tal que A[i] = x, ou se o valor não foi encontrado
    - A função receberá o array, o início e o fim do (sub)arry e o item x a ser encontrado
    - A cada iteração deve-se chamar a função BuscaBinariaRecursiva() recursivamente reduzindo o espaço de busca dividindo o array pela metade, através das variáveis de início e fim do sub-array



  - Em um arquivo .doc, analisar o Algoritmo desenvolvido:
    - Identificar as etapas da Divisão e Conquista (Dividir, Conquistar e Combinar) no algoritmo
    - Realizar a Análise da Complexidade T(n): Número de Operações, Fórmula de Recorrência e Árvore (Similar ao *Mergesort*)
    - Entregar em um arquivo zipado: o .doc e o algoritmo da busca binária desenvolvida

37

PANC: Projeto e Análise de Algoritmos

Aula 04: Paradigma de Divisão e Conquista e Ordenação utilizando MergeSort (Lógica e Complexidade)

Breno Lisi Romano

Dúvidas???

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO Campus São João da Boa Vista

38