PANC: Projeto e Análise de Algoritmos

Aula Complementar 01: Pesquisa Sequencial e Busca Binária

Breno Lisi Romano

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

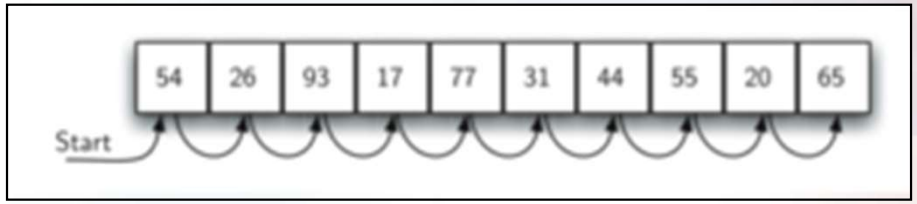
Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre

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

Pesquisa/Busca Sequencial (1)

- **Problema:**
  - O usuário deve fornecer um array ordenado de tamanho N e um valor x a ser procurado no array

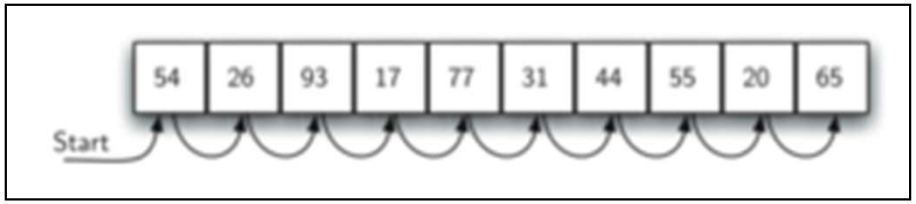
- **Lógica para Resolução:**
  - Deve-se fazer um loop, desde a primeira posição do array, comparando o valor armazenado com o valor buscado. Caso encontre o valor na posição pesquisada, deve-se parar a busca e imprimir a posição que encontrou o elemento. Caso não esteja nesta posição, incrementa-se o loop pesquisando na próxima posição, até o final do array. Necessita-se na Função:
    - Array
    - Valor Procurado (Alvo)
    - Tamanho do Array



Pesquisa/Busca Sequencial (2)

```
/* BuscaLinear */
void BuscaLinear(int arrayA[], int x, int n)
{
    int posicaoEncontrou = -1;
    for (int i=0; i < n; i++) {
        if(arrayA[i] == x) {
            posicaoEncontrou = i;
            break;
        }
    }

    if(posicaoEncontrou == -1)
        printf("\nValor nao encontrado no Array!");
    else
        printf("\nO Valor %d foi encontrado na posicao %d", v, posicaoEncontrou+1);
}
```

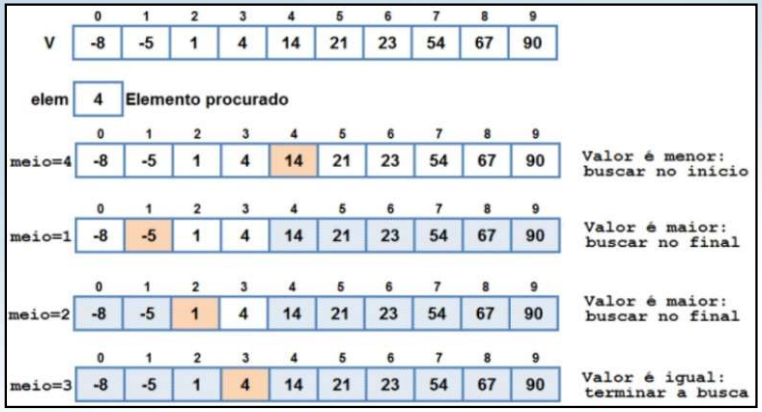


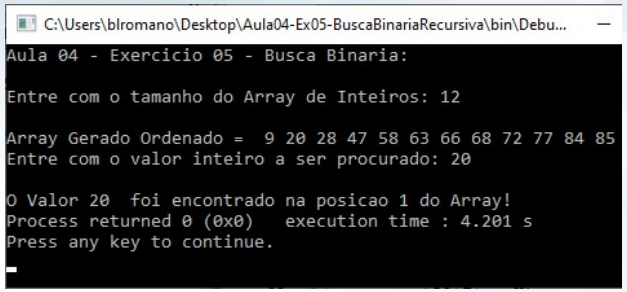
- **Análise de Complexidade – T(n):**
  - T(n) = O(n) → Pior Caso

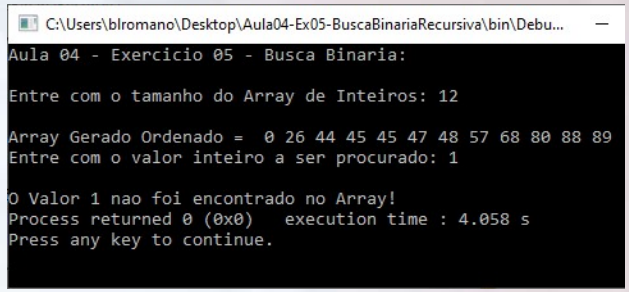
Listex 03 - Trabalhos para Casa

Exercício 05 – Busca Binária com o Paradigma de Divisão e Conquista:

- O usuário deve fornecer um **array ordenado 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

Busca Binária (1)

- **Problema:**
  - O usuário deve fornecer um array ordenado 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
- **Lógica para Resolução:**
  - A cada iteração deve-se chamar a função recursivamente reduzindo o espaço de busca dividindo o array pela metade, através das variáveis de início e fim do sub-array
  - Necessita-se na Função:
    - Array
    - Limite Inferior do Array após a Divisão
    - Limite Superior do Array após a Divisão
    - Valor Procurado (Alvo)

Busca Binária (2)

- **Exemplo:**
  - Simular para o Array = {-8, -5, 1, 4, 14, 21, 23, 54, 67, 90}
  - Valor Procurado: 4

Busca Binária (3)

- **Resolução Recursiva:**

```
/* BuscaBinaria() */
int BuscaBinaria (int x, int arrayA[], int inicio, int fim)
{
 int meio = (inicio + fim)/2;
 if (arrayA[meio] == x)
    return meio;
 if (inicio >= fim)
    return -1; // não encontrado
 else
     if (x > arrayA[meio])
        return BuscaBinaria(x, arrayA, meio+1, fim);
     else
        return BuscaBinaria(x, arrayA, inicio, meio-1);
}
```

- **Chamada do Método:**

```
//Definindo Valor a Ser Procurado
printf("\nEntre com o valor inteiro a ser procurado: ");
scanf("%d", &x);

//Busca Binaria
posicao = BuscaBinaria (x, arrayA, 0, n-1);

if(posicao == -1)
    printf("\nO Valor %d nao foi encontrado no Array!", x);
else
    printf("\nO Valor %d  foi encontrado na posicao %d do Array!", x, posicao+1);
```

Busca Binária (4)

```
int BuscaBinaria (int x, int Vet[], int inicio, int fim)
{
    int meio = (inicio + fim)/2;
    if (Vet[meio] == x)
        return meio;
    if (inicio >= fim)
        return -1; // não encontrado
    else
        if (Vet[meio] < x)
            return BuscaBinaria(x, Vet, meio+1, fim);
        else
            return BuscaBinaria(x, Vet, inicio, meio-1);
}
```

- **Análise de Complexidade – T(n):**
  - Seja **T(n)** a **complexidade de tempo** da busca binária:
    - **Caso Base**: T(1) = a
    - **Passo Recursivo**: T(n) = T(n/2) + b, para n > 1

- **Resolvendo a Recorrência**:
  - **Intuitivamente:** Quantas vezes se consegue dividir n por 2 até chegar em 1?
    - T(1) = a (Caso Base)
    - T(2) = T(1) + b → $T(2^1) = T(2^0) + b$
    - T(4) = T(2) + b → $T(2^2) = T(2^1) + b$
    - T(8) = T(4) + b → $T(2^3) = T(2^2) + b$
    - T(16) = T(8) + b → $T(2^4) = T(2^3) + b$
    - ...
    - **$T(2^k) = T(2^{k-1}) + b$**

Qual a profundidade? Ou seja, qual a altura da árvore?

Resposta: k

Complexidade: T(n) = k.b + a

Obs: k custos b + a (custo do caso base)

Busca Binária (5)

- **Análise de Complexidade – T(n):**
  - Para encontrar o valor da Complexidade **T(n) = k.b + a**
  - **Precisamos Resolver a Recorrência – Continuação...:**
    - Temos que: **T(n) = T(n/2) + b**
    - E também: **$T(2^k) = T(2^{k-1}) + b$**
    - **Assim, assumimos que: $n = 2^k$**
    - **Logo:**
      - Aplicando lg em ambos os lados, temos:
      - **k = lg n**
  - **Substituindo, temos:**
    - **T(n) = b. lg n + a**
    - **T(n) ∈ O(lg n) → Logarítmica**