Tópicos de Programação

Arthur Casals
(arthur.casals@usp.br)

IME - USP

Aula 3:

- Fundamentos de Estruturas de Dados

Na aula passada...

- Formas de se expressar um algoritmo:
  - Narrativa
  - Pseudocódigo
  - Fluxograma

Na aula passada...

Como diminuir a complexidade do problema a ser resolvido?

› Dividir para conquistar

› Planejamento reverso

Na aula passada...

- Dividir para conquistar:
  - Dividir o problema original em problemas menores
  - Re-dividir os problemas menores, caso seja necessário
  - **Analisar a estrutura final para garantir a coerência do problema original**

Na aula passada...

- Dividir para conquistar:
  - Algumas vezes, um problema é composto por instâncias menores do mesmo problema. Tais problemas possuem *estrutura recursiva*.
  - Algumas vezes, a solução de um problema envolve a repetição de um mesmo processo, até que uma certa condição seja atingida. Esta característica é chamada de *iteratividade*.

Na aula passada...

- Dividir para conquistar:
  - Programação dinâmica: recursividade + tabela
  - Ideia: ao invés de resolver um problema de forma recursiva, resolver instâncias menores do mesmo problema e armazenar os resultados obtidos em uma tabela, de forma que possam ser utilizados posteriormente.

Na aula passada...

- Planejamento reverso:
  - Identificar o resultado desejado
  - Identificar os dados de entrada
  - Estabelecer o processo que transforma os dados de entrada no resultado desejado
  - **Analisar a estrutura final para garantir a coerência do problema original**

Na aula passada...

- Metodologia para a construção de algoritmos:
  - Entender o problema inicial;
  - Quebrar o problema inicial em problemas menores, se possível;
  - Identificar entradas e saídas;
  - Determinar o que deve ser feito para cada problema;
  - Construir um esboço do algoritmo;
  - Executar o algoritmo, verificando se para cada entrada a saída esperada é obtida.

Fundamentos de estruturas de dados

- **Estrutura de dados:** no contexto de programação, refere-se à organização e representação de informações utilizada para se obter abstração e eficiência no processo computacional.

Fundamentos de estruturas de dados

Motivação:

- Algoritmos: técnicas para resolver problemas computacionais (metodologia para solução de problemas, busca, ordenação, etc.)
- Estruturas de dados: permitem que algoritmos manipulem informações eficientemente

Fundamentos de estruturas de dados

Estruturas de dados possuem:

› Diferentes representações;

› Diferentes finalidades;

› Diferentes **implementações**

Fundamentos de estruturas de dados

Implementações:

- Determinam os tipos de dados armazenados;
- **Determinam como a estrutura será armazenada em memória**

Fundamentos de estruturas de dados

Sobre a memória do computador:

- É como um grande estacionamento de carros, com vagas numeradas e de tamanho único
- Estruturas de dados utilizam este conceito em sua implementação

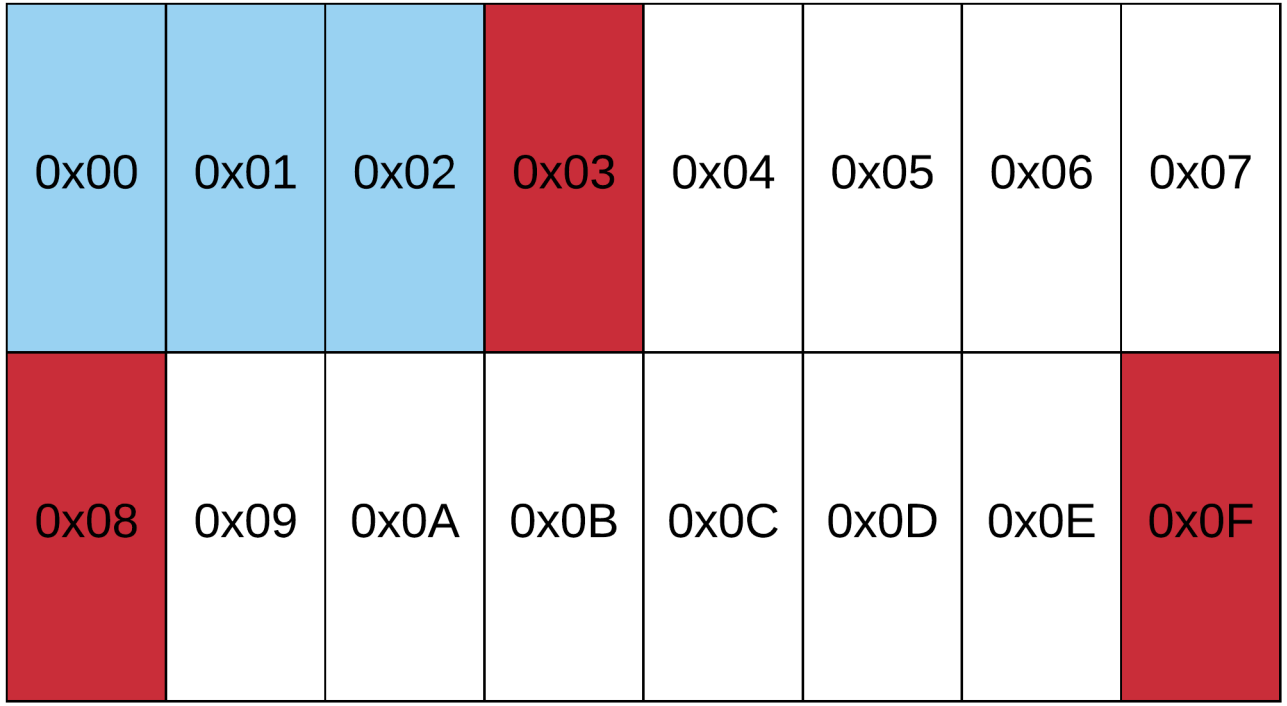
Fundamentos de estruturas de dados

Sobre a memória do computador:

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F

Fundamentos de estruturas de dados

Sobre a memória do computador:



Fundamentos de estruturas de dados

Estruturas de dados:

- Listas lineares (*arrays*)
- Listas ligadas
- Pilhas
- Filas
- Grafos
- Árvores

Fundamentos de estruturas de dados

Estruturas de dados:

- Listas lineares (*arrays*)
- Listas ligadas
- Pilhas
- Filas
- Grafos
- Árvores

Fundamentos de estruturas de dados

Listas lineares *(arrays):*

› Coleção de dados armazenados em espaços de memória *contínuos*

› Possui: *tamanho* e *início*

› Objetivo: agrupar dados do mesmo tipo

› Representação: espaços contínuos, numerados (*indexados*)

Fundamentos de estruturas de dados

Listas lineares *(arrays):*

› Exemplo: imagine uma lista com seis posições. Então:

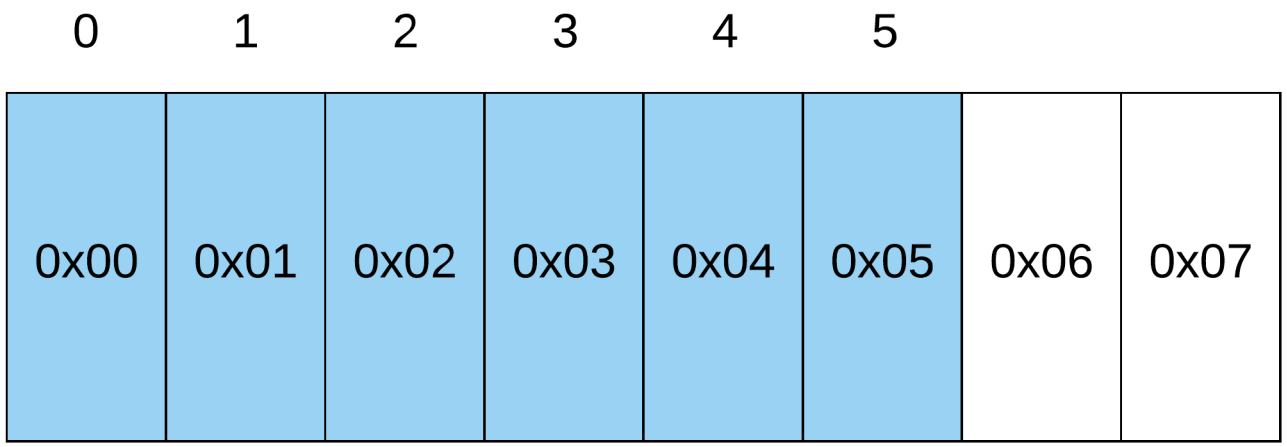
- Tamanho: 6
- Início: posição 0*

**Indexação: ao longo do curso, usaremos posição inicial = 0*

Fundamentos de estruturas de dados

Listas lineares *(arrays):*

› Exemplo: imagine uma lista com seis posições. Então:



Fundamentos de estruturas de dados

Listas lineares *(arrays):*

- Como fica essa lista em C?
  - int array1 = {12, 24, 36, 48, 60, 72}

Fundamentos de estruturas de dados

Listas lineares *(arrays)*:

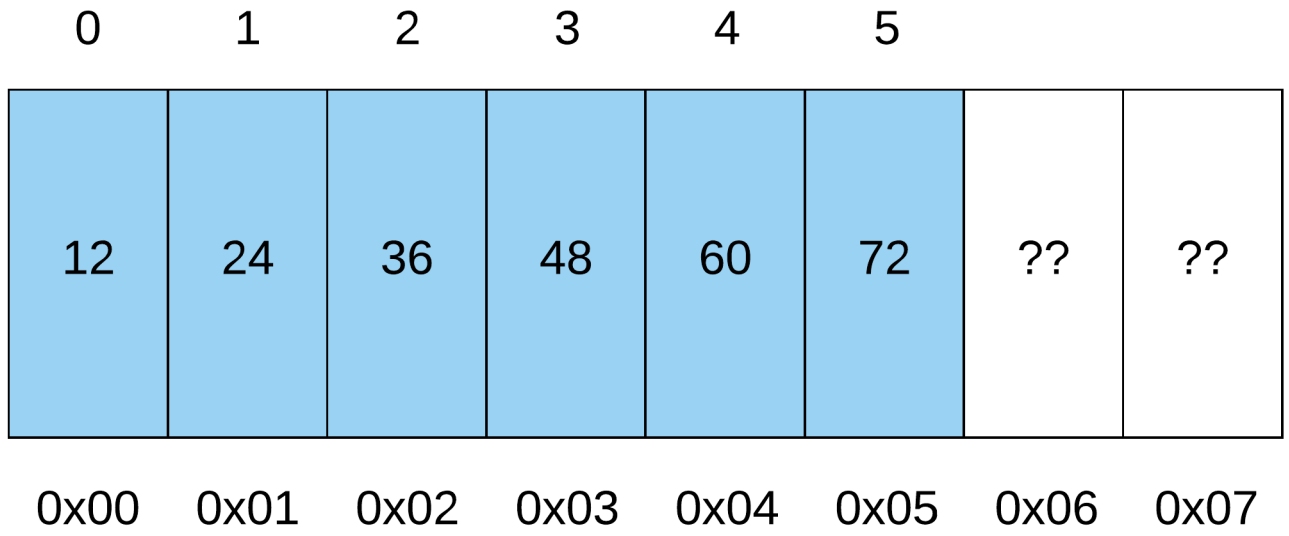
› Como fica essa lista em C?

- int batatinha[] = {12, 24, 36, 48, 60, 72}
- int: define o **tipo** dos dados na lista
- []: identifica “batatinha” como uma lista linear
- Cada elemento da lista pode ser acessado através do referenciamento sua posição: batatinha[2] acessa o número 36

Fundamentos de estruturas de dados

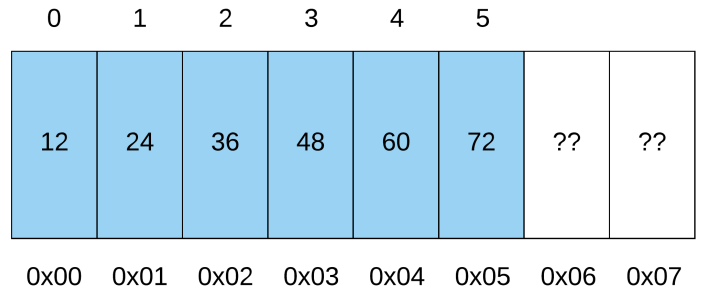
Listas lineares *(arrays)*:

› Como batatinha está na memória?



Fundamentos de estruturas de dados

Listas lineares *(arrays)*:



› Outras considerações:

- Ao ser alocada em memória, a lista ocupa um espaço aleatório
- A referência aos elementos da lista é feita através de posição na lista. ***Posição na lista não é posição em memória!***
- O tamanho de um *array* é sempre fixo
- Algumas operações podem custar caro!

Fundamentos de estruturas de dados

Listas lineares *(arrays):*

› Outro exemplo:

```
int pamonha[]  = {7, 14, 21, 28, 35, 42};
int primeiroElemento = pamonha[0]; //7
int terceiroElemento = pamonha[2]; //21
int quintoMaisSegundo = pamonha[4]  + pamonha[1]; //49
```

Fundamentos de estruturas de dados

Listas lineares *(arrays):*

› Outro exemplo:

```
char vogais[]  = {'a', 'e', 'i', 'o', 'u'};
char primeiroElemento = vogais[0]; //a
char terceiroElemento = vogais[2]; //i
char quintoMaisSegundo = vogais[4]  + vogais[1]; //??
```

Fundamentos de estruturas de dados

Listas lineares *(arrays)*:

› Operações com algoritmos:

- Como inserir elementos de forma ordenada?
- Como remover elementos e ainda assim manter a ordem?
- Como trocar elementos de posição?

Fundamentos de estruturas de dados

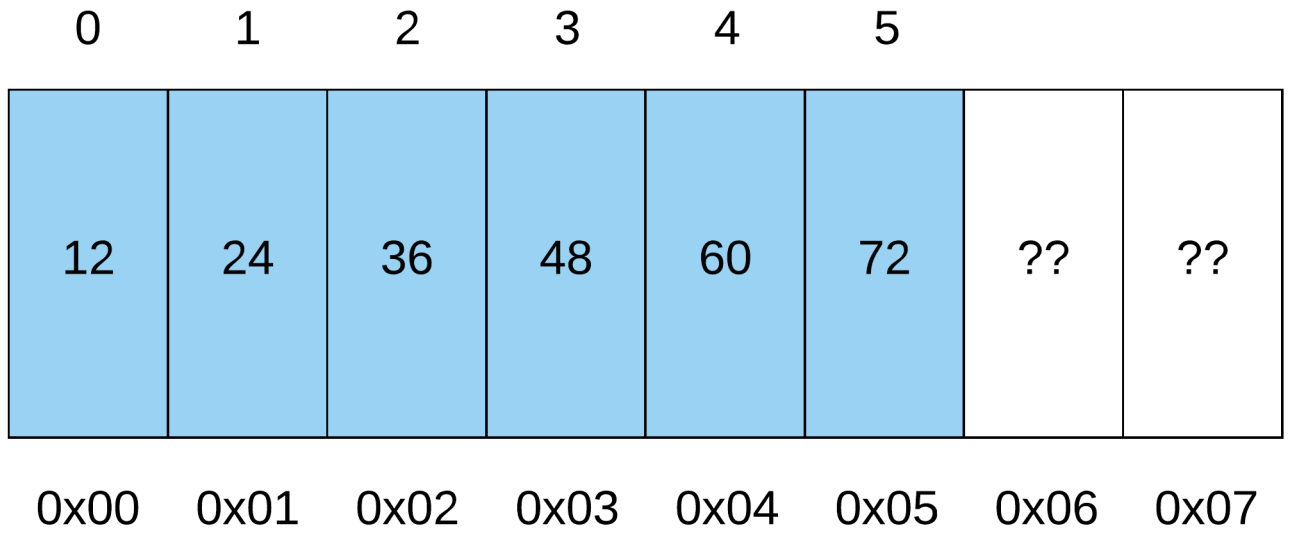
Listas lineares *(arrays)*:

- Inserção de elementos de forma ordenada:
  - Se a ordem deve ser mantida e o tamanho da lista é fixa, como fazer para inserir um novo elemento?
  - Ao se inserir um novo elemento, como garantir a ordenação geral?

Fundamentos de estruturas de dados

Listas lineares *(arrays):*

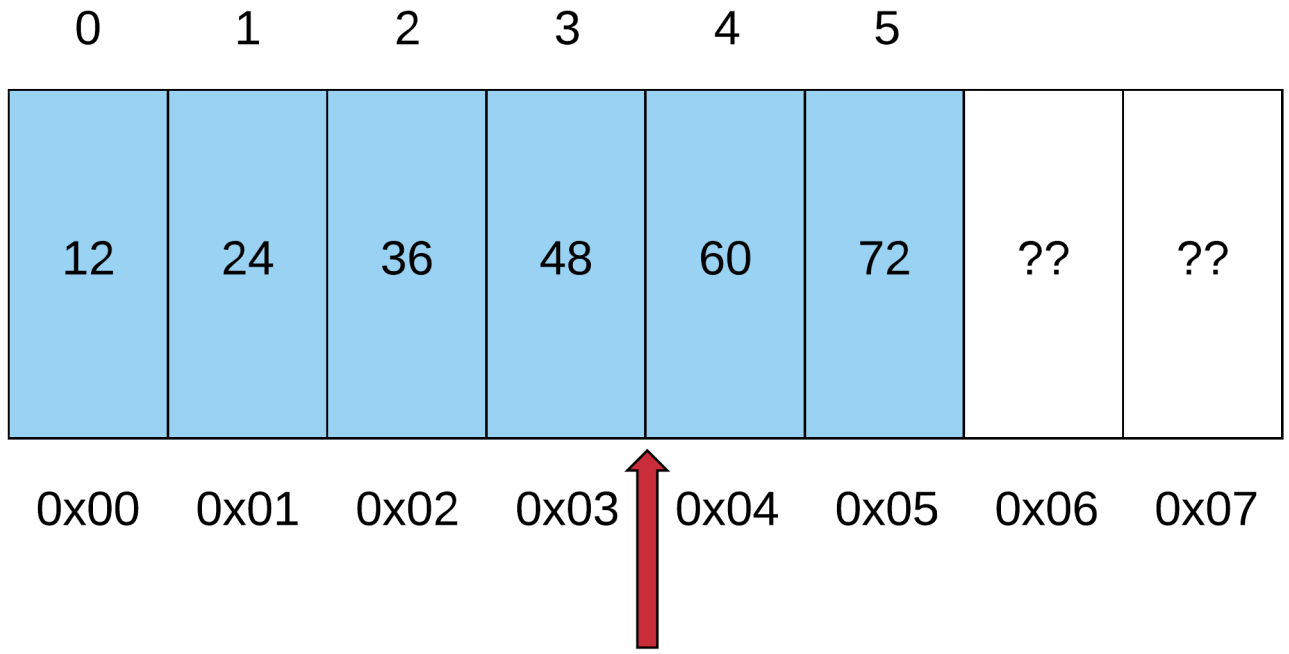
› Inserção de elementos de forma ordenada: **batatinha**



Fundamentos de estruturas de dados

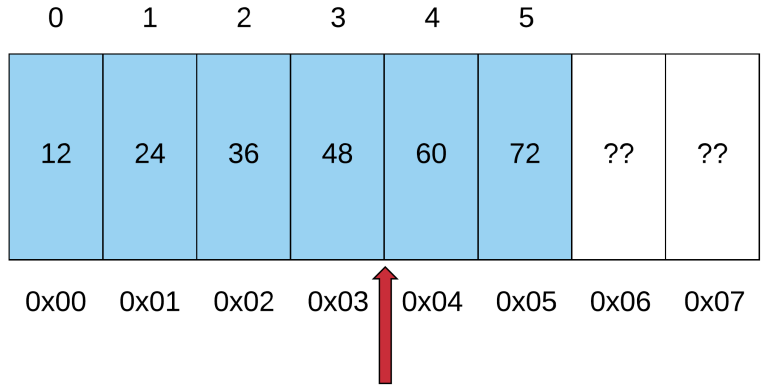
Listas lineares *(arrays)*:

› Objetivo: inserir o número 55 **ordenadamente**



Fundamentos de estruturas de dados

Listas lineares *(arrays):*



› Algoritmo:

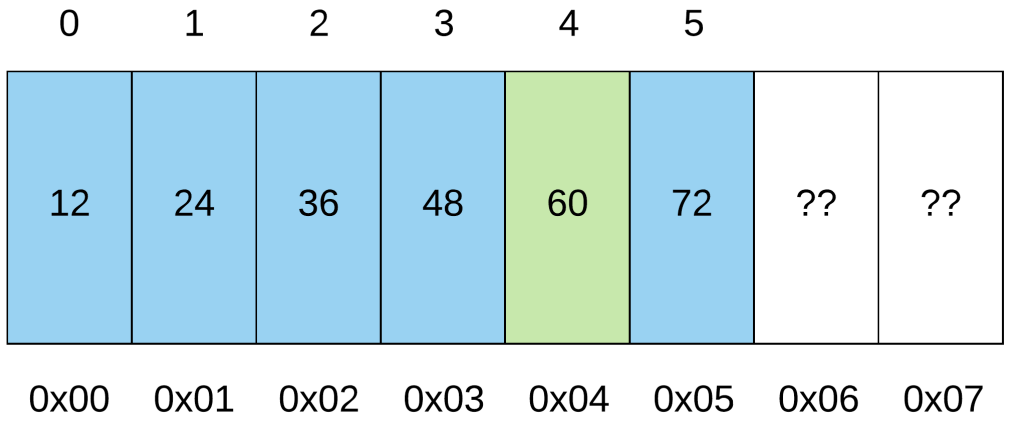
1. Achar a posição correta de inserção
2. “Abrir espaço”
3. Inserir o elemento desejado

Fundamentos de estruturas de dados

Listas lineares *(arrays):*

1. **Achar a posição correta de inserção:**

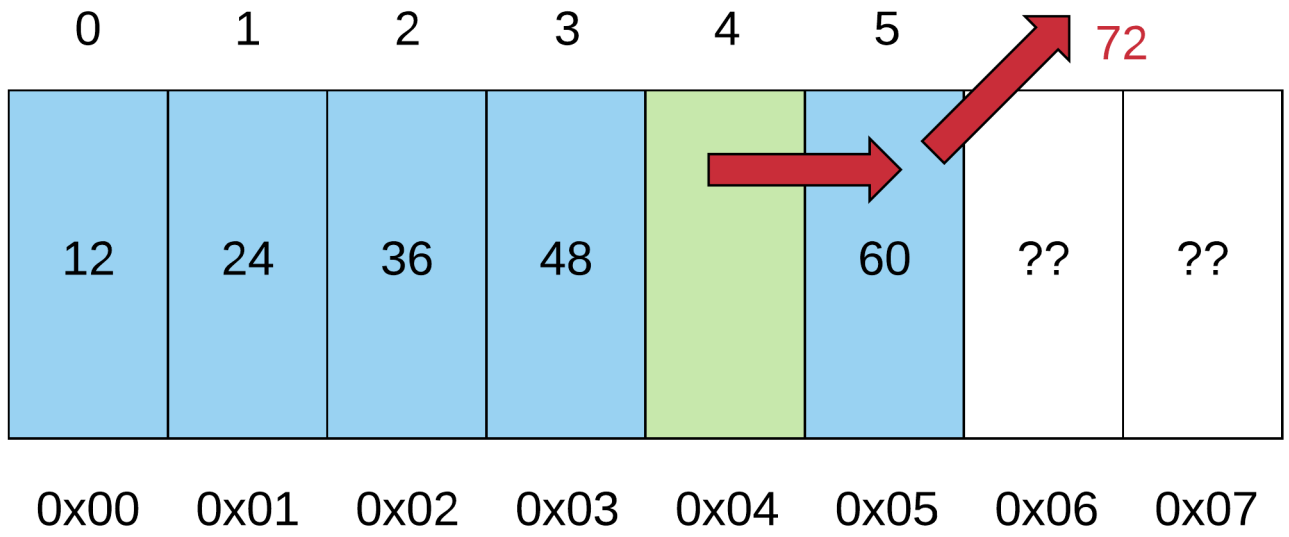
```
posição = 0;
achou = Falso;
Enquanto (achou = falso) e (posição < tamanho):
        se batatinha[posição] >= 55
                achou = Verdadeiro;
        senão
                posição = posição + 1;
Fim do Enquanto
Retorna(posição);
```



Fundamentos de estruturas de dados

Listas lineares *(arrays)*:

2. **Abrir espaço:**



Fundamentos de estruturas de dados

Listas lineares *(arrays)*:

2. **Abrir espaço:**

posição = 4;

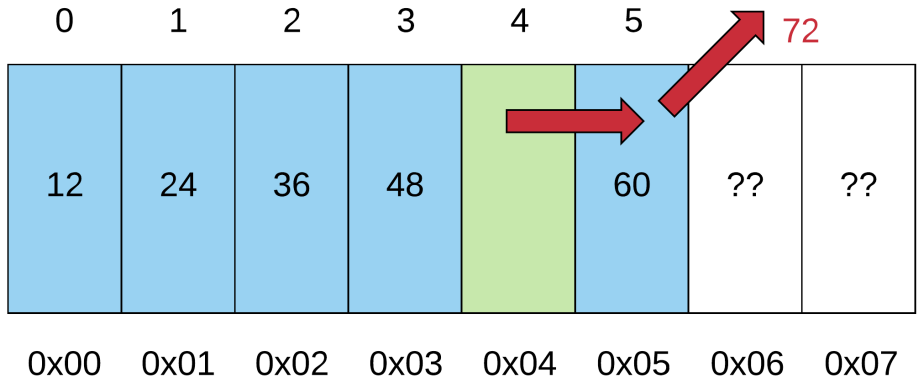
x = tamanho - 1;

Enquanto x > posição:

batatinha[x] = batatinha[x-1];

x = x - 1;

Fim do Enquanto

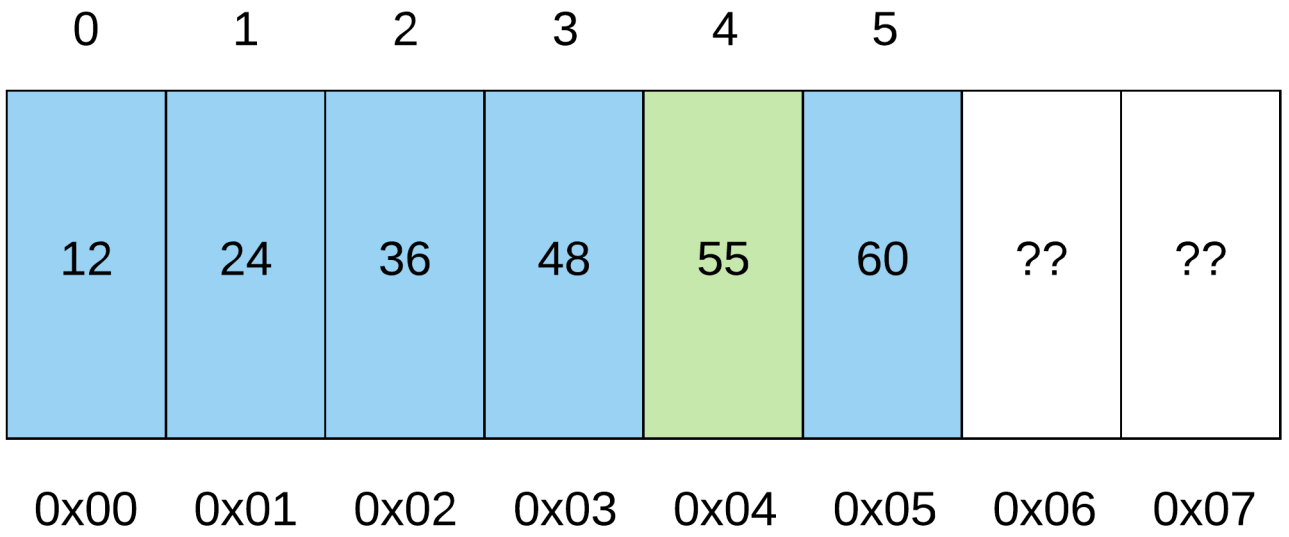


Fundamentos de estruturas de dados

Listas lineares *(arrays):*

3. **Inserir elemento desejado:**

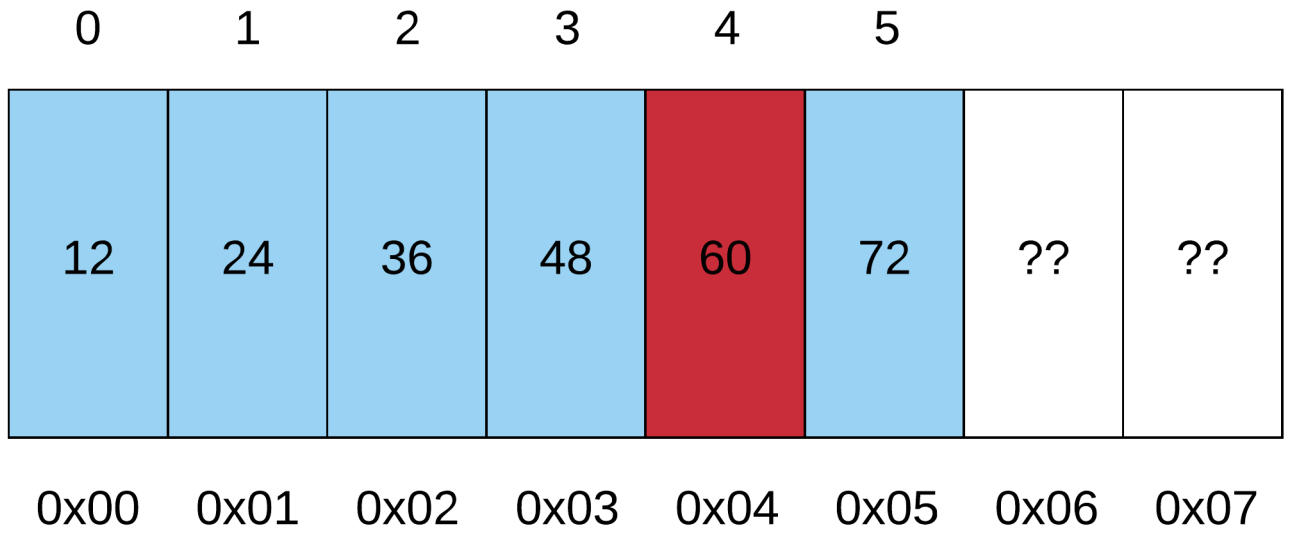
batatinha[4] = 55;



Fundamentos de estruturas de dados

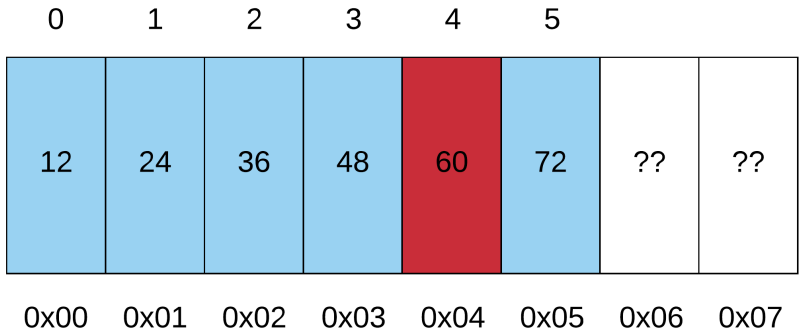
Listas lineares *(arrays)*:

› Remoção de elementos mantendo ordenação: **batatinha**



Fundamentos de estruturas de dados

Listas lineares *(arrays)*:

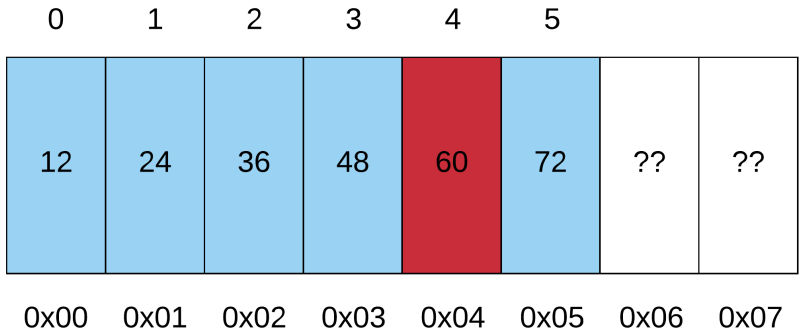


› Algoritmo:

1. Achar o elemento a ser removido
2. Deslocar os elementos posteriores para a esquerda

Fundamentos de estruturas de dados

Listas lineares *(arrays):*



1. Achar o elemento a ser removido:

elemento = 60;

posição = 0;

Enquanto (batatinha[posição] != elemento) e posição < tamanho

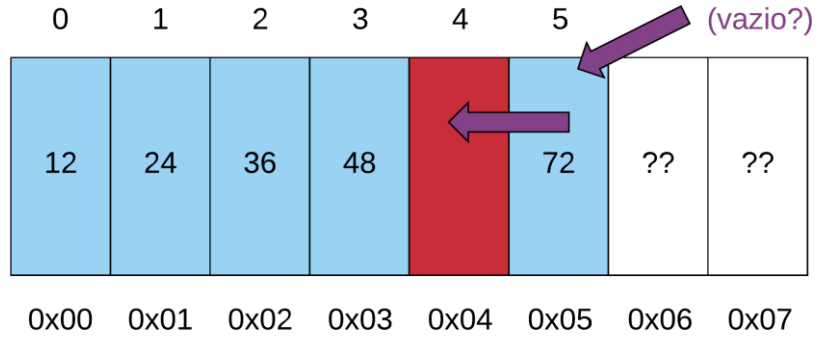
    posição = posição + 1;

Fim do Enquanto

Retorna posição;

Fundamentos de estruturas de dados

Listas lineares *(arrays):*



1. Deslocar os outros elementos:

posição = 4;

Enquanto posição < tamanho

batatinha[posição] = batatinha[posição + 1];

posição = posição + 1;

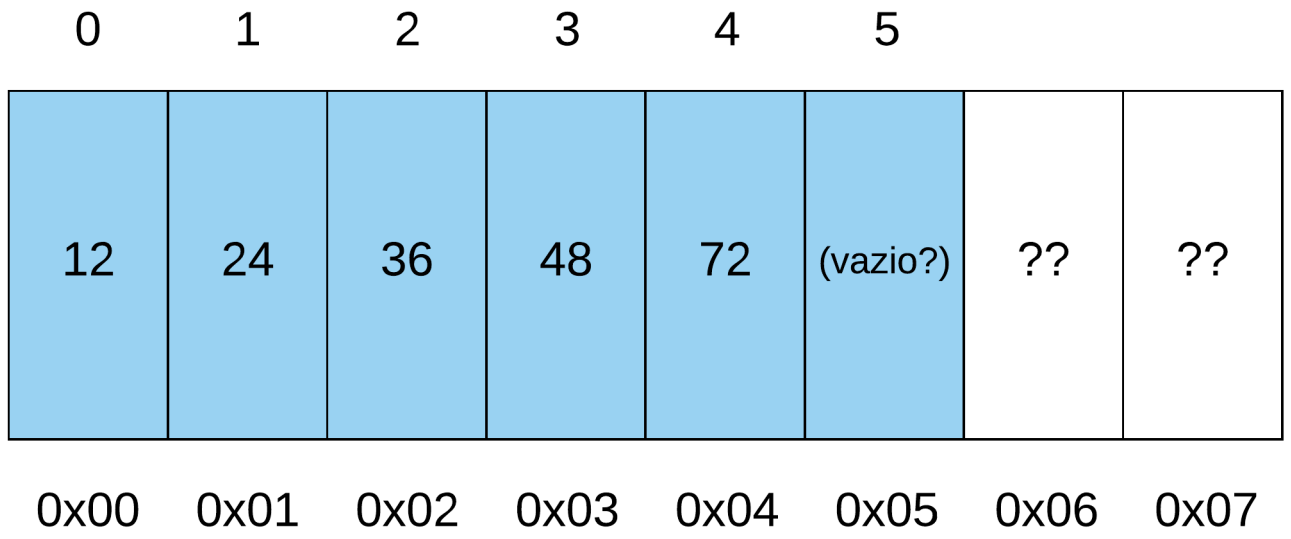
Fim do Enquanto

batatinha[posição] = (vazio?);

Fundamentos de estruturas de dados

Listas lineares *(arrays)*:

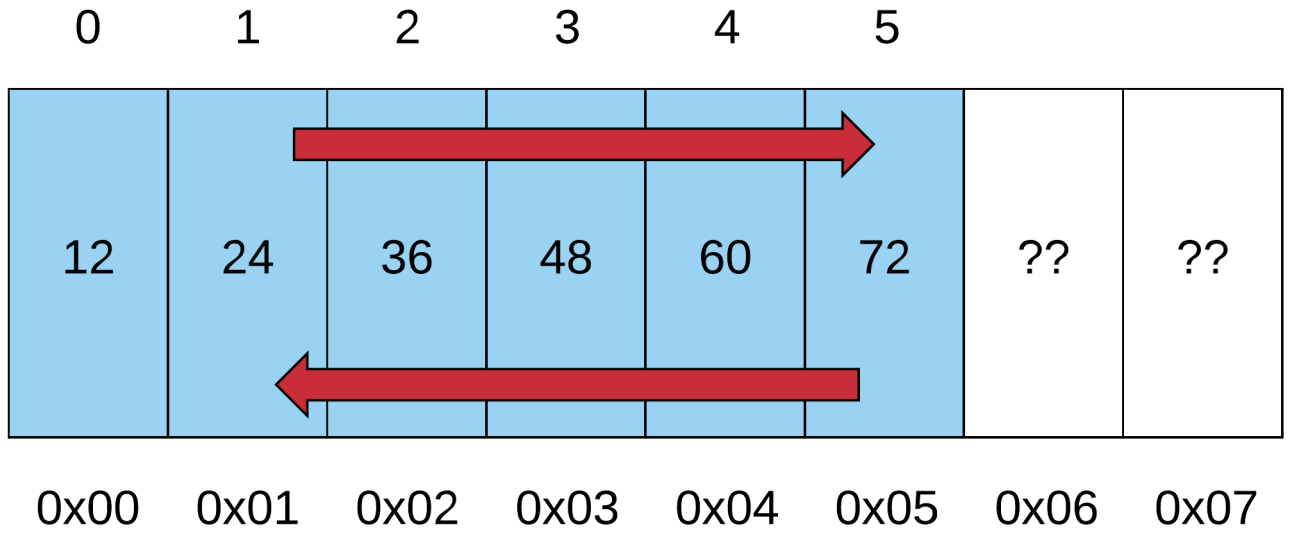
› Remoção de elementos mantendo ordenação: **batatinha**



Fundamentos de estruturas de dados

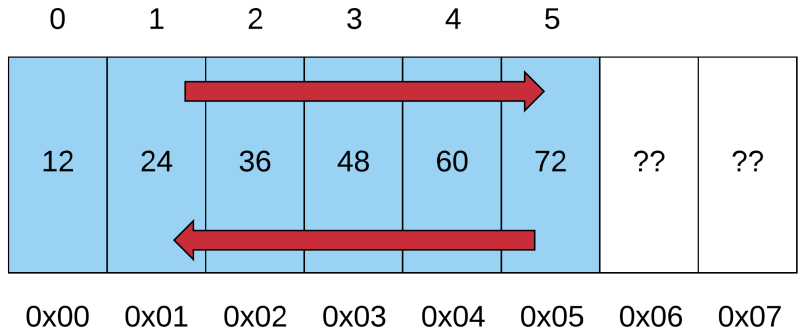
Listas lineares *(arrays)*:

› Trocar elementos de posição: **batatinha**



Fundamentos de estruturas de dados

Listas lineares *(arrays)*:



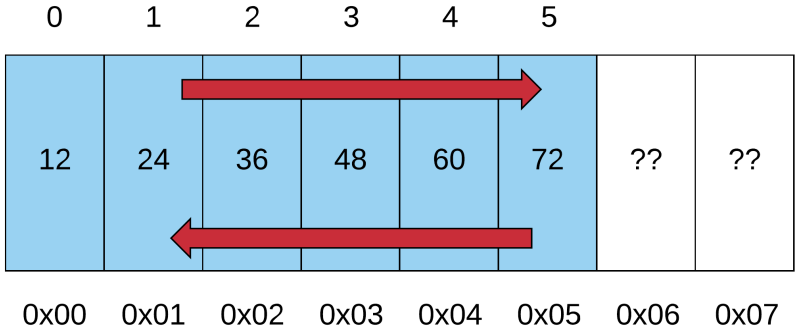
› Algoritmo:

1. Achar a posição do primeiro elemento
2. Achar a posição do segundo elemento
3. Trocar os elementos de lugar

Fundamentos de estruturas de dados

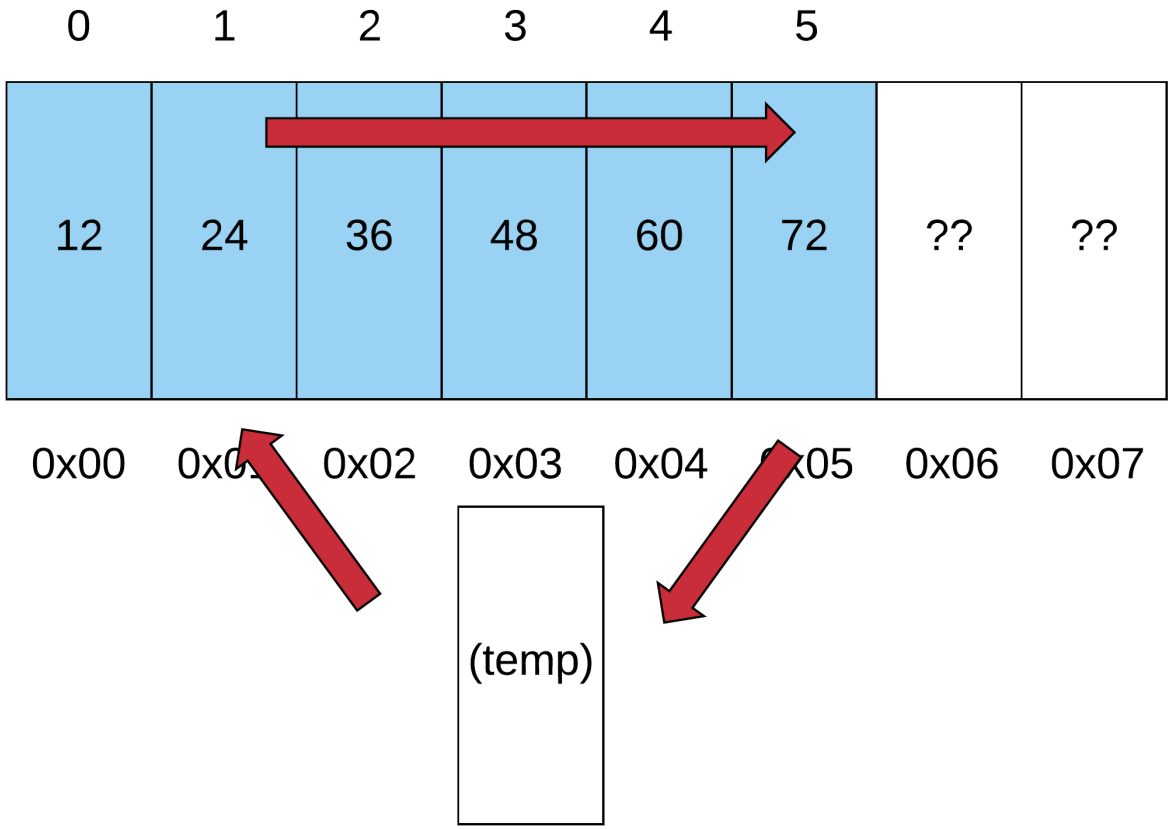
- Achando as posições:

```
posição24 = -1;
posição72 = -1;
De i = 0 até i = tamanho-1
        se batatinha[i]  = 24
                posição24 = i;
        se batatinha[i]  = 72
                posição72 = i;
        i = i + 1;
Fim
Retorna(posição24, posição72);
```



Fundamentos de estruturas de dados

› Trocando os elementos:



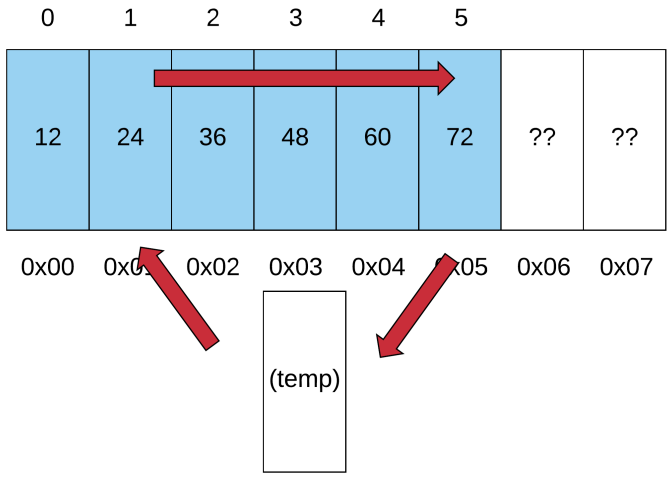
Fundamentos de estruturas de dados

› Trocando os elementos:

temp = batatinha[posição72];

batatinha[posição72] = batatinha[posição24];

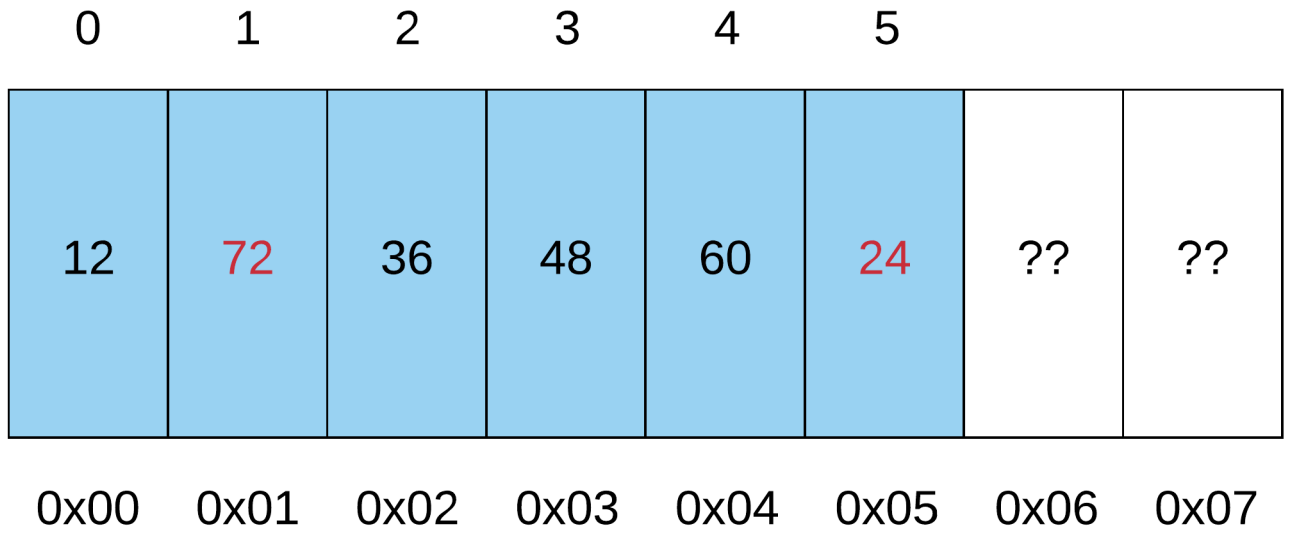
batatinha[posição24] = temp;



Fundamentos de estruturas de dados

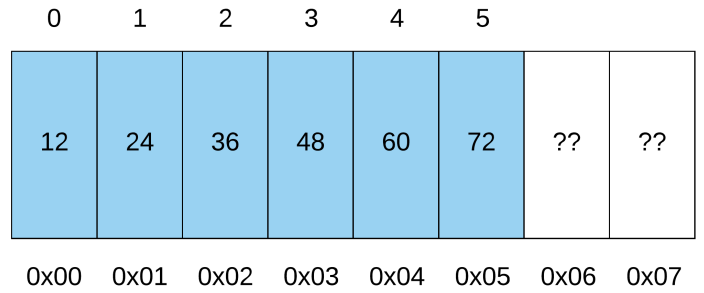
Listas lineares *(arrays)*:

› Trocar elementos de posição: **batatinha**



Fundamentos de estruturas de dados

Listas lineares *(arrays)*:

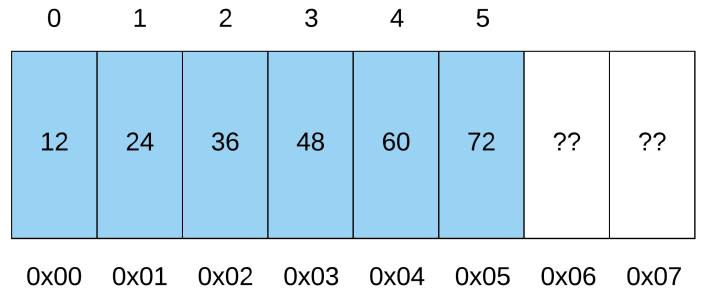


› Voltando às considerações:

- Operações que envolvem ordenação custam caro
- Operações que envolvem busca também custam caro

Fundamentos de estruturas de dados

Listas lineares *(arrays):*

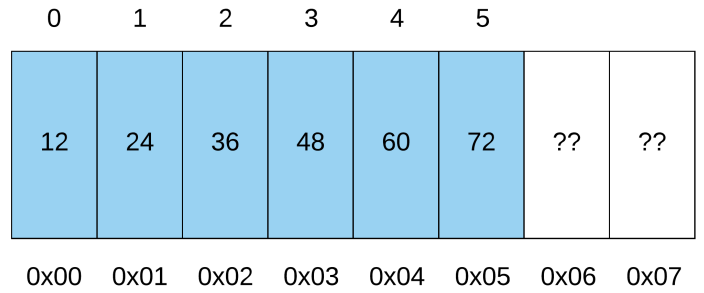


› Voltando às considerações:

– ...então, pra que serve um *array* ?

Fundamentos de estruturas de dados

Listas lineares *(arrays):*



› Vantagens:

- Acesso aos elementos via posicionamento é rápido
- Ao ocupar um espaço fixo na memória, maximiza eficiência de operações que envolvem *caching* (exemplo: multiplicação de matrizes)