ADT : List Insert, Delete, Find

Implementation as a linked list

	Stack	Queue
Insert	Push	Enqueue
Delete	Pop	Dequeue

Delete NEWEST element inserted

LIFO

Delete OLDEST element inserted

FIFO

Push : Insert element at "top" of stack

Pop : Delete (get) element at "top" of stack

Peek : Get element at "top" without deleting

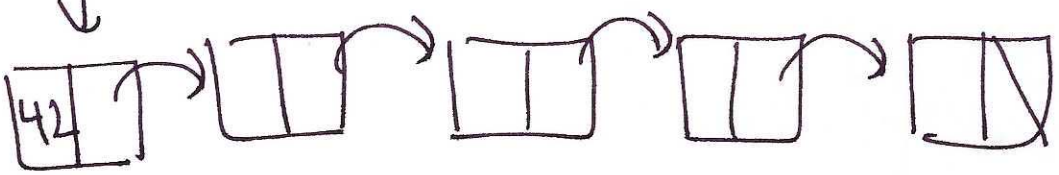
Enqueue : Insert at "end" of queue

Dequeue : Delete at "beginning" of queue

Stack can be implemented as a linked list

push: insert into linked list

(insert adds element at the head)



push(42)

pop: delete head of linked list, return old head

peek: return head

Checking parentheses

{ ( ) [( )] } { ( ] [ ) }

Correct Wrong

<HEAD> </HEAD> <BODY> </BODY>

  - -

Suppose all parentheses were of same type

(())(())(()()) (((()))).

A

)((())(

X

---

count = 0

for i = 0 to length(A) - 1

{

if A[i] is opening, count++

if A[i] is closing, count --

if count < 0, output WRONG

}

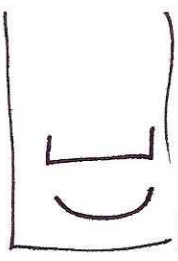
if count > 0, output WRONG

else output RIGHT

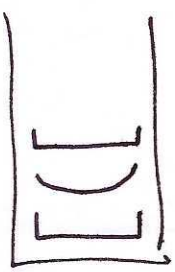
What about multiple paretheses types?

First guess: Maintain a counter for each different type.

([)]

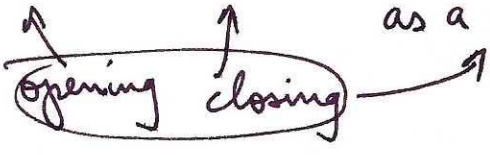


[([])]



Start with empty stack S

```
for i=0 to length(A) - 1
{
    if A[i] is open, S.push(A[i])
    if A[i] is closing
    {
        c = S.pop()
        if c and A[i] DON'T match, output WRONG
    }
}
```




```
if S is empty, output RIGHT
else output WRONG
```

Stacks and recursion

Any code that uses recursion can be converted to "non-recursive" code that uses a stack.

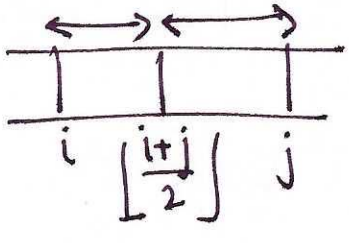
Stacks can be used to simulate recursion.

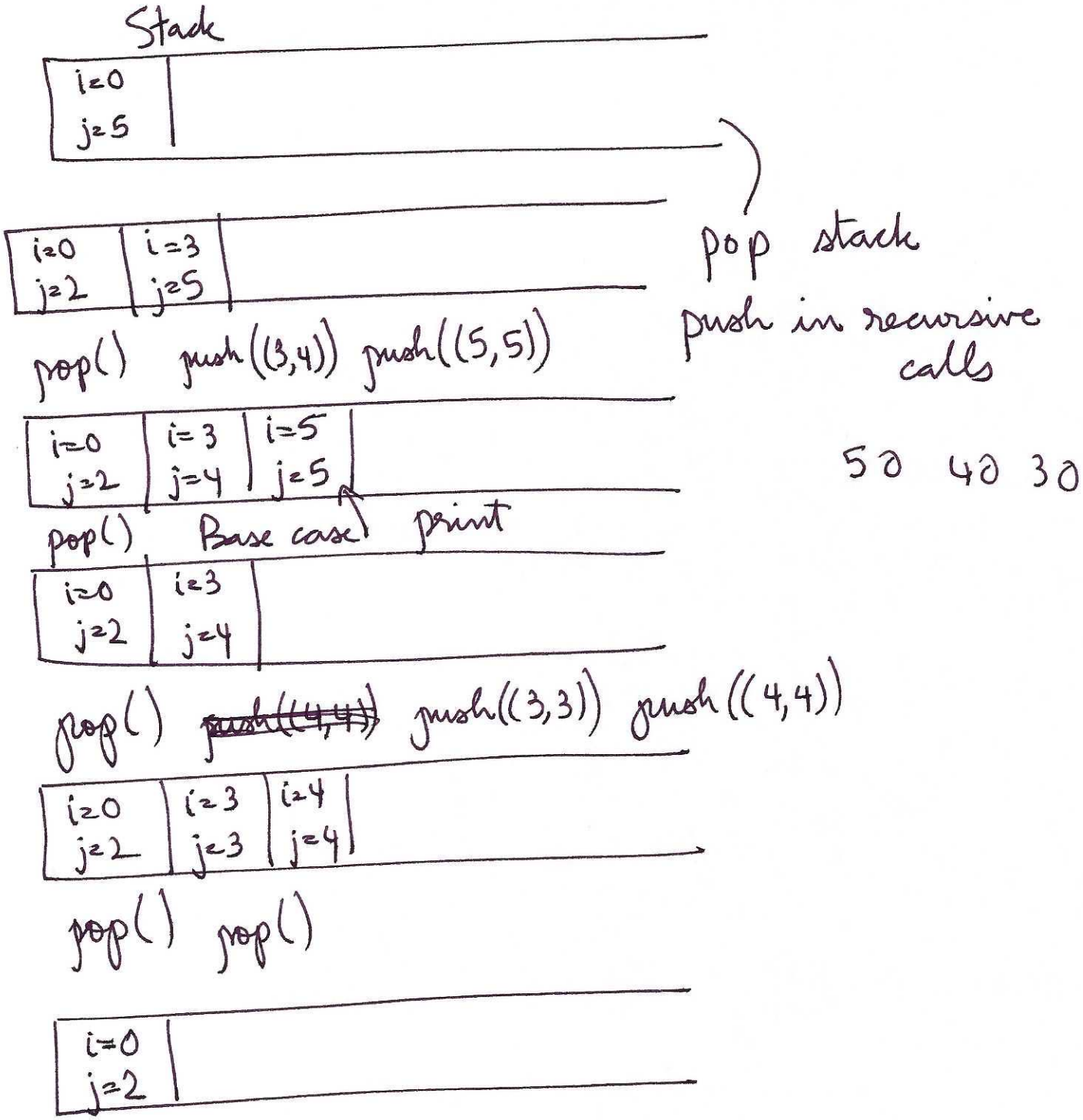
(Recursion is implemented thru stacks.)

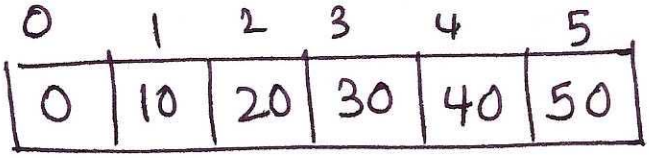
```
void printReverse(int[] A)

void printReverse(int[] A, int i, int j)
{
    if (i > j) return;
    if (i == j) {print(A[i]); return;} //base case
    printReverse(A, ⌊(i+j)/2⌋ + 1, j);
    printReverse(A, i, ⌊(i+j)/2⌋);
    return;
}
printReverse(A, 0, length(A) - 1)
```

indices in array




Stack
i=0
j=5
pop stack
push in recursive calls
i=0
j=2
i=3
j=5
pop()
push((3,4))
push((5,5))
i=0
j=2
i=3
j=4
i=5
j=5
50 40 30
pop()
Base case
print
i=0
j=2
i=3
j=4
pop()
push((4,4))
push((3,3))
push((4,4))
i=0
j=2
i=3
j=3
i=4
j=4
pop()
pop()
i=0
j=2



50 40 30 20 10 0

printRev(A, 0, 5)

Printing

50 40 30 20 10 0

printRev(A, 3, 5)

printRev(A, 0, 2)

printRev(A, 0, 1)

printRev(A, 5, 5)

printRev(A, 3, 4)

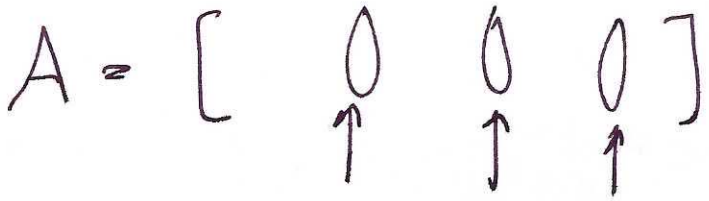
printRev(A, 2, 2)

printRev(A, 4, 4)

printRev(A, 3, 3)

```
void printRev(int A[], ~~int i, int j~~ int length)
{
    Stack s;  // stack of ~~pairs~~ pairs
    ~~int[]~~
     pair top;
     pair init;
     init = (0, length-1);
    s.push(init);
    while(s is not empty)
    {
        top = s.pop()
        ~~if~~ int i = top.first
          int j = top.second
          if (i==j) { print A[i]; return }
          s.push((i, ⌊(i+j)/2⌋))
          s.push((⌊(i+j)/2⌋+1, j))
    }
}
```

Printing all subarrays



A subarray is an array with a subset of elements (in same order)

Q. If A has length n, how many subarrays does it have?

(R) $2n$ (G) $n!$ ~~(B)~~ 2^n

[7 101 42]

3

Write a program that prints all subarrays.

~~[]~~

[7]

[7 101]

[7 42]

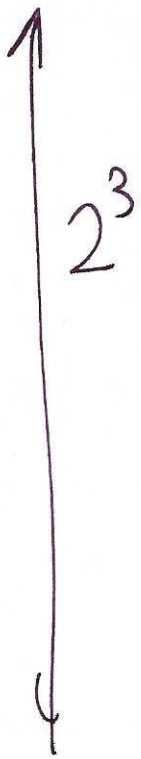
[7 101 42]

[101]

[101 42]

[42]

[]



2^3

~~Set of~~

Family of all subarrays can be partitioned into two subfamilies

(1) Subarrays of A[0...length-2]

(2) Subarrays that have element A[length-1]

[7 101 - - - 42]