Algoritmos e Estruturas de Dados 1 (AED1)
Lista em vetor

Uma lista (ou sequência) é uma coleção de itens que apresenta uma ordem estável.

Supondo uma lista com n elementos, queremos que ela aceite as operações:

- Imprimir: percorrer em ordem imprimindo cada elemento.
- Seleção: pegar o conteúdo do k-ésimo item.
- Busca: encontrar um item pelo seu conteúdo.
- Inserção: inserir um item na posição k.
- Remoção: remover um item da posição k.
    - Temos que k \in [0, n), i.e., k \in {0, 1, 2, …, n - 1}.

Um vetor é uma estrutura de dados que armazena

- uma sequência de objetos do mesmo tipo
    - em posições consecutivas da memória.
- Por isso, é bastante natural usar vetores para implementar listas.
    - Veremos como implementar as operações anteriores em um vetor.

Usar um vetor v de tamanho TAM_MAX

```
#define TAM_MAX 1000000
```

O vetor pode ser declarado:

- estaticamente
  ```
      int v[TAM_MAX];
  ```
- dinamicamente
  ```
      int *v = (int *)malloc(TAM_MAX * sizeof(int));
  ```

Quiz1: Existe alguma estratégia que não exige alocar um vetor de tamanho fixo?

Operações, implementações e eficiência:

Imprimir: percorrer em ordem imprimindo cada elemento.

```
void imprime(int v[], int n) {
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", v[i]);
    printf("\n");
}
```

- Eficiência de tempo: linear no tamanho da lista, i.e., O(n).

Seleção: pegar o conteúdo do k-ésimo item.

```
int selecao(int v[], int n, int k) {
    return v[k];
}
```

- Eficiência de tempo: constante, i.e., O(1).

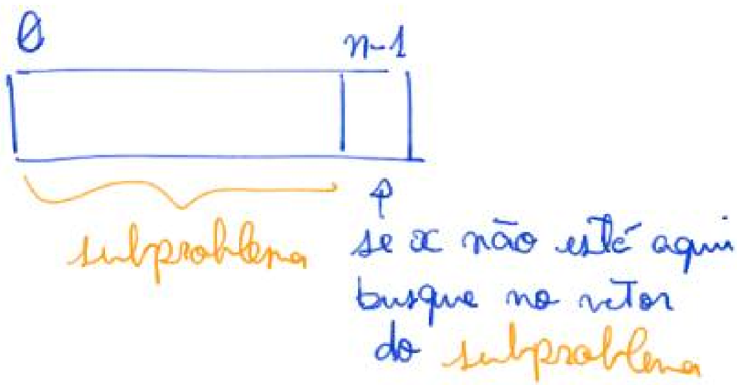
Busca: encontrar um item pelo seu conteúdo x.

- Ideia do algoritmo iterativo:
    - Percorrer o vetor verificando cada posição.

```
int buscaI(int v[], int n, int x) {
    int i;
    i = n - 1;
    while (i >= 0 && v[i] != x)
        i -= 1;
    return i;
}
```

    - Quiz2: como esse algoritmo indica que não encontrou?
- Eficiência de tempo: O(n) no pior caso.

- Ideia do algoritmo recursivo:
    - Se o item buscado não é o último elemento do vetor corrente,
        - busque recursivamente no subvetor com um elemento a menos.

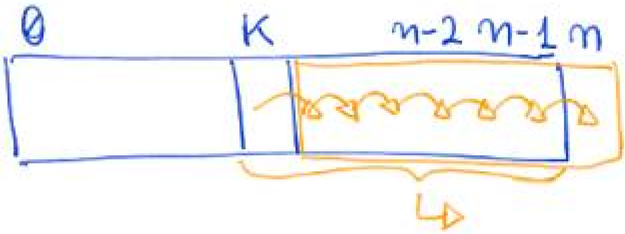



```
int buscaR(int v[], int n, int x) {
    if (n == 0)
        return -1;
    if (x == v[n - 1])
        return n - 1;
    return buscaR(v, n - 1, x);
}
```

- Eficiência de tempo: O(n) no pior caso.

Inserção: inserir um item x na posição k.

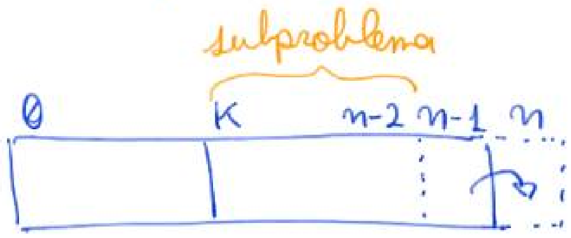
- Ideia do algoritmo iterativo:
  - Deslocar itens à direita da posição k uma posição para a direita.
    - Note que a ordem deste deslocamento faz diferença.
  - Então inserir na posição k, que foi liberada.




```c
int insereI(int v[], int n, int x, int k) {
    int i;
    for (i = n; i > k; i--)
        v[i] = v[i - 1];
    v[i] = x;
    return n + 1;
}
```

  - Quiz3: Se eu queria inserir na posição k, porque faço v[i] = x ?
- Eficiência de tempo: leva tempo O(n - k), que é O(n) no pior caso.

- Ideia do algoritmo recursivo:
  - Copie v[n - 1] para v[n] e insira recursivamente
    - no subvetor com um elemento a menos.



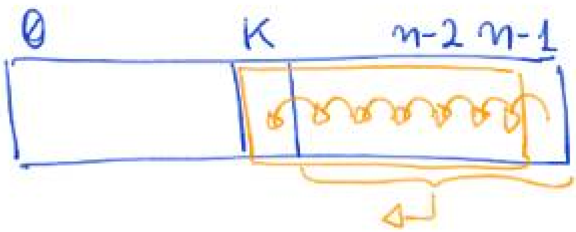
copie v[n-1] p/ v[n] e
insira x no início do
vetor do subproblema

```c
int insereR(int v[], int n, int x, int k) {
    if (k == n)
        v[n] = x;
    else {
        v[n] = v[n - 1];
        insereR(v, n - 1, x, k);
    }
    return n + 1;
}
```

- Eficiência de tempo: leva tempo O(n - k), que é O(n) no pior caso.

Remoção: remover um item da posição k.

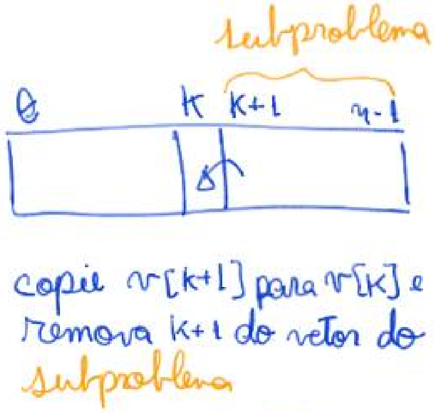
- Ideia do algoritmo iterativo:
    - Deslocar itens à direita da posição k uma posição para a esquerda.
        - Note que a ordem deste deslocamento faz diferença.




```
int removeI(int v[], int n, int k) {
    int i;
    for (i = k + 1; i < n; i++)
        v[i - 1] = v[i];
    return n - 1;
}
```

- Eficiência de tempo: leva tempo O(n - k), que é O(n) no pior caso.

- Ideia do algoritmo recursivo:
    - Copie v[k + 1] para v[k] e remova recursivamente o k + 1
        - do subproblema de tamanho n - (k + 1).




```
int removeR(int v[], int n, int k) {
    if (k == n - 1)
        return n - 1;
    v[k] = v[k + 1];
    return removeR(v, n, k + 1);
}
```

- Eficiência de tempo: leva tempo O(n - k), que é O(n) no pior caso.

Sintetizando, vimos como implementar listas em vetores contíguos:

- Imprimir custa O(n),
- Seleção custa O(1),
- Busca custa O(n),
- Inserção custa O(n - k),
- Remoção custa O(n - k).

Quiz5: Como modificar as operações para manter a lista em ordem crescente?

- E qual a eficiência das operações nesse caso?

Quiz6: Como modificar as operações se a ordem dos elementos não importar?

- E qual a eficiência das operações nesse caso?

Bônus:

- Considere o problema de remover todas as ocorrências de um elemento x.

```
int removeTodos(int v[], int n, int x) {
    int k;
    while ((k = buscaI(v, n, x)) != -1)
        n = removeI(v, n, k);
    return n;
}
```

  - Quiz7: Qual a eficiência de tempo de pior caso de removeTodos?

- Considere o seguinte algoritmo para o mesmo problema.

```
int removeTodos2(int v[], int n, int x) {
    int i = 0, j;
    for (j = 0; j < n; j++)
        if (v[j] != x) {
            v[i] = v[j];
            i++;
        }
    return i;
}
```

  - Quiz8: Qual a eficiência de tempo de pior caso de removeTodos2?
  - Quiz9: Como mostrar que a função anterior está correta?
    - O que a variável i representa?
    - Qual o invariante principal do algoritmo?

Quiz1: Existe alguma estratégia que não exige alocar um vetor de tamanho fixo?

- Podemos redimensionar automaticamente um vetor quando ele fica cheio.
  - Como fica a eficiência desses vetores auto dimensionáveis?