# 一、数组 Array

• **常见写法**: Java, C++: int a[100];　　可泛型

Python: list = []

JavaScript: let x = [1,2,3]

• **硬件实现**: 内存管理器 (Memory Controller)

申请数组 → 开辟连续的地址，每个地址可直接通过内存管理器访问.

∴ 访问哪个元素时间复杂度一样: O(1)　快

但，增删 慢　O(n)

• **插入**:

**删除**: 把删的那个元素设置为空，唤起Java垃圾回收机制即可.

or -size

**源码**: ArrayList:

```
/**
 * Appends the supplied element to the end of this list.
 * The element, e, can be an object of any type or null.
 *
 * @param e the element to be appended to this list
 * @return true, the add will always succeed
 */
public boolean add(E e)          加到数组末尾
{
  modCount++;
  if (size == data.length)
    ensureCapacity(size + 1);    → 保证size
  data[size++] = e;
  return true;
}

/**
 * Adds the supplied element at the specified index, shifting all
 * elements currently at that index or higher one to the right.
 * The element, e, can be an object of any type or null.
 *
 * @param index the index at which the element is being added
 * @param e the item being added
 * @throws IndexOutOfBoundsException if index &lt; 0 || index &gt; size()
 */
public void add(int index, E e)
{
  checkBoundInclusive(index);    → 检查上下界
  modCount++;                    → 标记操作次数
  if (size == data.length)
    ensureCapacity(size + 1);
  if (index != size)             拷贝                        需挪动的长度
    System.arraycopy(data, index, data, index + 1, size - index);
  data[index] = e;               原数组与起点位置    目标     长度
  size++;
}
```

```java
/**
 * Guarantees that this list will have at least enough capacity to
 * hold minCapacity elements. This implementation will grow the list to
 * max(current * 2, minCapacity) if (minCapacity &gt; current). The JCL says
 * explictly that "this method increases its capacity to minCap", while
 * the JDK 1.3 online docs specify that the list will grow to at least the
 * size specified.
 *
 * @param minCapacity the minimum guaranteed capacity
 */
public void ensureCapacity(int minCapacity)
{
  int current = data.length;

  if (minCapacity > current)
    {
      E[] newData = (E[]) new Object[Math.max(current * 2, minCapacity)];
      System.arraycopy(data, 0, newData, 0, size);
      data = newData;
    }
}
```

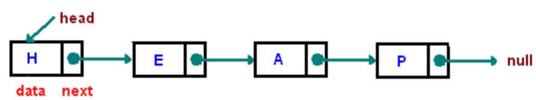
保证数组长度

直接new一个，长度×2，暴力美学

老数组直接copy

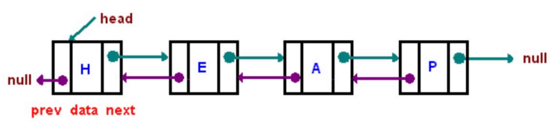
## 二、链表 Linked List

- 单链表




节点 class: Node, Value也可为类

- 双向链表




Node的简单实现:

```java
private static class Node<AnyType>
{
   private AnyType data;
   private Node<AnyType> next;

   public Node(AnyType data, Node<AnyType> next)
   {
      this.data = data;
      this.next = next;
   }
}
```

泛型

- 源码

节点类:

```java
/**
 * Class to represent an entry in the list. Holds a single element.
 */
private static final class Entry<T>
{
  /** The element in the list. */
  T data;

  /** The next list entry, null if this is last. */
  Entry<T> next;

  /** The previous list entry, null if this is first. */
  Entry<T> previous;

  /**
   * Construct an entry.
   * @param data the list element
   */
  Entry(T data)
  {
    this.data = data;
  }
} // class Entry
```

双向

成员变量:

```java
/**
 * Compatible with JDK 1.2.
 */
private static final long serialVersionUID = 876323262645176354L;

/**
 * The first element in the list.
 */
transient Entry<T> first;

/**
 * The last element in the list.
 */
transient Entry<T> last;

/**
 * The current length of the list.
 */
transient int size = 0;
```

- 时间复杂度:

| | |
|---|---|
| prepend (加在最前) | O(1) |
| append (加在最后) | O(1) |
| lookup (访问) | O(n) |
| insert | O(1) |
| delect | O(1) |

## 三、跳表 skip list

1、特点：① 1990年左右诞生，比平衡树(AVL)晚，但打败AVL和二分查找.

② 只用于元素有序

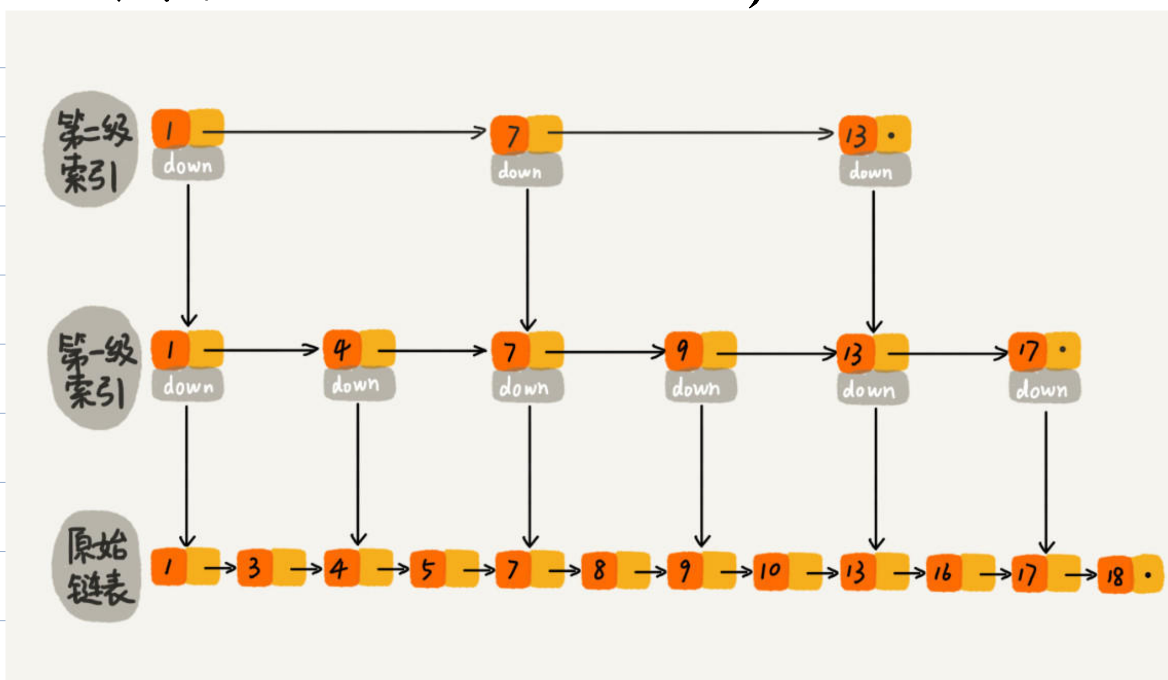
③ 插/删/查： O(log n)

④ 原理简单，易实现，方便扩展，效率高.

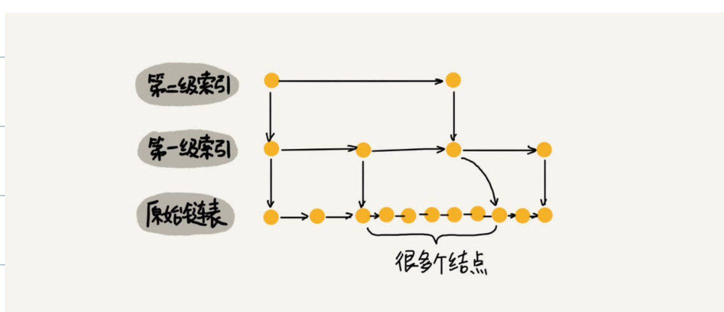
在 Redis、LevelDB 等中用来代替 AVL Tree.

2、实现：

(将链表的O(n)提速： 升维，空间换时间)




3、现实使用时，




维护成本高

4、空间复杂度 $O(n)$

## 四、工程中应用

eg. LRU Cache → Linked list (双链表) 力扣146

Redis → Skip list