COMP 250 - Lecture 19

Lists → Abstract Data Type (ADT)

A "container" for holding data

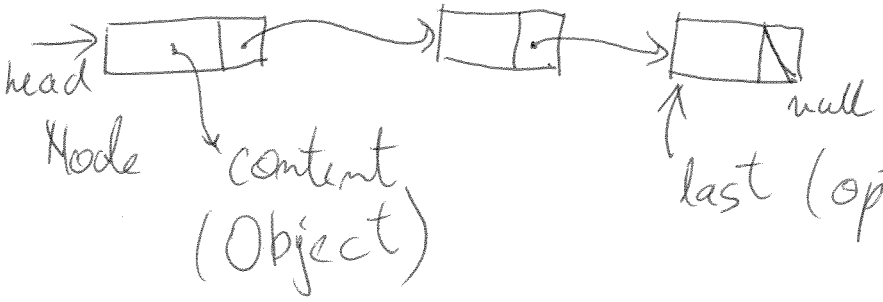
Memory vs time needed to search vs time to add / remove

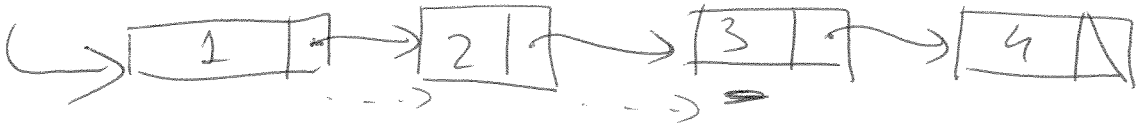
Array is a kind of container too

↳ predefined size

Lists → size can grow / shrink easily

Spend more time searching (potentially)





search: for some content

return true / false

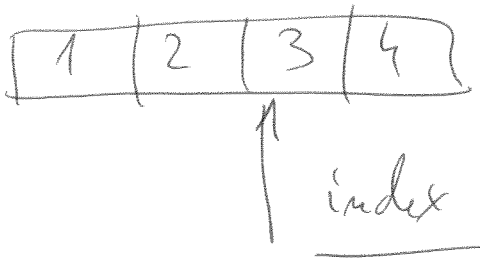
or return reference to Node where content is,

null if not there.

search (3)

Search is linear (element by element) $\Rightarrow O(n)$

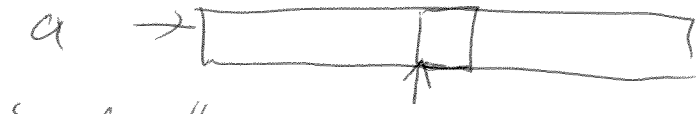
Sorted array:



binary search

Compute memory address of a[i]

name of array a gives starting address

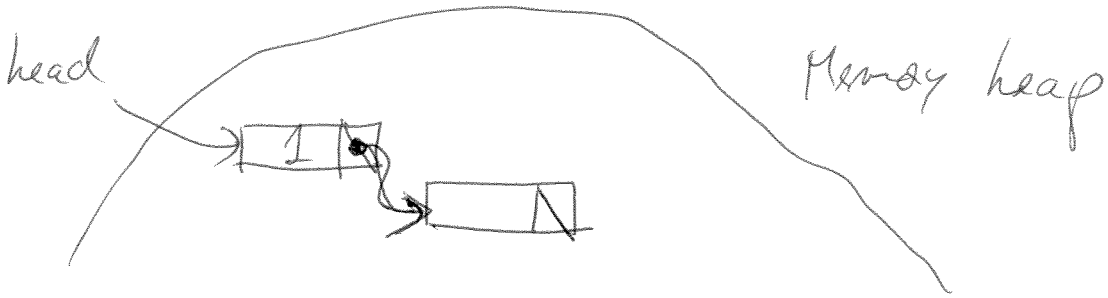


i * length of type of elements

Computing where a[i] is for any i is O(1)!

Arrays are random access (supports space efficient algs like binary search)

List: memory gets allocated as we go!



Insert a new element (eg 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 SingleLinkedLists;
```
← at start of each Java class in the package

```
class Node {
  Object content;
  Node next;
  public Node(Object c){
    content = c;
    next = null; // nothing after
  }
  public Object getContent(){
      return content;
  }
  // may be setContent
}
```

Node:

content

Neither private nor public so public in package, private from outside

Generic Types

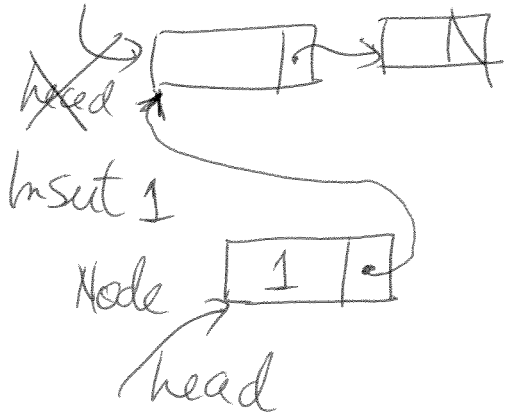
Type variable, will be specified later

```
class Node<T> {
   T content;
     Object → T
```

```
Node<Integer> n = new Node<Integer>(new Integer(5));
```

```
package SingleLinkedLists;
public class LinkedList {
    // Reference to 1st element
    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)!



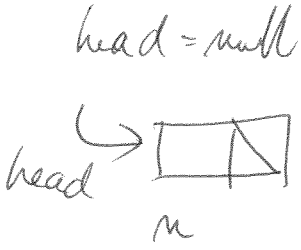
// insert in Last position in a list

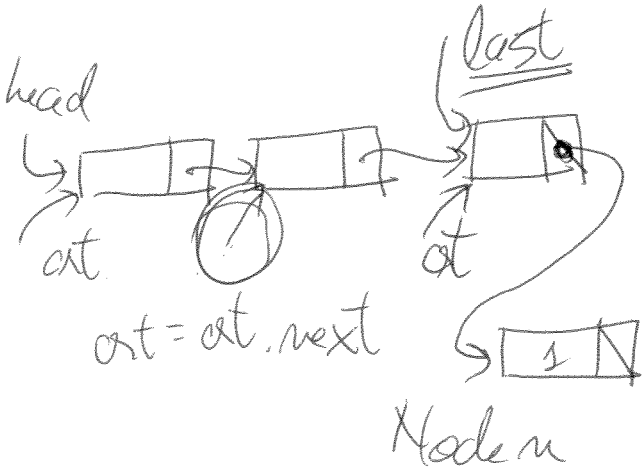
Find the right spot!

Iterate until we are on last node; then link

If list is empty → modify head

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





crt is an Iterator

List Type + ListIterator Type

O(n)!