Aula 08

Engenharia da Computação – 3ª série

Recorrências

(E1, E2)

2024

Recorrências

Pergunta

- Como analisar Algoritmos Recursivos?

Recorrências

Resposta

- **Algoritmos Recursivos**, em geral, são muito elegantes;
- Porém, a análise de Algoritmos Recursivos é mais trabalhosa, uma vez que se emprega uma **Equação de Recorrência**;
- Para se analisar o consumo de tempo de um algoritmo recursivo é necessário resolver-se uma **Equação de Recorrência.**

Recorrências

Pergunta

- O que é Recorrência?

Recorrências

Resposta

- **Recorrência** é uma expressão que dá o valor de uma função em termos dos valores “anteriores” da mesma função;

Recorrências

Pergunta

- O que é uma Equação de Recorrência?

Recorrências

Resposta

- Uma **Equação de Recorrência** define sentenças matemáticas que o algoritmo deve atender em tempo de execução, por exemplo:

$$T(n) = \begin{cases} 3 & \text{se } \mathbf{n = 1} \\ T(n-1) + 7 & \text{para } \mathbf{n > 1} \end{cases}$$

Recorrências

Resposta

- A **recorrência** consiste em uma equação que corresponde à quantidade de operações executadas pelo algoritmo, em tempo de execução;

- Na **análise recursiva** do algoritmo, deve-se contar cada comparação, referência à vetores (*arrays*), chamadas recursivas e retornos de funções como operações **básicas** (**Modelo de Knuth**).

Recorrências

Exemplo

- O exemplo clássico de **Recorrência**, provavelmente o mais famoso, é a fórmula de **Fibonacci**:

$$F(n) = \begin{cases} 0, & \text{se } n=0 \\ 1, & \text{se } n=1 \\ F(n-1) + F(n-2) & \text{se } n>1 \end{cases}$$

Recorrências

Pergunta

- Como derivar uma Fórmula Explícita a partir de uma Fórmula de Recorrência?

Recorrências

<u>Resposta</u>

- Embora uma **fórmula de recorrência** nos dê uma boa ideia de como um determinado termo está relacionado com o anterior, ela não nos ajuda a encontrar – de forma direta – um determinado termo sem passar pelos termos relacionados na recorrência;

Recorrências

Resposta

- Essa fórmula explícita fornece uma melhor visão da ordem de complexidade do algoritmo. Por exemplo, **prova-se** na série de **Fibonacci** que:

$$F(n) = \begin{cases} 0, & \text{se } n=0 \\ 1, & \text{se } n=1 \\ F(n-1) + F(n-2) & \text{se } n > 1 \end{cases}$$

Fórmula Fechada →

$$\boxed{F_n = \frac{1}{\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n}$$

Ordem de Complexidade Exponencial

Recorrências

Resposta

- Outro exemplo clássico de **Recorrência** é o cálculo do **Fatorial** de um número natural ***n***:

$$\text{fatorial(n)} = \begin{cases} 1 & \text{se } n = 0 \\ \text{n. fatorial (n-1)} & \text{se } n \geq 1 \end{cases}$$

Recorrência

$$\begin{aligned}
\text{Fatorial(n)} &= \text{n . Fatorial (n-1)} \\
&= \text{n . (n-1) . Fatorial(n-2)} \\
&= \text{n . (n-1) . (n-2) . Fatorial (n-3)} \\
&= \ldots \\
&= \text{n . (n-1).(n-2) . . . Fatorial(0)} \\
&= \text{n . (n-1).(n-2) . . . 1 = n!}
\end{aligned}$$

Fórmula Fechada

Recorrências

Pergunta

- O que significa Resolver uma Recorrência?

Recorrências

Resposta

- Resolver uma **Recorrência** significa encontrar uma **fórmula fechada**, ou **explícita**, que forneça o valor da função diretamente em termos de seu argumento;

- Embora uma fórmula de recorrência dê uma boa ideia de como um determinado termo está relacionado com o anterior, ela não ajuda a encontrar – de forma direta – um determinado termo sem passar pelos termos relacionados na recorrência;

- Uma das formas mais simples de se resolver a recorrência é pelo método da **Substituição**.

Recorrências

Pergunta

- O que é o Método da Substituição?

Recorrências

Resposta

- A solução de uma **equação** de **recorrência** por **substituição** requer uma prévia instância da fórmula para ser substituída na equação dada;

- O processo é continuado até que seja capaz de alcançar a condição inicial.

Recorrências

Exemplo

- Através de um Algoritmo Recursivo, dado um vetor (*array*) ordenado de números, efetuar a busca de um determinado elemento nesse vetor;
- Em outras palavras, dado um vetor ***A*** ordenado, de ***n*** elementos, a função ***f*** correspondente à busca deve retornar o índice ***i*** de um elemento ***x*** de ***A***, tal que ***A[i] = x***;
- A função ***f*** deve retornar ***-1*** caso o elemento pesquisado não esteja no vetor ***A***:

10	12	20	22	35	37	39	40	56	70	71	75

Recorrências

Exemplo

- Pode-se tirar proveito do fato que os dados estão sequencialmente **ordenados**.

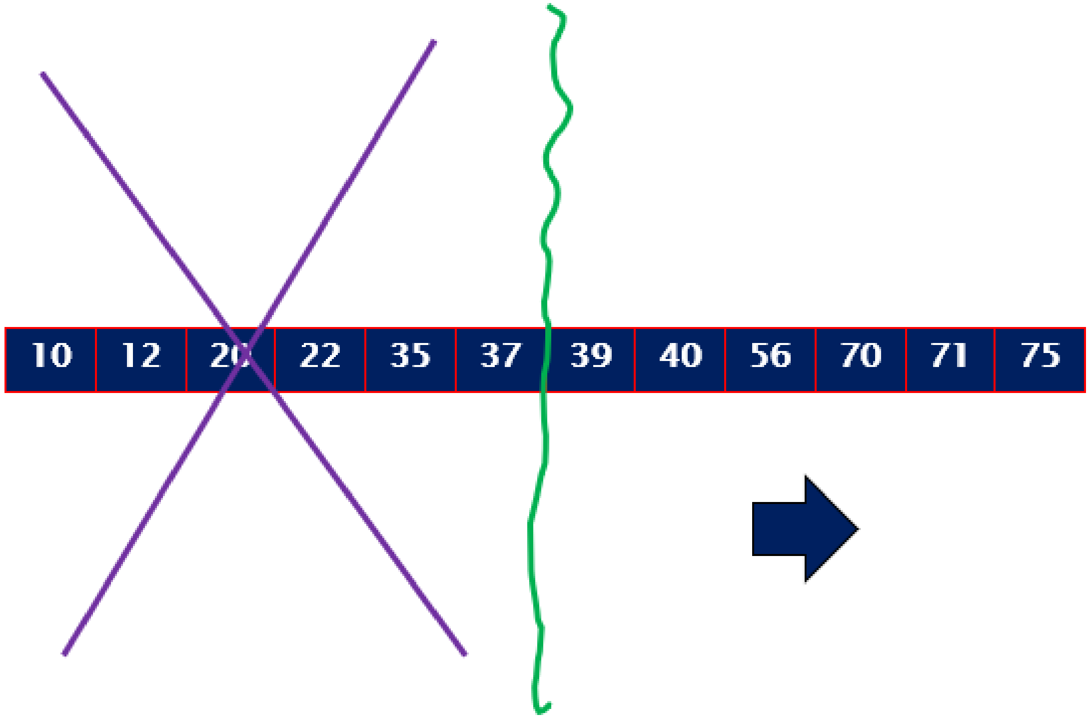
- Pode-se fazer uma **comparação** do dado a ser pesquisado com o elemento **central** do vetor;

10	12	20	22	35	37	39	40	56	70	71	75

Recorrências

Exemplo

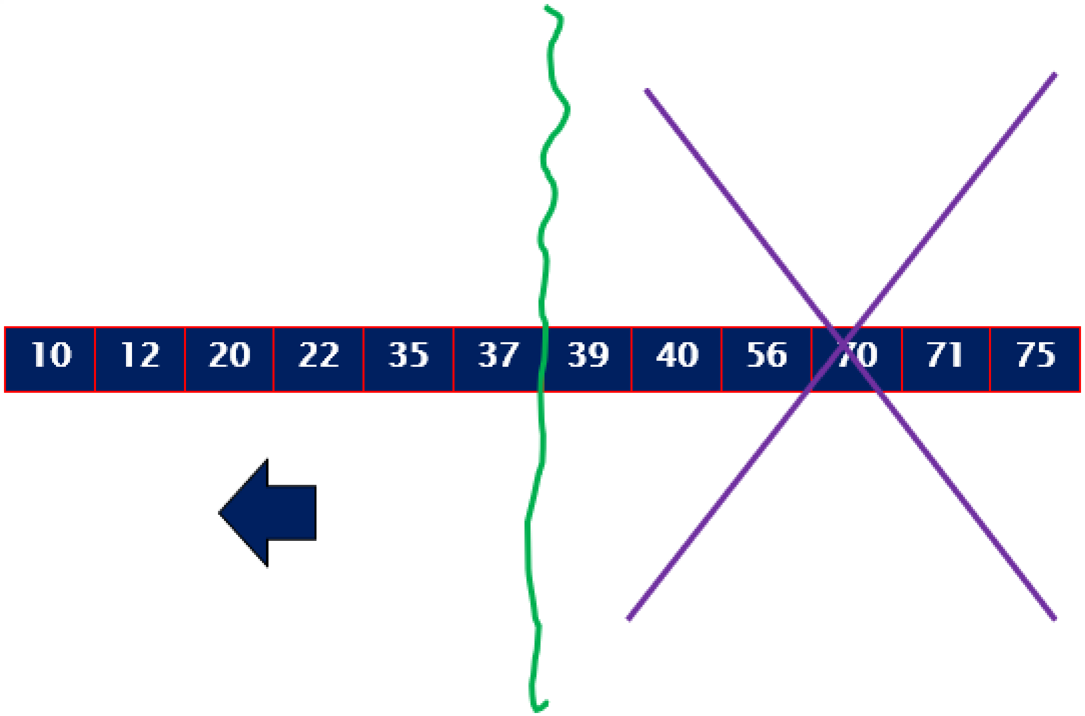
- Se o dado a ser pesquisado for **maior**, a busca caminha para a **direita**, desprezando-se os dados à **esquerda**.



Recorrências

Exemplo

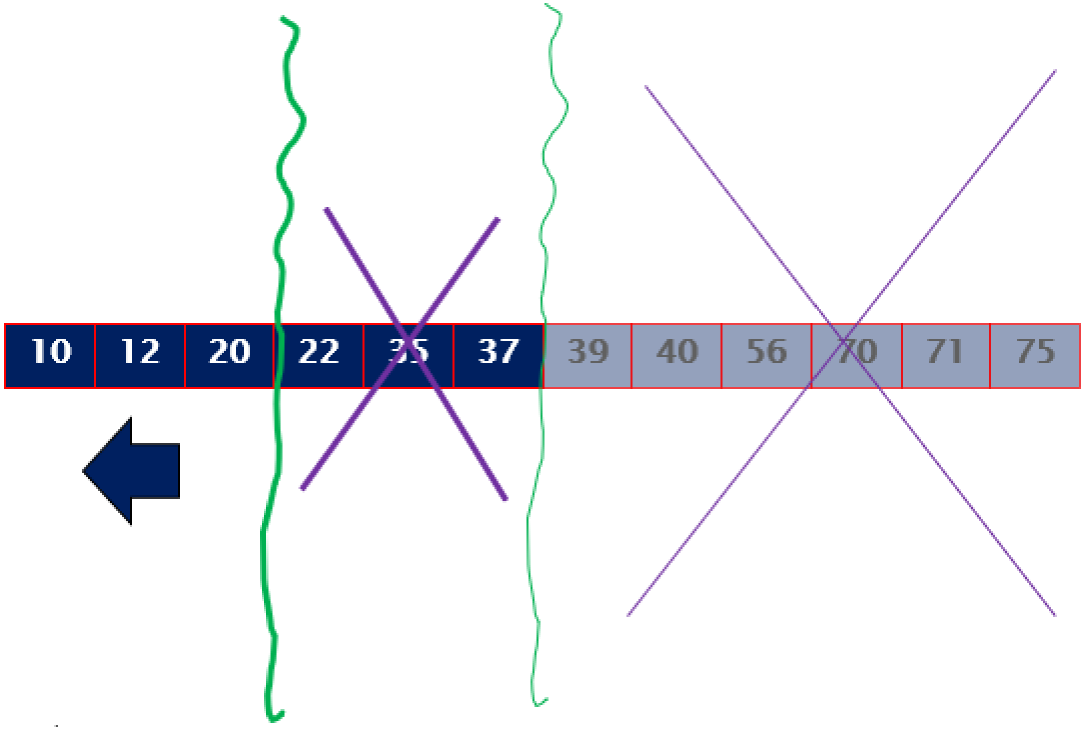
- Se o dado a ser pesquisado for **menor**, a busca caminha para a **esquerda**, desprezando-se os dados à **direita**.



Recorrências

Exemplo

- Esse processo de **divisão e conquista** prossegue até a divisão **não** ser mais possível, caso base da recursão.



Recorrências

Exemplo


```java
package br.uscs;

public class BinarySearch {
    public static int nComparacoes = 0;
    public static int indice;
    public static int[] A = {10,12,20,22,35,37,39,40,56,70,71,75};

    public static void main(String[] args) {

        int n = 71;  //item a ser pesquisado

        System.out.println(" -----   Busca Sequencial -----");

        seqSearch(n);

        if (indice != -1) {
            System.out.println("Valor encontrado na posição: " +
                        indice);
            System.out.println("Total de Comparações: " +
                        nComparacoes);
        }
        else {
            System.out.println("Valor NÃO encontrado na posição: " +
                    indice);
            System.out.println("Total de Comparações: " +
                    nComparacoes);
        }

        nComparacoes = 0;
```

Recorrências

Exemplo

```
    System.out.println("\n\n---- Busca Binária ----");
    binSearch(n, 0, A.length-1); //algoritmo de busca Binária

    if (indice != -1) {
        System.out.println("Valor encontrado na posição: " + indice);
        System.out.println("Total de Comparações: " + nComparacoes);
    }
    else {
        System.out.println("Valor NÃO encontrado no array");
        System.out.println("Total de Comparações: " + nComparacoes);
    }

}
```

Recorrências

Exemplo


```java
//----------  Busca Sequencial ------------------
public static void seqSearch(int n) {
    for(int i = 0; i < A.length; i++) {
        if (A[i] == n) {
            nComparacoes++;
            indice = i ;
            break;
        }
        nComparacoes++;
    }

    indice = -1;
}
```

Recorrências

Exemplo

```
// ------------ Busca Binária ------------------

public static void binSearch(int item, int begin, int end) {
    int metade = (begin + end)/2;

    if (begin > end)  { //ponto de parada => caso base
        indice = -1;
        nComparacoes++;
        return;
    }

    if (A[metade] == item) {
        indice = metade;
        nComparacoes++;
        return;
    }

    if (A[metade] < item)  {
        nComparacoes++;
        binSearch(item, metade+1, end);

    }
    else {
        nComparacoes++;
        binSearch(item,begin, metade);
    }
}

}
```

Recorrências

Exemplo

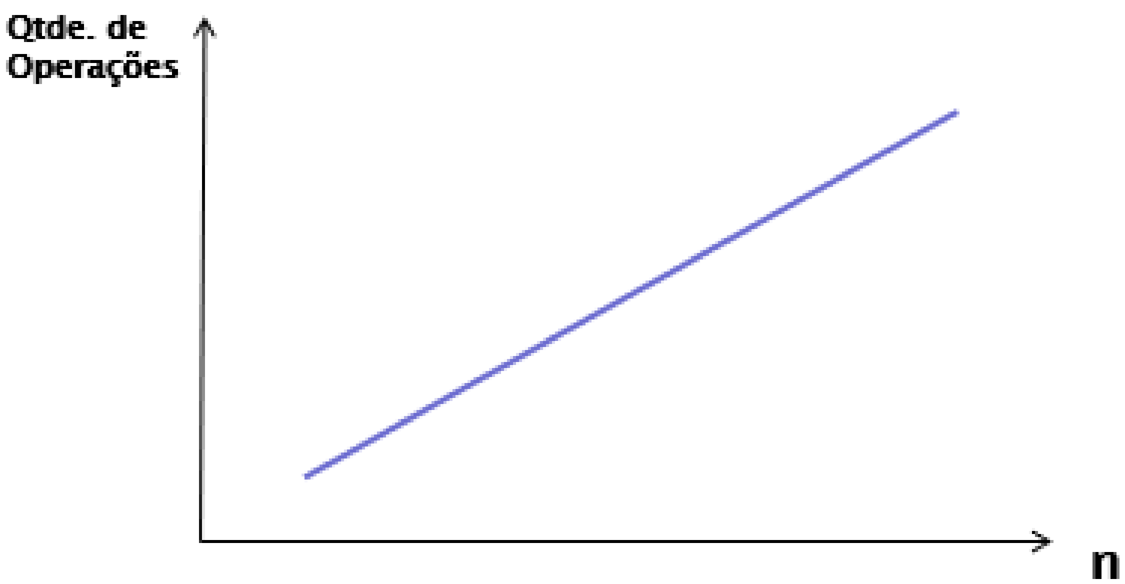

```
 -----   Busca Sequencial -----
Valor encontrado na posição: 10
Total de Comparações: 11
 -----   Busca Binária  -----
Valor encontrado na posição: 10
Total de Comparações: 4
```

ECM306 – Tópicos Avançados em Estrutura de Dados

Recorrências

Exemplo

- Facilmente, pode-se concluir que a Busca Sequencial, **NO PIOR CASO**, é ***O(n)***.



Recorrências

Pergunta

- Qual a ordem de Complexidade do Algoritmo Recursivo ***binSearch()***?

Recorrências

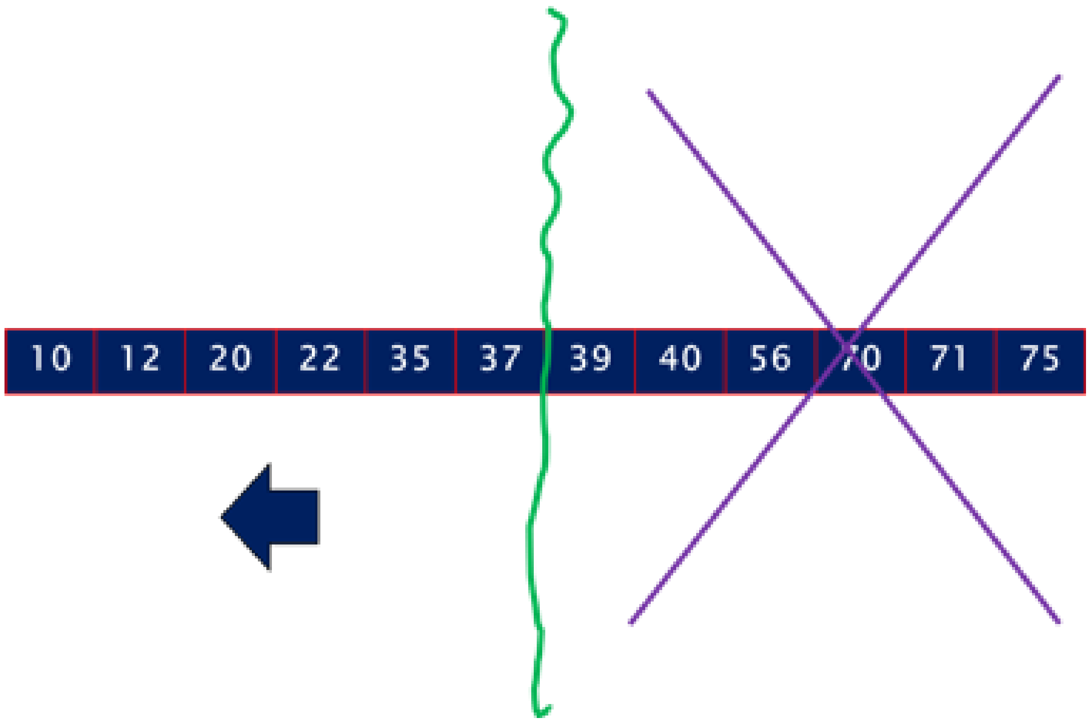
Pergunta

- Como analisar Algoritmos Recursivos?

Recorrências

Resposta

- Em cada iteração, o algoritmo **DESCARTA metade** do vetor:



Recorrências

Resposta

- A partir da segunda iteração, o algoritmo trabalha com **metade** do vetor;

- Ou seja, a partir da segunda iteração, o algoritmo é exatamente o mesmo, porém trabalhando com **metade** dos valores do vetor;

- Denotando-se por ***T(n)*** o número máximo de iterações realizadas pela busca binária sobre um **vetor** de ***n*** elementos;

Recorrências

Resposta

- A primeira iteração é com o vetor inteiro, mas a partir da segunda o algoritmo trabalha com metade do vetor, e assim, sucessivamente.

$$\text{Recorrência} \begin{cases} T(n) = 1 + T(n/2) & ; n > 1 \\ T(n) = 1 & ; n = 1 \end{cases}$$

Recorrências

Resposta

- ***n*** sempre é inteiro, mas ***n/2*** pode não ser inteiro;
- Ou seja, a divisão nem sempre ocorre exatamente no meio do vetor;
- Reescrevendo-se a Equação de Recorrência:

$$\text{Recorrência} \begin{cases} T(n) = 1 + \lfloor T(n/2) \rfloor & ; n > 1 \\ T(n) = 1 & ; n = 1 \end{cases}$$

Recorrências

Resposta

- Para se obter a ordem de complexidade, é preciso resolver a Equação de Concorrência, determinando-se a fórmula fechada da equação;

$$\text{Recorrência} \begin{cases} T(n) = 1 + \lfloor T(n/2) \rfloor & ; n > 1 \\ T(n) = 1 & ; n = 1 \end{cases}$$

Recorrências

Resposta

- Para se obter uma solução, supõe-se inicialmente que ***n = 2^k***;
- Com isso, ***T(n/2)*** sempre será inteiro e, portanto: $\lfloor T[n/2) \rfloor = T(n/2)$, descartando-se nesse caso o piso;

$$\text{Recorrência} \begin{cases} T(n) = 1 + T(n/2) & ; n > 1 \text{ e } n = 2^k,\ k_{\text{inteiro}} > 0 \\ T(n) = 1 & ; n = 1 \end{cases}$$

Recorrências

Resposta

- Aplicando-se o método da Substituição:

$$
\begin{aligned}
T(n) &= 1 + (\,T(n/2)\,) \\
&= 1 + (1 + T(n/4)\,) \\
&= 1 + (1 + T(n/2^2)\,) \\
&= 1 + 1 + (1 + T(n/2^3) \\
&\;\dots \\
&= k + T(n/2^k)
\end{aligned}
$$

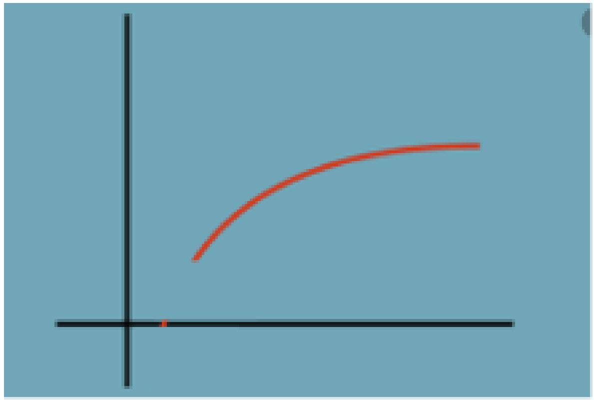
$$
\begin{cases}
T(n) = 1 + T(n/2) & ;\, n > 1 \text{ e } n = 2^k,\ k_{\text{ inteiro}} > 0 \\
T(n) = 1 & ;\, n = 1
\end{cases}
$$

Recorrências

Resposta

- Portanto:

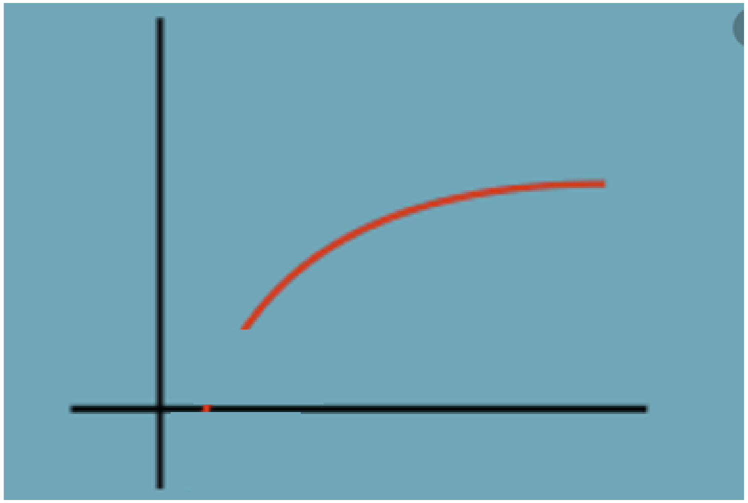
$$T(n) = \log_2 n + 1 \text{ , para } n = 2^k, \text{ k inteiro} > 0$$



$$\begin{cases} T(n) = 1 + T(n/2) & ; n > 1 \text{ e } n = 2^k, \text{ k}_{\text{ inteiro}} > 0 \\ T(n) = 1 & ; n = 1 \end{cases}$$

Conclusão

- Assim, no pior caso, a ordem de complexidade da Busca Binária é: ***O(log n)***.



n	Quantidade de Iterações
2	1
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10
2048	11
4096	12

Referências bibliográficas

- CORMEN, T.H. et al. Algoritmos: Teoria e Prática (Caps. 13). Campus. 2002.
- ZIVIANI, N. Projeto de algoritmos: com implementações em Pascal e C (Cap. 1). 2.ed. Thomson, 2004.
- FEOFILOFF, P. Minicurso de Análise de Algoritmos, 2010. Disponível em: http://www.ime.usp.br/~pf/livrinho-AA/
- DOWNEY, A.B. *Analysis of algorithms* (Cap. 2), Em: *Computational Modeling and Complexity Science.* Disponível em: http://www.greenteapress.com/compmod/html/book003.html
- ROSA, J.L. Notas de Aula de Introdução a Ciência de Computação II. Universidade de São Paulo. Disponível em: http://coteia.icmc.usp.br/mostra.php?ident=639

Referências bibliográficas

- GOODRICH, Michael T. et al: *Algorithm Design and* Applications. Wiley, 2015.
- LEVITIN, Anany. *Introduction to the Design and Analysis of Algorithms*. Pearson, 2012.
- SKIENA, Steven S. *The Algorithm Design Manual.* Springer, 2008.
- Série de Livros Didáticos. *Complexidade de Algoritmos.* UFRGS.
- BHASIN, Harsh. *Algorithms – Design and Analysis.* Oxford University Press, 2015.
- FREITAS, Aparecido V. de – 2022 – Estruturas de Dados: Notas de Aula.
- CALVETTI, Robson - 2015 – Estruturas de Dados: Notas de Aula.

Aula 08

FIM