

Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

AULA 06

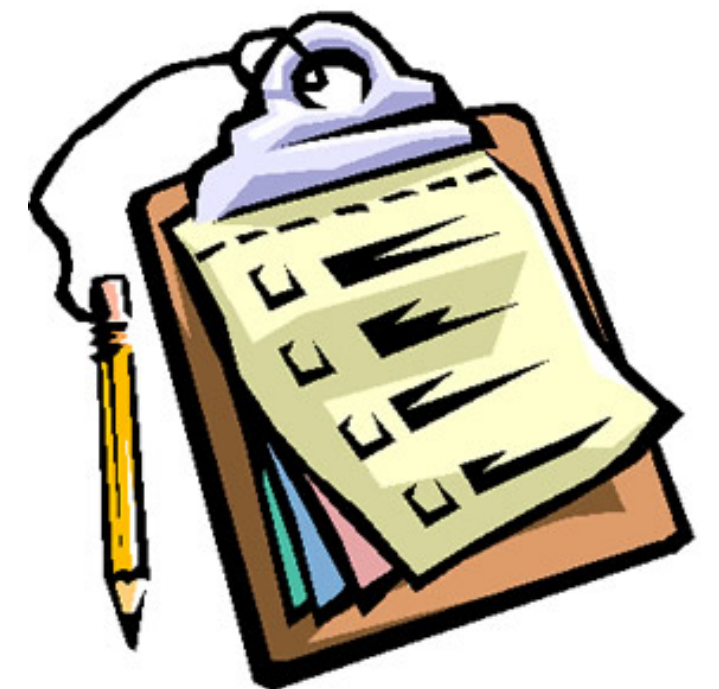
Estrutura de dados básico I (EDB1)

Prof. Msc. Janiheryson Felipe (Felipe)

Natal, RN
2023

OBJETIVO DA AULA

- Apresentar algoritmos de ordenação;
 - Conhecer a busca sequencial:
 - Conhecer a busca binária:





ALGORITMOS DE BUSCA

COMO RESOLVER?

Dada uma chave de busca e uma coleção de elementos, onde cada elemento possui um identificador único, desejamos encontrar o elemento da coleção que possui o identificador igual ao da chave de busca ou verificar que não existe nenhum elemento na coleção com a chave fornecida.

INFORMAÇÕES INICIAIS

Nos nossos exemplos, a coleção de elementos será representada por uma lista de inteiros.

- O identificador do elemento será o próprio valor de cada elemento.

Apesar de usarmos inteiros, os algoritmos que estudaremos servem para buscar elementos em qualquer coleção de elementos que possuam chaves que possam ser comparadas

INTRODUÇÃO

O problema da busca é um dos mais básicos na área de Computação e possui diversas aplicações.

- Buscar um aluno por sua matrícula.
- Buscar um cliente dado o seu CPF.
- Buscar uma pessoa dado o seu RG.



INTRODUÇÃO

Estudaremos algoritmos simples para realizar a busca

assumindo que os dados estão em uma lista.

- Existem estruturas de dados e algoritmos mais complexos utilizados para armazenar e buscar elementos. Estas abordagens não serão estudadas nesta disciplina.

O PROBLEMA DA BUSCA

Vamos criar uma função busca(lista, chave):

- A função deve receber uma lista de números inteiros e uma chave para busca.
- A função deve retornar o índice da lista que contém a chave ou o valor -1, caso a chave não esteja na lista.

O PROBLEMA DA BUSCA

chave = 45

lista	20	5	15	24	67	45	1	76	21	11
	0	1	2	3	4	5	6	7	8	9

chave = 100

lista	20	5	15	24	67	45	1	76	21	11
	0	1	2	3	4	5	6	7	8	9

No primeiro exemplo, a função deve retornar 5, enquanto no segundo exemplo, a função deve retornar -1.



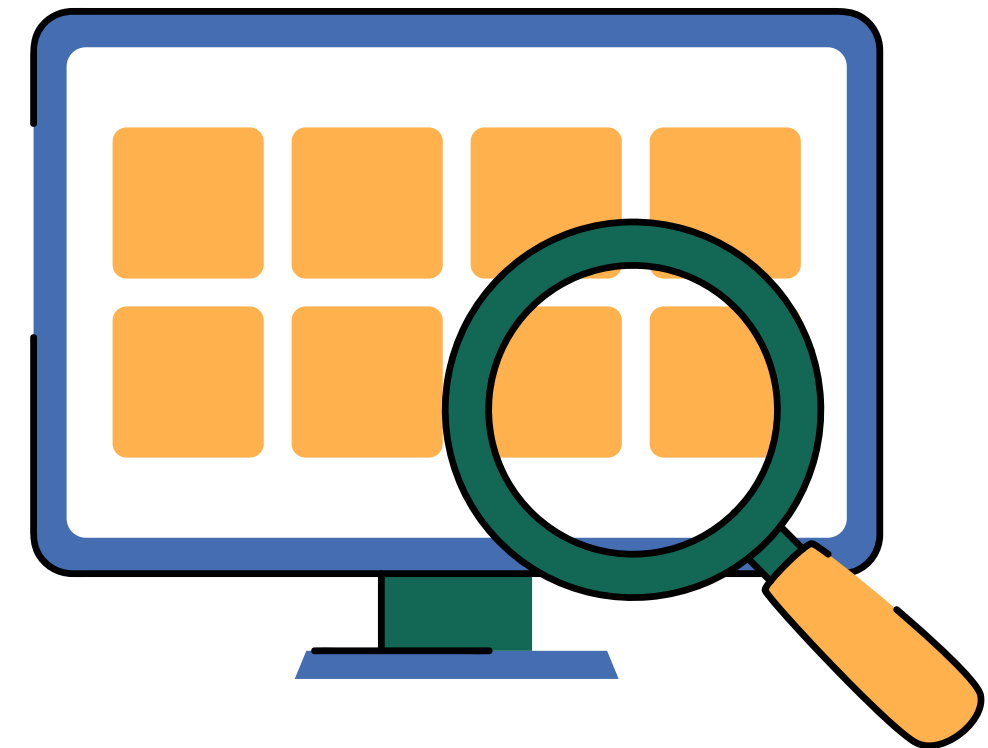
BUSCA LINEAR OU SEQUENCIAL

BUSCA LINEAR OU SEQUENCIAL

- O algoritmo de busca linear, também conhecido como busca sequencial, é um método simples e direto de encontrar um valor específico em um conjunto de dados. Esse algoritmo percorre todos os elementos de uma lista ou vetor, verificando se o valor desejado está presente em cada posição.

BUSCA LINEAR OU SEQUENCIAL

- Percorra a lista comparando a chave com os valores dos elementos em cada uma das posições.
- Se a chave for igual a algum dos elementos, retorne a posição correspondente na lista.
- Se a lista toda foi percorrida e a chave não for encontrada, retorne o valor -1.



PRÁTICA

- Dado um vetor qualquer elabore um algoritmo que busque uma determinado valor neste vetor. caso encontre retorne a posição desse valor e caso não encontre retorne -1



PRÁTICA

```
int buscaSequencial(vector<int> vetor, int chave){  
    for(int i = 0; i < vetor.size(); i++){  
        if(vetor[i] == chave){  
            return i;  
        }  
    }  
    return -1;  
}
```

PRÁTICA

```
def buscaSequencial(lista, chave):  
    n = len(lista)  
    for índice in range(n):  
        if lista[índice] == chave:  
            return índice  
  
    return -1
```

BUSCA LINEAR OU SEQUENCIAL

- O algoritmo de busca linear é muito simples de implementar, mas pode ser ineficiente para conjuntos de dados muito grandes. Isso porque ele precisa percorrer todos os elementos da lista, o que pode levar muito tempo em conjuntos de dados com muitos elementos.

BUSCA LINEAR OU SEQUENCIAL

- Quando a lista está organizada em ordem crescente ou decrescente, pode-se usar uma variação da busca linear chamada de busca linear ordenada. Nesse caso, o algoritmo pode parar a busca assim que encontrar um elemento maior do que o valor desejado, já que sabe que o valor não estará presente na lista a partir daquele ponto.

PRÁTICA

```
int buscaSequencial(vector<int> vetor, int chave){  
    for(int i = 0; i < vetor.size(); i++){  
        if(vetor[i] == chave){  
            return i;  
        }  
        if(vetor[i] > chave){  
            return -1;  
        }  
    }  
    return -1;  
}
```

BUSCA LINEAR OU SEQUENCIAL

- A complexidade do algoritmo de busca linear é $O(n)$, onde n é o tamanho do conjunto de dados a ser pesquisado. Isso ocorre porque o algoritmo precisa percorrer cada elemento da lista, um por um, até encontrar o valor desejado ou percorrer toda a lista e determinar que o valor não está presente.

BUSCA LINEAR OU SEQUENCIAL



O algoritmo também pode ser modificado de forma que mostre quantas ocorrências de uma determinada chave foram encontrada no vetor. Nessa abordagem o vetor deve ser totalmente percorrido caso não esteja ordenado.

PRÁTICA

```
int buscaSequencial(vector<int> vetor, int chave){  
    int ocorrencias = 0;  
    for(int i = 0; i < vetor.size(); i++){  
        if(vetor[i] == chave){  
            ocorrencias = ocorrencias + 1;  
        }  
    }  
    return ocorrencias;  
}
```



BUSCA BINÁRIA

BUSCA BINÁRIA

- A busca binária é um algoritmo mais eficiente, entretanto, requer que a lista esteja ordenada pelos valores da chave de busca.
- A ideia do algoritmo é a seguinte (assuma que a lista está ordenada pelos valores da chave de busca):

BUSCA BINÁRIA

- Verifique se a chave de busca é igual ao valor da posição do meio da lista.
- Caso seja igual, devolva esta posição.
- Caso o valor desta posição seja maior que a chave, então repita o processo, mas considere uma lista reduzida, com os elementos do começo da lista até a posição anterior a do meio.
- Caso o valor desta posição seja menor que chave, então repita o processo, mas considere uma lista reduzida, com os elementos da posição seguinte a do meio até o final da lista.

BUSCA BINÁRIA

chave = 15


lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9

pos_ini = 0
pos_fim = 9
pos_meio = 4

Como `lista[pos_meio] > chave`, devemos continuar a busca na primeira metade da região e, para isso, atualizamos a variável `pos_fim`.

BUSCA BINÁRIA

chave = 15




lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9
pos_ini	= 0									
pos_fim	= 3									
pos_meio	= 1									

Como $\text{lista}[\text{pos_meio}] < \text{chave}$, devemos continuar a busca na segunda metade da região e, para isso, atualizamos a variável `pos_ini`.

BUSCA BINÁRIA

chave = 15



lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9

pos_ini = 2


pos_fim = 3

pos_meio = 2

Finalmente, encontramos a chave ($\text{lista}[\text{pos_meio}] = \text{chave}$) e, sendo assim, devolvemos a sua posição na lista (pos_meio).

BUSCA BINÁRIA

chave = 50



lista

1	5	15	20	24	45	67	76	78	100
0	1	2	3	4	5	6	7	8	9

pos_ini = 0


pos_fim = 9

pos_meio = 4

Como `lista[pos_meio] < chave`, devemos continuar a busca na segunda metade da região e, para isso, atualizamos a variável `pos_ini`.

BUSCA BINÁRIA

chave = 50



lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9

pos_ini = 5


pos_fim = 9

pos_meio = 7

Como `lista[pos_meio] > chave`, devemos continuar a busca na primeira metade da região e, para isso, atualizamos a variável `pos_fim`.

BUSCA BINÁRIA

chave = 50



lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9

pos_ini = 5


pos_fim = 6

pos_meio = 5

Como $\text{lista}[\text{pos_meio}] < \text{chave}$, devemos continuar a busca na segunda metade da região e, para isso, atualizamos a variável `pos_ini`

BUSCA BINÁRIA

chave = 50



lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9

pos_ini = 6

pos_fim = 6

pos_meio = 6

Como `lista[pos_meio] > chave`, devemos continuar a busca na primeira metade da região e, para isso, atualizamos a variável `pos_fim`.

BUSCA BINÁRIA

chave = 50

lista

1	5	15	20	24	45	67	76	78	100
0	1	2	3	4	5	6	7	8	9

pos_ini = 6

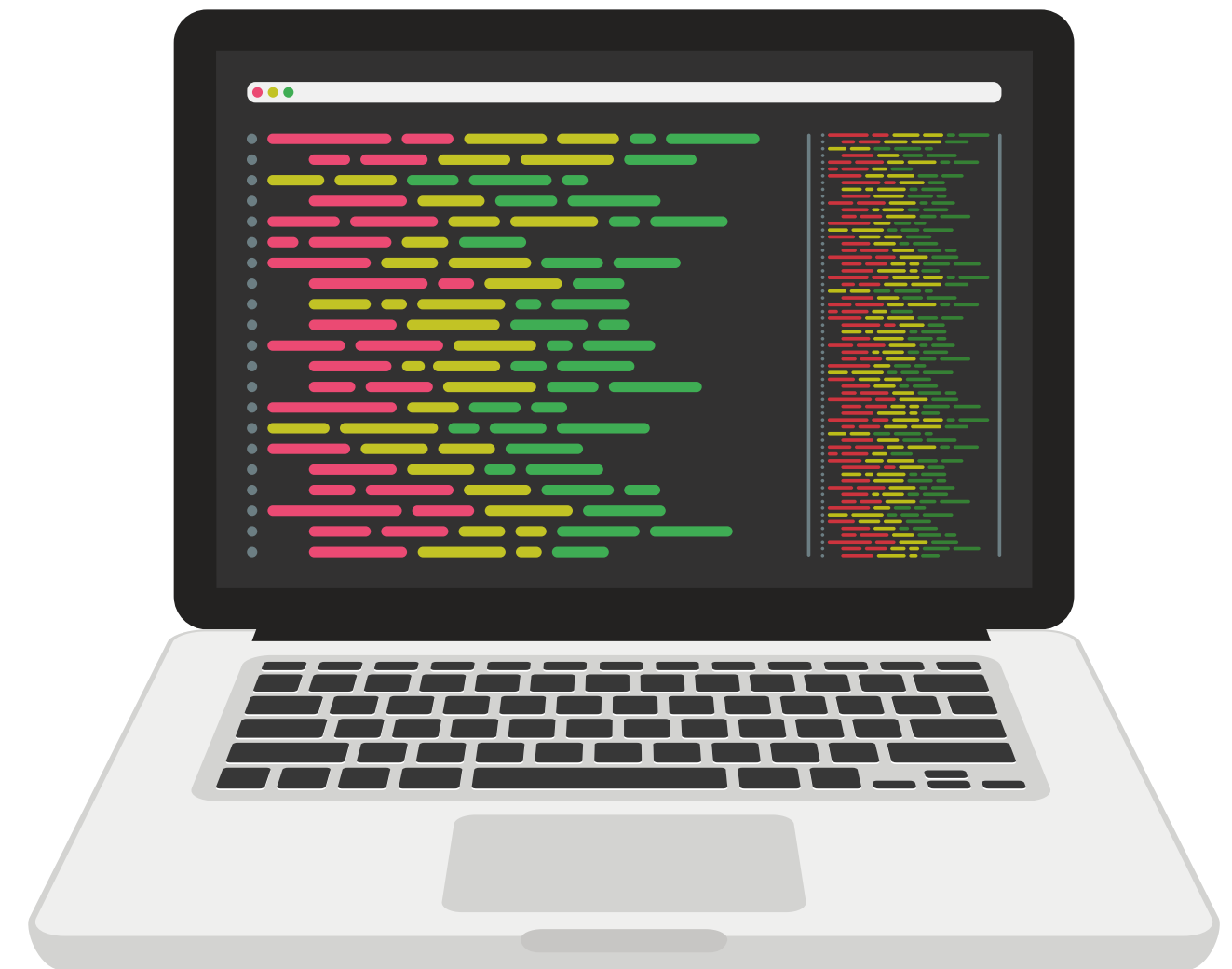
pos_fim = 5

pos_meio = 5

Como $\text{pos_ini} > \text{pos_fim}$, determinamos que a chave não está na lista e retornamos o valor -1.

PRÁTICA

- Escreva o algoritmo da busca binária, de forma que os passos necessários sejam corretamente executados.



PRÁTICA

```
int buscaBinaria(vector<int> vetor, int chave){
    int pos_ini = 0;
    int pos_fim = vetor.size() - 1;

    while (pos_ini <= pos_fim){
        int pos_meio = (pos_ini + pos_fim);
        if (vetor[pos_meio] == chave){
            return pos_meio;
        }if (vetor[pos_meio] > chave){
            pos_fim = pos_meio - 1;
        }else{
            pos_ini = pos_meio + 1;
        }
    }
    return -1;
}
```

BUSCA BINÁRIA

O algoritmo de busca binária é mais eficiente do que a busca linear em listas ordenadas, pois reduz pela metade o tamanho da lista de busca a cada iteração. A complexidade do pior caso do algoritmo de busca binária é $O(\log n)$, onde n é o número de elementos na lista. Isso significa que o algoritmo leva um tempo de busca proporcional ao logaritmo do tamanho da lista, tornando-o eficiente mesmo para grandes conjuntos de dados.



COMPARANDO OS ALGORITMOS

BUSCA SEQUENCIAL

Na melhor das hipóteses, a chave de busca estará na posição 0. Portanto, teremos um único acesso em `lista[0]`.

- Na pior das hipóteses, a chave é o último elemento ou não pertence à lista e, portanto, acessamos todos os n elementos da lista.
- É possível mostrar que, se as chaves possuírem a mesma probabilidade de serem requisitadas, o número médio de acessos nas buscas cujas chaves encontram-se na lista será igual a $(n+1)/2$

BUSCA BINÁRIA

Na melhor das hipóteses, a chave de busca estará na posição do meio da lista. Portanto, teremos um único acesso.

- Na pior das hipóteses, dividimos a lista até a que ela fique com um único elemento (último acesso realizado à lista).
- Note que, a cada acesso, o tamanho da lista é diminuído, pelo menos, pela metade.

BUSCA BINÁRIA

Quantas vezes um número pode ser dividido por dois antes dele se tornar igual a um?

- Esta é exatamente a definição de logaritmo na base 2.
- Ou seja, no pior caso o número de acesso é igual a $\log_2 n$.
- É possível mostrar que, se as chaves possuírem a mesma probabilidade de serem requisitadas, o número médio de acessos nas buscas cujas chaves encontram-se na lista será igual a: $(\log_2 n) - 1$

BUSCA SEQUENCIAL VS BINÁRIA

Para se ter uma ideia da diferença de eficiência dos dois algoritmos, considere uma lista com um milhão de itens (10^6 itens).

- Com a busca sequencial, para buscar um elemento qualquer da lista necessitamos, em média, de:

$$(10^6 + 1)/2 \approx 500000 \text{ acessos.}$$

- Com a busca binária, para buscar um elemento qualquer da lista necessitamos, em média, de:

$$(\log_2 10^6) - 1 \approx 19 \text{ acessos.}$$

QUESTÃO 1 - BANCA COMVEST

Assinale a proposição VERDADEIRA:

- ☐ A Na busca binária o vetor não precisa estar ordenado.
- ☐ B Na busca sequencial o vetor não precisa estar ordenado.
- ☐ C Na busca sequencial o vetor precisa estar ordenado.
- ☐ D A busca sequencial sempre garante que o elemento a ser procurado será encontrado.
- ☐ E A busca binária sempre garante que o elemento a ser procurado será encontrado.

QUESTÃO 2 - BANCA CESGRANRIO

Seja uma função que realiza uma busca binária sobre um array de números inteiros ordenados. Não se sabe, em princípio, se os números estão ordenados ascendente ou decendentemente. O cabeçalho dessa função é o seguinte:

int busca (int [] vet, int elem)

Isto é, a função busca recebe um array de números inteiros (vet) e um número inteiro (elem) como parâmetros, e retorna um número inteiro. Caso exista em vet um inteiro igual a elem, a função retornará o índice desse inteiro no array; caso contrário, a função retornará -1.

O algoritmo de busca binária produz um índice (ind) a cada iteração sobre o array, tendo em vista comparar o elemento que se deseja procurar (elem) com o elemento vet [ind]. Isto é:

if (vet [ind] == elem) return ind;

No comando acima, diz-se que houve uma visita ao elemento vet [ind].

Admita que a função busca foi chamada por meio do comando a seguir:

int resp = busca (vet, 50);

Sabendo-se que os elementos visitados foram **54, 17, 33 e 50**, nesta ordem, qual array foi passado como parâmetro para a função busca?

- A** [95, 90, 87, 54, 52, 50, 33, 17, 11, 10]
- B** [5, 10, 11, 17, 33, 50, 54, 87, 90, 95]
- C** 121, 111, 93, 87, 60, 54, 50, 33, 17, 5]
- D** [5, 17, 33, 50, 54, 60, 87, 93, 111, 121]
- E** [130, 121, 111, 90, 70, 60, 54, 50, 33, 17]

QUESTÃO 3 - BANCA UFAM

Considere um vetor de n posições, composto de números de matrículas de alunos de uma universidade. Ao executarmos uma busca sequencial para verificar se a matrícula de determinado aluno está contida, ou não, no vetor, o número de comparações realizadas na busca de uma matrícula dada no vetor, considerando o pior caso, é:

A $n - 1$

B $n + 1$

C n

D $n - 2$

E $n + 2$

QUESTÃO 4 - BANCA METRÓPOLE

É um algoritmo de busca em vetores que segue o paradigma de divisão e conquista.

I. Percorrer a lista comparando a chave com os valores dos elementos em cada uma das posições.

II. Se a chave for igual à algum dos elementos, retornar a posição correspondente na lista.

III. Se a lista toda for percorrida e a chave não for encontrada, retornar o valor -1.

A I, apenas.

B II, apenas.

C III, apenas.

D I e II, apenas.

E I, II e III

QUESTÃO 5 - BANCA METRÓPOLE

Sobre o algoritmo de busca binária, marque a alternativa INCORRETA.

- ☐ A É um algoritmo mais eficiente, entretanto, requer que a lista esteja ordenada pelos valores da chave de busca.
- ☐ B É um eficiente algoritmo para encontrar um item em uma lista ordenada de itens. Funciona dividindo repetidamente pela metade a porção da lista que deve conter o item, até reduzir as localizações possíveis à apenas um.
- ☐ C O funcionamento consiste em, a partir do primeiro registro, pesquisar sequencialmente até encontrar o valor procurado ou até chegar ao fim do vetor e então parar.
- ☐ D Um dos modos mais comuns de se usar a busca binária é para encontrar um item em um array.
- ☐ E É um algoritmo de busca em vetores que segue o paradigma de divisão e conquista.

QUESTÃO 7 - BANCA CESGRANRIO

Em uma agência bancária, as filas de atendimento são ordenadas da esquerda para a direita, e o gerente dessa agência percebeu a presença equivocada de um idoso, com a senha 52, na fila de atendimento não preferencial. Visando a sanar o equívoco, o gerente resolveu que, na primeira oportunidade, faria uma busca no sistema para saber se a senha 52 ainda estava ativa, indicando a presença do idoso na fila de atendimento não preferencial. Em caso de resposta positiva, procuraria o cliente para trocar sua senha por outra de atendimento preferencial; se não, apenas registraria o fato para posterior discussão no grupo de qualidade de atendimento. Considerando o uso de um algoritmo de busca sequencial otimizado, partindo da esquerda para a direita, e as sequências hipotéticas das senhas da fila de atendimento não preferencial e suas regras de ordenação, segundo as quais quem está à esquerda é atendido antes de quem está à direita, o menor número de comparações para o gerente conhecer o resultado de sua busca ocorre em

A

Regras de ordenação
atendimento não preferencial
Sequência ordenada crescentemente

Sequência das senhas na fila de
23; 45; 81; 97; 112; 138; 154

B

Regras de ordenação
atendimento não preferencial
Sequência ordenada crescentemente

Sequência das senhas na fila de
13; 25; 37; 44; 52; 78; 83; 91

C

Regras de ordenação
atendimento não preferencial
Sequência ordenada crescentemente

Sequência das senhas na fila de
17; 28; 32; 49; 67; 85; 94; 103

D

Regras de ordenação
atendimento não preferencial
Sequência desordenada

Sequência das senhas na fila de
27; 95; 148; 117; 33; 59; 52

E

Regras de ordenação
atendimento não preferencial
Sequência desordenada

Sequência das senhas na fila de
32; 48; 12; 55; 93; 27; 66

QUESTÃO 7 - BANCA CESGRANRIO

ANO: 2021 BANCA:
CESGRANRIO ÓRGÃO:
BANCO DO BRASIL
PROVA: CESGRANRIO -
2021 - BANCO DO BRASIL
- AGENTE DE
TECNOLOGIA

QUESTÃO 8

Reescreva o algoritmo da busca binária de forma que ele seja capaz de verificar se a lista está ordenada de forma crescente ou decrescente e desta forma adapte a busca considerando essa ordenação



DÚVIDAS???

