Example: insertion-sort algorithm

A = 5 9 17 2 3

Pseudocode:
1) for i = 1 to n-1 do
2) insert A[i+1] at the right position among the sorted (increasingly) elements A[1...i]
(results in A[1...i+1] sorted)

n = length of the array
Pseudocode

Example: insertion-sort algorithm

High-level pseudocode:

\[ n = \text{length of the array} \]

\[ \text{pseudocode:} \]

1) for \( i = 0 \) to \( n-1 \) do \{ 
    2) insert \( A[i+1] \) at the right position among the sorted (increasingly) elements \( A[1...i] \) 
    (results in \( A[1...i+1] \) sorted) 
\} 

A more detailed pseudocode:

1) for \( i=0 \) to \( n-2 \) do \{ 
2) \( j = i \) (\( j \geq 0 \) and \( j < i \)) 
3) while \( (A[j+1] < A[j]) \) \{ 
4) \( \text{switch } A[j] \text{ with } A[j+1] \) 
5) \( j-- \) 
6) \} 

Pseudocode

Example: insertion-sort algorithm

More detailed pseudocode: ✗ done 😊
Pseudocode

Example: insertion-sort algorithm

Java-code: see the code
Pseudocode

- What is a pseudocode?
  - a programming-language-independent description of an algorithm
  - tends to follow typical conventions like indentation for inner parts of a loop etc.
  - different levels of detail, depending on needs

- Why do we use it?
  - any programmer can implement in the language of their choice
  - very helpful conceptually before starting coding
Singly linked lists

What are they?

points to the "car" - the object containing info + pointer
Singly linked lists

Which classes and methods do we need?

Node - the "car"

variables

- int info
- Node next

+ get/set methods

we did this on the whiteboard,
see the Java-code posted on the
course website

List

variables

- Node first
- int size

methods

- add methods
- remove methods
- size
- sorting
Singly linked lists

Inserting an element
- at the beginning

Algorithm: `insertElemBeg`

Input: an element (int `newElem`)

Output: none but the element is inserted at the beginning

Pseudocode:
1) create a new Node with `newElem` and pointing to `first`
2) increase size by 1
3) update `first`

Java:
```java
Node tmp = new Node(newElem, first);
size++; 
first = tmp;
```
Singly linked lists

Inserting an element
- at the end

**Pseudocode**

**High-level:**
1) start at the first,
   then hop through the next links
to find the Node with next == null 
   \( \leftarrow \) let it be last
2) create Node with newElem and next = null
3) increment size by 1
4) set the next variable of last to X

\[\begin{align*}
\text{have to check if list empty} &< \\
0) & \text{if (size = 0) then } \\
& \quad \text{first = new Node (newElem, null);}
& \quad \text{size++;}
\end{align*}\]

**Detailed pseudocode**

1) Node current = first;
   while (getNext() != null) {
      current = current . getNext();
   }
   \( \leftarrow \) now current = the last Node
2) Node X = new Node 
   (newElem, null);
3) size++;
4) current . setNext (X);
Deleting an element
- at the beginning

```java
if (size > 0) {
    first = first.getNext();
    size--;
} else {
    throw an exception
}
```
Singly linked lists

Deleting an element - at the end?

- Second to get the last element and set its `next = null` size--

```
idea 1: loop through size-1 guys careful with size=1
idea 2: or do the while loop with current but check current.getNext().getNext()
idea 3: check current.get_next() == last (if we are keeping track of it)
```
**Doubly linked lists**

**Picture:**

```
null ← 5 ← 5 ← 17 ← null
null ← 75 ← 91 ← 14 ← null
```

What classes do we need?

```java
public class DLNode {
    DLNode next, prev;
    int elem;
}
```

```java
public class DLinkedList {
    DLNode first, last;
    int size;  // maybe
}
```
Doubly linked lists

Insert an element
- at the head
1) create a node with 8
   with prev = null, next = first
2) set the prev of first to X
3) update first
4) size++

- at the tail
same but backwards - try on your own

DNode X = new DNode (newElem, null, first)
first. setPrev(X); ← do only if size>0
first = X
size++

what if
size=0
first = last = null
Doubly linked lists

Delete an element

- the head

```
if size > 1 then {
    first getNext().setPrev(null)
    first = first getNext();
    size --
} else {
    if size == 0 then throw an exception
    else { // size == 1
        first = null
        size --
        last = null
    }
}
```

- the tail

Same but backwards
Insert an element in the middle

Input: newElem
where (Node) ← meaning we are inserting newElem before where

Pseudocode:

```java
DLNode Y = new DLNode (newElem, where.getPrev(), where);
where.setPrev(Y);
Y.getPrev().setNext(Y); ← only if Y.getPrev() ≠ null (or if where ≠ first)
size++
if where == first then first = Y
```
Doubly linked lists

Delete an element in the middle

Input: DLNode deleteMe

Pseudocode:

```java
deleteMe.peek().setNext() = deleteMe.peekNext();
deleteMe.peekNext().setPrev() = deleteMe.peekPrev();
size--; // problem! if deleteMe is the first or the last then what?
```

Think about it.
Stacks

Last-in-first-out (LIFO)

Which methods might be useful?

boolean - isEmpty();
            isFull();
            push(newElem); - for inserting an elem.
            pop(); - for getting the top of the stack out
                      - returns an elem.
Stacks

How to implement?

```java
public class Stack {
    int size, maxsize;
}
```

Possibilities:
- Array
  - Only if implementing with arrays
- Linked list

Switch to Emacs
The first application: matching parentheses (or tags in html)

ignore everything in the input except parentheses, suppose we allow (){}
check if a given input is well-parenthesized

Example:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(())(){}</td>
<td>OK YES</td>
</tr>
<tr>
<td>{}</td>
<td>NO</td>
</tr>
<tr>
<td>()(</td>
<td>NO</td>
</tr>
</tbody>
</table>

input: a string of parentheses
output: YES if ok parenthesized, NO otherwise

idea: use a stack to push left parentheses
if we see a right par. then check if it matches the top of the stack

empty stack

done OK
The first application: matching parentheses (or tags in html)

Pseudocode:

Input: a string of parentheses
Output: YES/NO if well-parenthesized

1) create an empty stack
2) for i = 0 to length of s-1 do {
   3) let c be the parenthesis at position i → assume O(1)
   4) if c is ( or & then → O(1)
      push c onto the stack → O(1) bec. adding an element to the front of a linked list
   5) else {
      6) if the stack is empty then return NO → O(1) e.g. )
      7) let d be the symbol popped from the stack → O(1) like deleting from the end
      8) if c and d do not match then return NO → O(1) e.g. { }
   9) }
10) }
11) }
12) if stack is empty then return YES
13) return NO

Running time:

LINEAR is the fastest possible
Queues

First-in-first-out (FIFO)

Which methods might be useful?

possible implementations

linked list
array
Queues

How to implement?
Pseudocode for implementation via circular arrays

`9 17 23 8`

**shift**  — but circular arrays better

`9 17 23 8`  ...

check the implementation in the book
A little bit about time and **space** requirements:

So far, we discussed running times

- e.g. adding/deleting element from the beg. of a linked list: $O(1)$
  - (also push/pop from stack)

For queues:
- e.g. if implementing with singly linked lists
  - need to insert at the end, remove from the beginning: $O(1)$
  - if we are keeping track of last: $O(1)$

Space: $O(n)$ where $n$ is the number of elements

---

**Note:**

Other variants exist: circular lists, double-ended queues, etc.