Universidade Cruzeiro do Sul

Estruturas de Dados

Conteúdo **(revisão)**

- Recursividade (recursão).
- Métodos recursivos.
- Método de ordenação Quick Sort.
- Método de ordenação Merge Sort.

Elaboração

Prof. Manuel F. Paradela Ledón

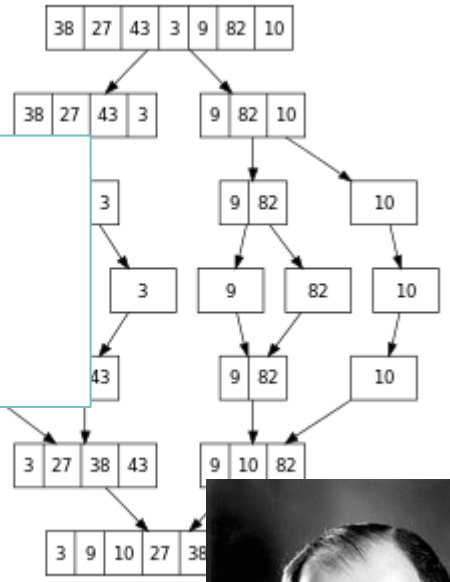
Divisão e conquista

A solução de um problema (algoritmo etc.) poderá utilizar o paradigma da **divisão e conquista**. Esse paradigma (ou estratégia para a resolução de problemas) consiste no seguinte:

- o problema é dividido em dois ou mais problemas menores;
- cada parte menor (ou subproblema) será resolvida utilizando normalmente o mesmo método de solução sendo utilizado;
- as soluções das instâncias menores são combinadas para produzir uma solução do problema original.

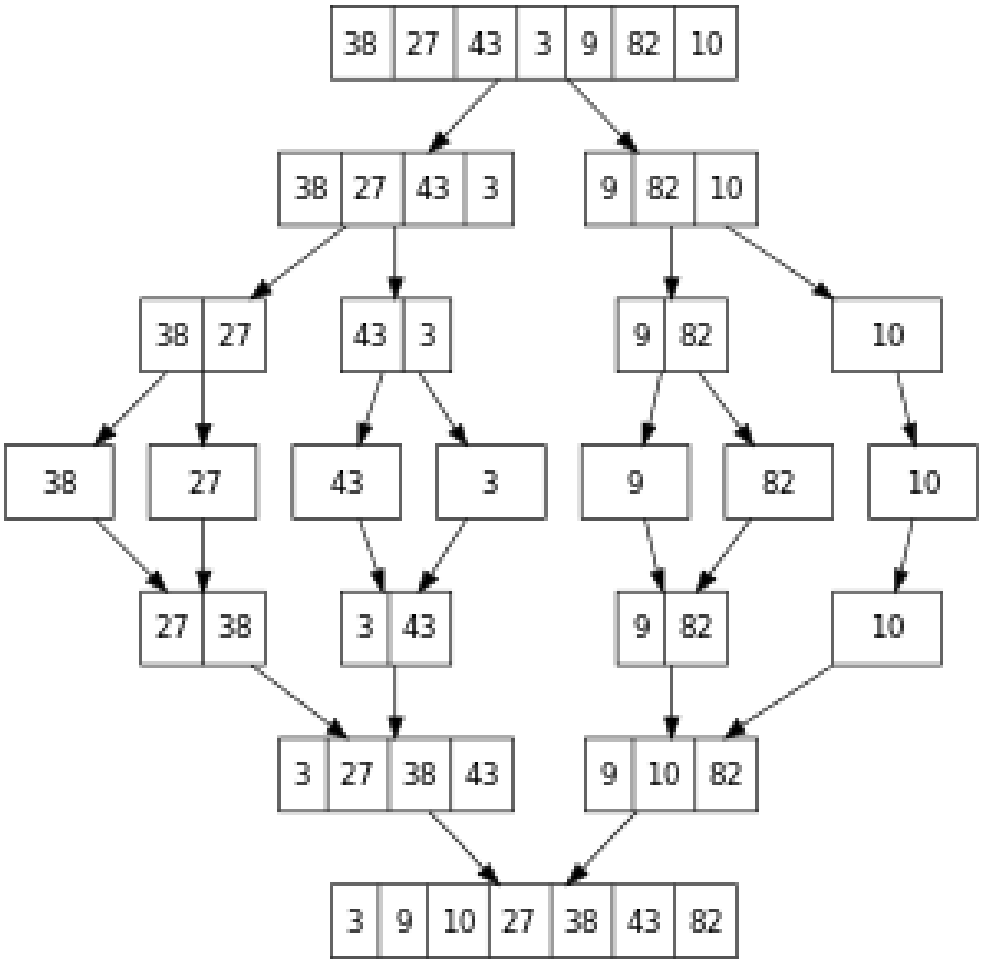
Os métodos de ordenação **Quick Sort** e **Merge Sort** são "naturalmente" **recursivos** e utilizam a abordagem da **divisão e conquista** para resolver o problema.

Merge Sort



John von Neumann: método proposto em 1945
Matemático, cientista da computação húngaro, 1903-1957

Método de ordenação Merge Sort



Fonte da figura:
https://commons.wikimedia.org/wiki/File:Merge_sort_algorithm_diagram.svg

O Merge Sort (merge significa: fundir, misturar) se baseia no método de divisão e conquista (dividir para conquistar, divide-and-conquer, análise e síntese).

Podemos considerar três etapas ou fases:

1. **Divisão**: se o tamanho da entrada for menor que um certo limite (normalmente consideramos um ou dois elementos), resolvemos diretamente e retornamos a solução obtida. Em qualquer outro caso, os dados de entrada são divididos, normalmente em uma ou duas partes.

2. **Recursão**: Observemos que cada parte obtida no passo anterior será resolvida, utilizando o mesmo método, em forma recursiva.

3- **Conquista**: as soluções dos subproblemas são unidas em uma única solução, também em etapas para cada tamanho de partição, fundindo ordenadamente até chegar no vetor final ordenado.

Preparando a ordenação

Universidade Cruzeiro do Sul

```
package ordenacaomergesort;
// Programação: Ledón (implementação baseada no algoritmo de Mark Allen Weiss)
public class OrdenacaoMergeSort {
    public static void main(String[] args) {
        new OrdenacaoMergeSort();
    }

    public OrdenacaoMergeSort() {
        double vet[] = {71.2, 0.3, 6.3, -1.2, 5.4, 0.5, 0.2, 91.5, 33.3, 0.9};  //vetor que queremos ordenar
        double tempVet [] = new double[vet.length];  // vetor auxiliar
        System.out.println("Vetor desordenado:");
        visualizarVetor(vet);
        mergeSort(vet, tempVet, 0, vet.length-1);  // ordenamos o vetor vet completo
        System.out.println("Vetor ordenado:");
        visualizarVetor(vet);
    }
```

Solicitando a ordenação

mergeSort: dividir em trechos mais pequenos

Universidade Cruzeiro do Sul

```
public void mergeSort( double vet[], double tempVet[], int esq, int dir ){
    if (esq < dir) {
        // caso contrário (se o trecho do vetor tiver mais de um elemento) abandonaremos
        int centro = (esq + dir)/2;
        mergeSort(vet, tempVet, esq, dir:centro);   //ordenar (dividir) o trecho esquerdo
        mergeSort(vet, tempVet, centro+1, dir);     //ordenar (dividir) o trecho direito
        merge(vet, tempVet, esq, centro+1, dir);    //misturar (fusionar) dois sub-trechos
    }
}
```

- **Divisão.** As duas chamadas recursivas ao método **mergeSort** irão dividindo o vetor em dois trechos cada vez mais pequenos.
- **Conquista.** O método **merge** chamado aqui (ver a lógica no próximo slide) mistura (funde), os dois trechos analisados, que são os trechos entre [esq, centro+1] e [centro+1, dir].

merge: fusionar, fundir

Universidade Cruzeiro do Sul

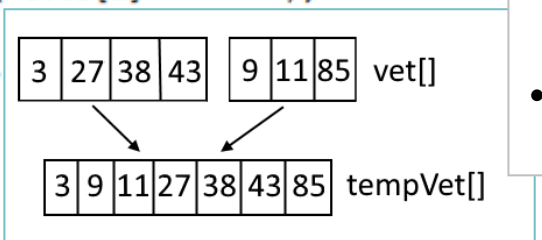
```
public void merge( double vet [], double tempVet [] , int esq, int centro, int dir ) {
    int fimTrechoEsquerdo = centro - 1;
    int i = esq;
    int qtdeElementos = dir - esq + 1;
    //------------ Mistura, fusão inicial de elementos:
    while( esq <= fimTrechoEsquerdo && centro <= dir )
        if( vet[ esq ] <= vet[ centro ] )
            tempVet[ i++ ] = vet[ esq++ ];
        else
            tempVet[ i++ ] = vet[centro++];
    //------------ Ciclo para copiar o resto da metade esquerda:
    while( esq <= fimTrechoEsquerdo )
        tempVet[ i++ ] = vet[ esq++ ];
    //------------ Ciclo para copiar o resto da metade direita:
    while( centro <= dir )
        tempVet[ i++ ] = vet[centro++];
    //-------- Finalmente, copiamos o trecho do vetor temporário para o vetor original:
    for( i = 0; i < qtdeElementos; i++, dir-- )
        vet[dir] = tempVet[dir];
}

public void visualizarVetor(double vetor[]) {
    for (int i = 0; i < vetor.length; i++) {
        System.out.print(vetor[i] + "   ");
    }
    System.out.println();
}
}
```

- adiciona no vetor temporário **tempVet** o menor dos dois elementos de **vet**, seja da parte esquerda ou da direita;
- o índice do item copiado (esq ou centro) será incrementado e também o índice i de **tempVet**

com estes dois ciclos copiamos de **vet** para **tempVet** os itens remanescentes que poderiam ter ficado na parte esquerda ou na parte direita do trecho analisado do vetor **vet**

- este último ciclo copia os elementos do vetor auxiliar **tempVet** para o vetor original **vet** (parâmetro por referência, de entrada/saída => resultado retornado);
- é um ciclo que repete **qtdeElementos** vezes;
- observe que o índice utilizado para copiar é **dir** (extremo direito do trecho que este método está misturando [merge], que é o único índice que não foi alterado nos ciclos anteriores);
- **dir** será decrementado com o comando **dir--** até chegar no início do trecho analisado.



Analisando os métodos de ordenação estudados

Existem diferentes fatores a serem considerados quando comparamos os diferentes algoritmos de ordenação: a quantidade de dados a serem ordenados, a ordenação prévia dos dados, a eficiência quanto a velocidade, eficiência quanto à memória auxiliar utilizada, a complexidade da implementação e os ajustes específicos que melhoram cada método.

Da lista a seguir, na média, os dois algoritmos mais eficientes quanto a desempenho são o Quick Sort e o Merge Sort (porque n*log(n) é melhor que n^2). O Merge Sort, possui uma limitação importante quanto a utilização de memória adicional (vetor temporário utilizado). Ambos algoritmos são recursivos e utilizam memória adicional por causa da recursão (da pilha de execução, stack), mas o Quick Sort utiliza menos memória auxiliar.

Apesar de ser mais ineficientes, Bubble, Insertion e Selection se caracterizam pela simplicidade e pouco requerimento de memória adicional.

Algoritmo	**Complexidade (tempo)**			**Complexidade (espaço)**
	melhor caso	**médio**	**pior caso**	**---**
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$	O(log n)
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)

Bibliografia (oficial) para a disciplina

BIBLIOGRAFIA BÁSICA	BIBLIOGRAFIA COMPLEMENTAR
CORMEN, T. H.; et al. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012. GOODRICH, M. T.; TAMASSIA, R. Estrutura de dados e algoritmos em java. 5. ed. Porto Alegre: Bookman, 2013. (livro físico e e-book) CURY, T. E., BARRETO, J. S., SARAIVA, M. O., et al. Estrutura de Dados (1. ed.) ISBN 9788595024328, Porto Alegre: SAGAH, 2018 (e-book)	ASCENCIO, A. F. G.; ARAÚJO, G. S. Estruturas de Dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++. São Paulo: Pearson Prentice Hall, 2010. (eBook) PUGA, S.; RISSETTI, G. Lógica de programação e estruturas de dados, com aplicações em Java. 3. ed. São Paulo: Pearson Education do Brasil, 2016. (eBook) DEITEL, P.;DEITEL, H. Java como programar. 10. ed. São Paulo: Pearson Education do Brasil, 2017. (eBook) BARNES, D. J.; KOLLING, M. Programação Orientada a Objetos com Java: uma introdução prática usando o Blue J. São Paulo: Pearson Prentice Hall, 2004. (eBook) BORIN, V. POZZOBON. Estrutura de Dados. ISBN: 9786557451595, Edição: 1ª . Curitiba: Contentus, 2020 (e-book)

Referências adicionais

- Quick Sort | **GeeksforGeeks** em: https://www.youtube.com/watch?v=PgBzjlCcFvc&t=89s
- AURÉLIO. Aurélio Buarque de Holanda Ferreira. **Mini Aurélio. Dicionário da Língua Portuguesa**. Curitiba: Positivo, 2004.
- HOUAISS. **Mini Houaiss. Dicionário da Língua Portuguesa**. Rio de Janeiro: Objetiva, 2009.
- PADOVANI, U.; CASTAGNOLA, L. **História da Filosofia**. São Paulo: Melhoramentos, 1972.
- WEISS, M. A. **Data Structures and Algorithm Analysis**. 2nd Edition. The Benjamin/Cummings Publishing Company: California, 1994.