Algoritmos e Estrutura de Dados II
(ESTCMP011)
2do PERIODO 2018

Algoritmos de ordenação: Merger-sort.

Prof. Luis Cuevas Rodríguez, PhD
E-mail: lcuevasrodriguez@gmail.com / lrodriguez@uea.edu.br
Celular: 9298154648

Conteúdo

- Projeto de algoritmos
- Especificado o algoritmo de ordenação por:
  - Merger sort
- Analisar seu tempo de execução.

PROJETO DE ALGORITMO

Abordagem no Projeto de algoritmos

- Existem muitas maneiras de projetar algoritmos.
- A ordenação por inserção utiliza uma abordagem incremental.
- Abordagem de **dividir e conquistar**:
  - desmembram o problema em vários subproblemas que são semelhantes ao problema original, mas menores em tamanho,
  - resolvem os subproblemas **recursivamente** e depois combinam essas soluções com o objetivo de criar uma solução para o problema original

Ordenação por intercalação

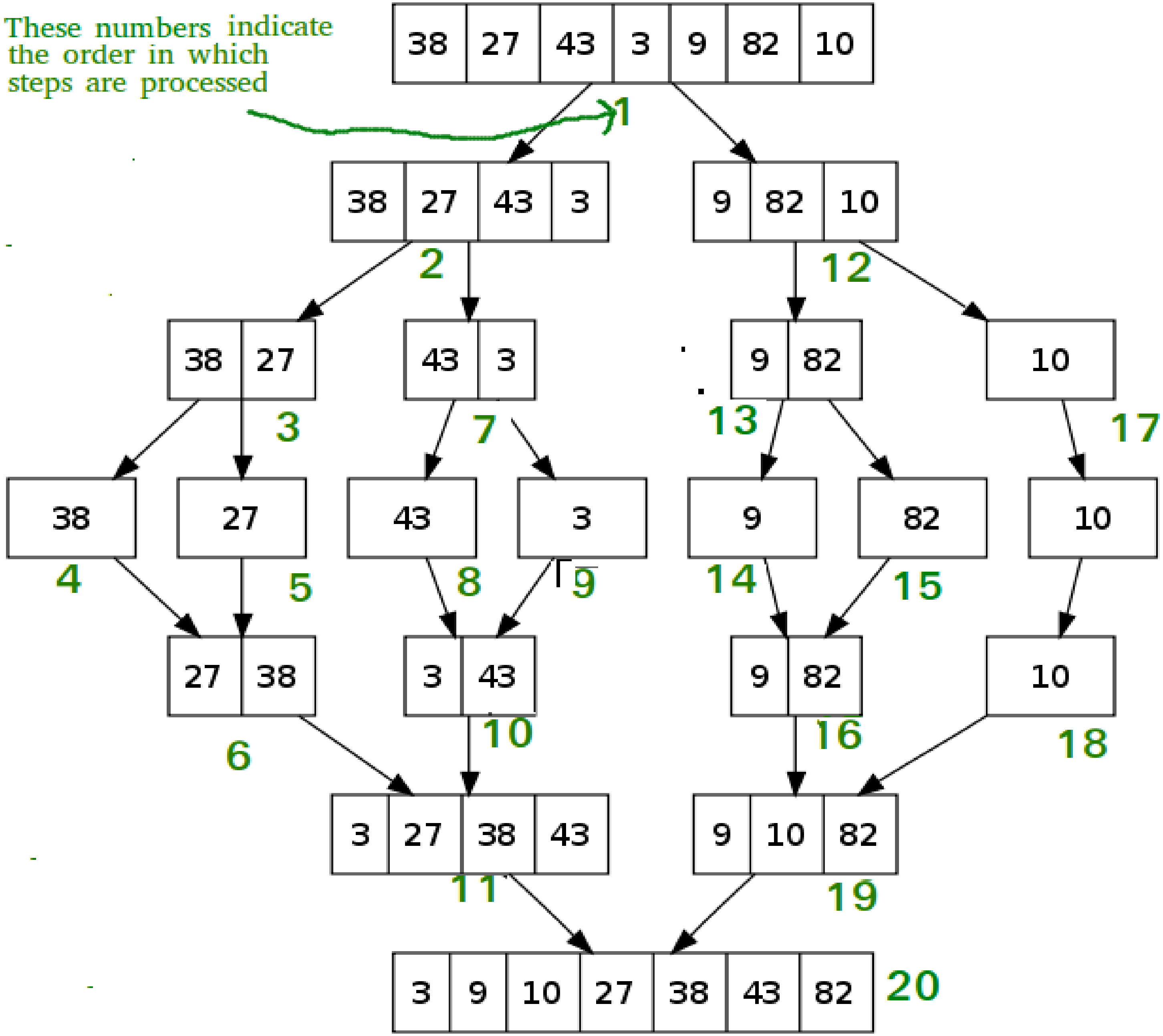
Merge-sort (ordenação por intercalação)

- **Dividir**: Divide a sequência de n elementos em duas partes, cada uma com (n / 2) elementos
- **Conquistar**: Ordena as duas partes usando o merge-sort ***recursivamente***
- **Combinar**: Faz a intercalação das duas sequências ordenadas

- A recursão para quando a sequência é de tamanho 1, pois não há nada a ser feito

Procedimento MERGE

- Procedimento MERGE leva o tempo ***O(n)***, onde n = r-p + 1 é o número de elementos que estão sendo intercalados
  - Vamos supor que temos duas pilhas de cartas com a face para cima sobre uma mesa.
  - Cada pilha está ordenada, com as cartas de menor valor em cima.
  - Juntar as duas pilhas (fazendo a intercalação) em uma única pilha de saída ordenada, que ficará com a face para baixo na mesa.
  - Passo básico: escolher a menor das duas cartas superiores nas duas pilhas viradas para cima, removê-la de sua pilha (o que irá expor uma nova carta superior) e colocar essa carta com a face voltada para baixo sobre a pilha de saída.
  - Repetimos esse passo até uma pilha de entrada se esvaziar e, nesse momento, simplesmente pegamos a pilha de entrada restante e a colocamos virada para baixo sobre a pilha de saída.



Prof. Luis Cue

https://www.geeksforgeeks.org/merge-sort/

UNIVERSIDADE DO ESTADO DO AMAZONAS

Ordenação por intercalação

```
MERGE(A, P, q, r)
n1 ← q − p + 1
n2 ← r − q
criar arranjos L[1..n1 + 1] e R[1..n2 + 1]
for i ← 1 to n1
        do L[i] ← A[p + i − 1]
for j ← 1 to n2
        do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
 i ← 1
 j ← 1
 for k ← p to r
        do if L[i] ≤ R[j]
              then A[k] ← L[i]
                   i ← i + 1
              else A[k] ← R[j]
                   j ← j + 1
```

- MERGE(A,p, q, r), onde:
  - *A* é um arranjo e
  - *p, q* e *r* são índices de enumeração dos elementos do arranjo, tais que $p \leq q < r$
- O procedimento pressupõe que os subvetores *A[p .. q]* e *A[q + 1 .. r]* estão em seqüência ordenada.
- Ele são intercalados para formar um único vetor ordenado que substitui o vetor atual *A[p . . r]*.

UEA UNIVERSIDADE DO ESTADO DO AMAZONAS

Ordenação por intercalação

- Usar o procedimento *MERGE* como uma sub-rotina no algoritmo de ordenação por intercalação.
- O procedimento ***MERGE-SORT(A, p, r)*** ordena os elementos do sub vetor ***A[p .. r]***.
  - Se p ≥ r, o sub vetor tem no máximo um elemento e, portanto, já está ordenado.
- Caso contrário, a etapa de divisão simplesmente calcula um índice ***q*** que particiona **A[p ..r]** em dois sub vetor:
  - **A[p..q]**, contendo n/2 elementos, e
  - **A[q + 1..r]**, contendo n/2 elementos.

```
MERGE-SORT(A, p, r)
if p < r
    then q ← ⌊(p + r)/2⌋
         MERGE-SORT(A, p, q)
         MERGE-SORT(A, q + 1, r)
         MERGE(A, p, q, r)
```

Exercícios

- Ilustre a operação de MERGER-SORT nos vetores
  - A = (31, 41, 59, 26, 41, 58).
  - B = (5, 1, 9, 6, 8).

Implementação

```
void merge(int arr[], int inicio, int medio, int fim)
{
    int i, j, k;
    int n1 = medio - inicio + 1;
    int n2 =  fim - medio;

    int Esquerda[n1], Dereita[n2];
    for (i = 0; i < n1; i++)
        Esquerda[i] = arr[inicio + i];
     for (j = 0; j < n2; j++)
        Dereita[j] = arr[medio + 1+ j];
    i = 0;
    j = 0;
    k = inicio;

    while (i < n1 && j < n2)
    {
        if (Esquerda[i] <= Dereita[j])
        {
            arr[k] = Esquerda[i];
            i++;
        }
        else
        {
            arr[k] = Dereita[j];
            j++;
        }
        k++;
    }
    while (i < n1)
    {
        arr[k] = Esquerda[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        arr[k] = Dereita[j];
        j++;
        k++;
    }
}
```

Implementação

```c
void mergeSort(int arr[], int inicio, int fim)
{
    if (inicio < fim)
    {
        int medio = inicio+(fim-inicio)/2;
        mergeSort(arr, inicio, medio);
        mergeSort(arr, medio+1, fim);
        merge(arr, inicio, medio, fim);
    }
}
```

```c
int main()
{
    int lista[TAM]={38,27,43,3,9,82,10};
    imprime_vetor(lista);
    mergeSort(lista,0,TAM);
    imprime_vetor(lista);
    return 0;
}
```

Análise de algoritmos de dividir e conquistar

MERGE(A, P, q, r)	
$n_1 \leftarrow q - p + 1$	1
$n_2 \leftarrow r - q$	1
criar arranjos $L[1..n_1 + 1]$ e $R[1..n_2 + 1]$	1
for $i \leftarrow 1$ **to** n_1	n/2
do $L[i] \leftarrow A[p + i - 1]$	
for $j \leftarrow 1$ **to** n_2	n/2
do $R[j] \leftarrow A[q + j]$	
$L[n_1 + 1] \leftarrow \infty$	1
$R[n_2 + 1] \leftarrow \infty$	1
$i \leftarrow 1$	1
$j \leftarrow 1$	1
for $k \leftarrow p$ **to** r	r = n
do if $L[i] \le R[j]$	n-1
then $A[k] \leftarrow L[i]$	n-1
$i \leftarrow i + 1$	
else $A[k] \leftarrow R[j]$	
$j \leftarrow j + 1$	

T(n)=2(n/2)+2 = n

T(n)= n

T(n)= 2n = $\Theta(n)$

UNIVERSIDADE DO ESTADO DO AMAZONAS

Análise de algoritmos de dividir e conquistar

```
MERGE-SORT(A, p, r)
if p < r                                   1
    then q ← ⌊(p + r)/2⌋                   1
        MERGE-SORT(A, p, q)                T(n/2)
        MERGE-SORT(A, q + 1, r)            T(n/2)
        MERGE(A, p, q, r)                  n
```

$$T(n) = \begin{cases} 1 & , se\ n = 1 \\ 2T(n/2) + n & , \mathrm{se\ n} > 1 \end{cases}$$

Prof. Luis Cuevas Rodríguez, PhD

UEA UNIVERSIDADE DO ESTADO DO AMAZONAS

Análise de algoritmos de dividir e conquistar

- *Para facilitar a análise, suponha que n=2k*
- $T(n) = 2T(n/2) + n$

$$= 2[2T(n/4) + n/2] + n = 4T(n/4) + 2n$$

$$= 4[2T(n/8) + n/4] + 2n = 8T(n/8) + 3n$$

$$= 8[2T(n/16) + n/8] + 3n = 16T(n/16) + 4n$$

- $T(n) = 2^a T(n/2^a) + an$

Análise de algoritmos de dividir e conquistar

- $Quando\ a = k\ temos$
- $T(n) = 2^a T\left(n/2^a\right) + an$

$$= 2^k T\left(2^k / 2^k\right) + kn$$

$$= 2^k T\left(1\right) + kn = \mathbf{2^k + kn}$$

como n - 2k

então k = $\log_2$ n

- $T(n) = \mathbf{2^{log2n} + n\ log_2\ n}$

$$= \boldsymbol{n + n\ log_2\ n}$$

$$= \Theta(\text{n lg n})$$

Bibliografía

- CORMEN, T.H., LEISERSON, C.E., RIVEST, R.L., STEIN, C. Algoritmos: Teoria e Prática. Tradução da 3a. edição. Rio de Janeiro: Elsevier, 2012.
- SZWARCFITER, J, L., MARKENZON, L. Estruturas de Dados e seus Algoritmos. 2a edição. Rio de Janeiro: LTC, 2010.
- ZIVIANI, N. Projeto de Algoritmos com Implementação em Pascal e C. 3a edição. São Paulo: Cengage Learning, 2010.
- STROUSTRUP, B. Programming: Principles and Practice Using C++. 2nd Edition. Addison-Wesley Professional. 2014.
- B. Stroustrup. The C++ Programming Language, 4th Edition, 2013.
- Feofiloff, P. Algoritmos em Linguagem C. Elsevier. 2009.
- AHO, A. V. et al. Data Structure and Algorithms. Readings, Addison-Wesley.
- WIRTH, N. Algoritmos e Estruturas de Dados. Rio de Janeiro: Ed. Prentice Hall do Brasil.
- KNUTH, D. E. The Art of Computer Programming. Vol. 1, Addison-Wesley, Reading, Mass.