Lists are Abstract Data Type (ADT)

- A "container" for holding data
- Memory vs time needed to search vs time to add/remove
- Array is a kind of container too
  - predefined size
- Lists let size grow/shrink easily
  - Spend more time searching (potentially)

![Diagram of linked list nodes: head -> node (content: object) -> node (null, last: optional)]

```
```

Search: for some content
- return true/false
- or return reference to node where content is
  - null if not there

Search is linear (element by element) => O(n)
Sorted array:

1 2 3 4

binary search

index in the middle

Compute memory address of a[i]
Name of array a gives starting address

a -> [ ]

i * length of type of elements
Computing where a[i] fits for any i is O(1)!

Arrays are random access (supports some efficient algorithms like binary search)
List: memory gets allocated as we go!

head

Memory heap

Insert a new element (e.g., 2)
Must follow the link to next element
No way to compute in $O(1)$ where an element is sequential access worst-case, operations will be $O(n)$
Can't do binary search, we need linear search.
Lists are good if we do a lot of insert/delete not a lot of search.

E.g. Insertion is $O(1)$ if inserting first
$O(n)$ if we insert last
package SingleLinked Lists;  // at start of each Java class in the package

class Node {
  Object content;
  Node next;
}

public Node (Object c) { // so public in package, private from outside
  content = c;
  next = null; // nothing after
}

public Object getContent () {
  return content;
}

// may be setContent

class Node<T> {  // Generic Types
  T content;
  Object -> T

  Node<Integer> n = new Node<Integer>(new Integer(5));
package SingleLinkedList;

public class LinkedList {

    Node head;

    public LinkedList() {
        head = null;
    }

    public boolean isEmpty() {
        return (head == null);
    }

    // insert into a list → first position
    public void insertFirst(Object c) {
        Node n = new Node(c);
        n.next = head;
        head = n;
    }

    // O(1)!
}
// Insert in last position in a list

head

Find the right spot!

Iterate until we are on last node; then link
If list is empty \( \rightarrow \) modify head

```java
public void insertLast(Object c) {
    Node n = new Node(c);
    if (isEmpty()) {
        head = n;
    } else {
        Node curr = head;
        while (curr.next != null) {
            curr = curr.next;
        }
        curr.next = n;
    }
    return;
}
```

\[ O(n) \] List Type + ListIterator Type