COL106 - Data Structures and Algorithms

1

Basic Data Structures

LIST ADT

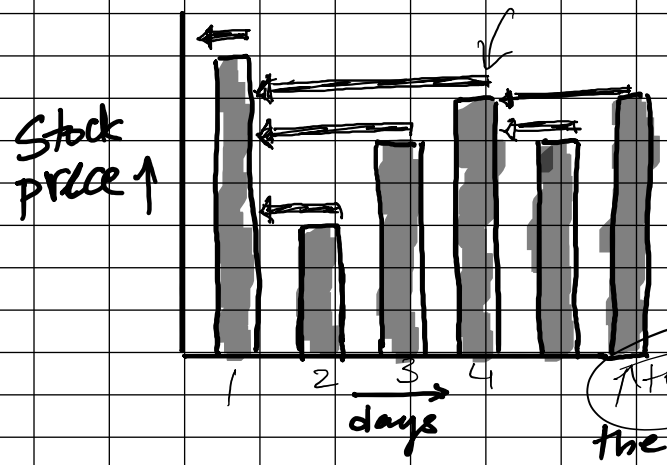
- Arrays
  - Dynamic Arrays
- Linked List
  - Singly Linked List
  - Doubly Linked List

Specialized ADTs

- Stacks (~~FIFO~~) LIFO
- Queues (~~LIFO~~) FIFO

2

Consider Stock Trading

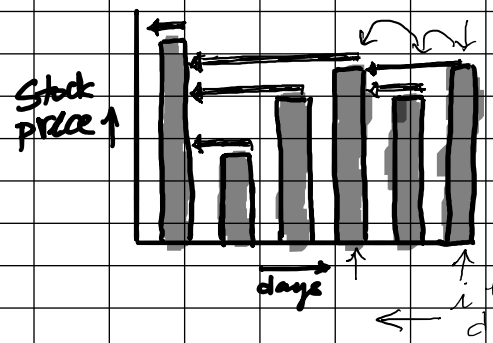


We are interested in knowing for each day, what are the no. of consecutive sequence of previous days when the stock price is less than that days.

3

$O(n^3)$

Consider Stock Trading



Let's say we are on i-th day.

⊛ Starting from the previous day keep increasing the window of time past as long as the stock price is lower.

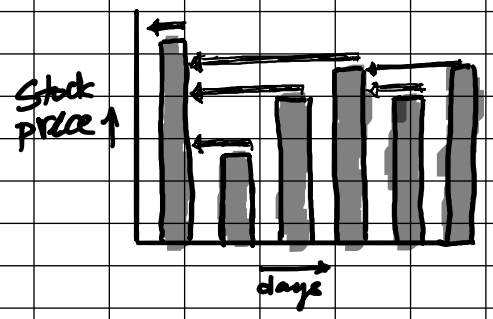
loop runs for (i-1) times

```
i-th day.
let S ← 1
for ((S ≤ i) && StockPrice[i-S] ≤ StockPrice[i])
    S ← S+1
Span[i] ← S
```

ith day span is stored in array Span.

4

Consider Stock Trading



Let's say we are on i-th day.

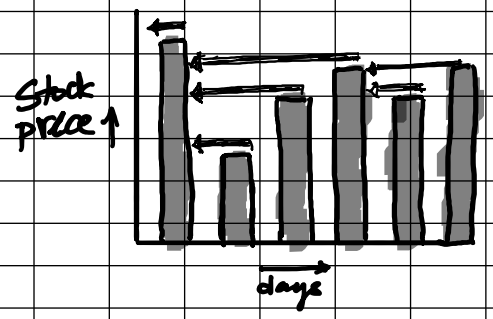
* Starting from the previous day keep increasing the window of time past as long as the stock price is lower.

Over all values of i ⇒ $O(n^2)$

```
i-th day.
let S ← 1
for (S ≤ i && StockPrice[i-S] ≤ StockPrice[i])
    S ← S+1
Span[i] ← S
```

5

Consider Stock Trading



Let's say we are on i-th day.

* Starting from the previous day keep increasing the window of time past as long as the stock price is lower.

```
i-th day.
let S ← 1
for (S ≤ i && StockPrice[i-S] ≤ StockPrice[i])
    S ← S+1
Span[i] ← S
```

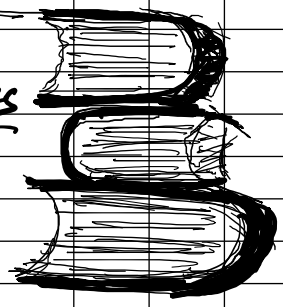
Can we do better?

6

Stack ADT

- Stores Objects
- Insertions & deletions are strictly "Last in - First out"

A stack of books



That book you bought first but too afraid to read :-)

or a stack of plates

7

Stack ADT

- Stores Objects
- Insertions & deletions are strictly "Last in - First out"

Two operations on stacks

push (x) : inserts object x

pop () : removes the last inserted object from the stack and returns it.

8

Stack ADT

- Stores Objects
- Insertions & deletions are strictly "Last in - First out"

Two operations on stacks

push(x) : inserts object x

pop() : removes the last inserted object from the stack and returns it. ← return will in the end

Additionally,

top() : returns the last inserted object

size() :

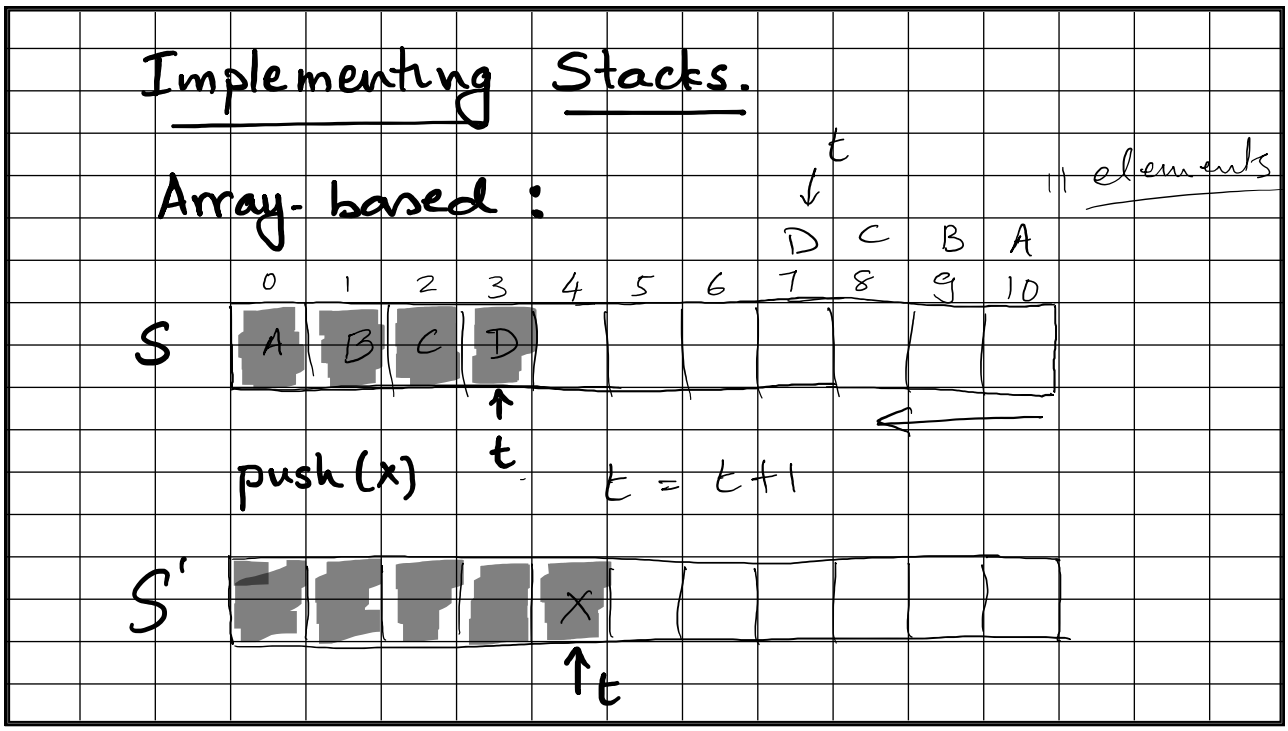
9

Stack Class in Java

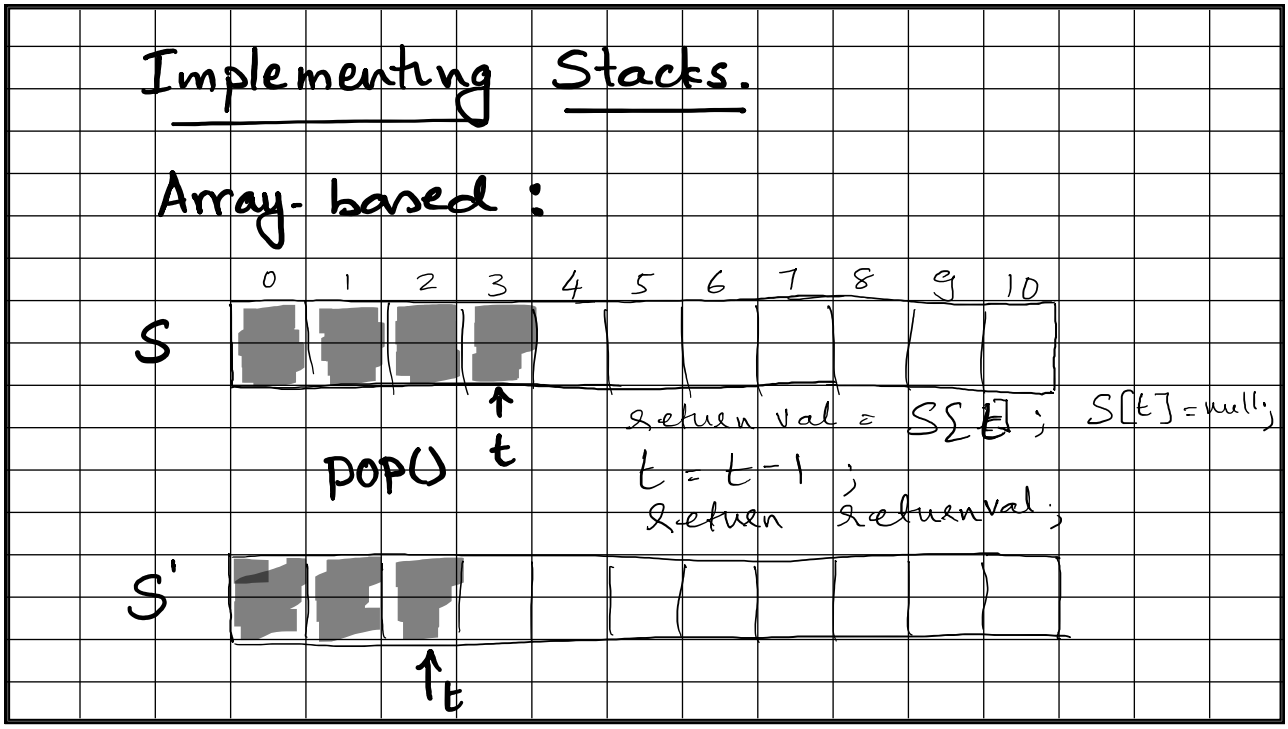
```
public class Stack<E> .... {
    boolean ~~empty()~~;  isEmpty();
    E       peek ();      // same as top()
    E       pop ();
    E       push (E o);
    :
}
```

If the stack is empty, pop() and top() return null.

10


Implementing Stacks.
Array-based:
t
D C B A
11 elements
0 1 2 3 4 5 6 7 8 9 10
S
A B C D
t
push(x)
t = t+1
S'
X
t

11


Implementing Stacks.
Array-based:
0 1 2 3 4 5 6 7 8 9 10
S
t
pop()
return val = S[t]; S[t] = null;
t = t-1;
return returnval;
S'
t

12

Array-based Stack

```
pop() {
    if (isEmpty()) then
        return null;
    t ← t-1;
    return S[t+1];
}
```

13

Array-based Stack

```
pop() {                          // isEmpty(): (t == -1)
    if (isEmpty()) then
        return null;
    t ← t-1;
    return S[t+1];
}
```

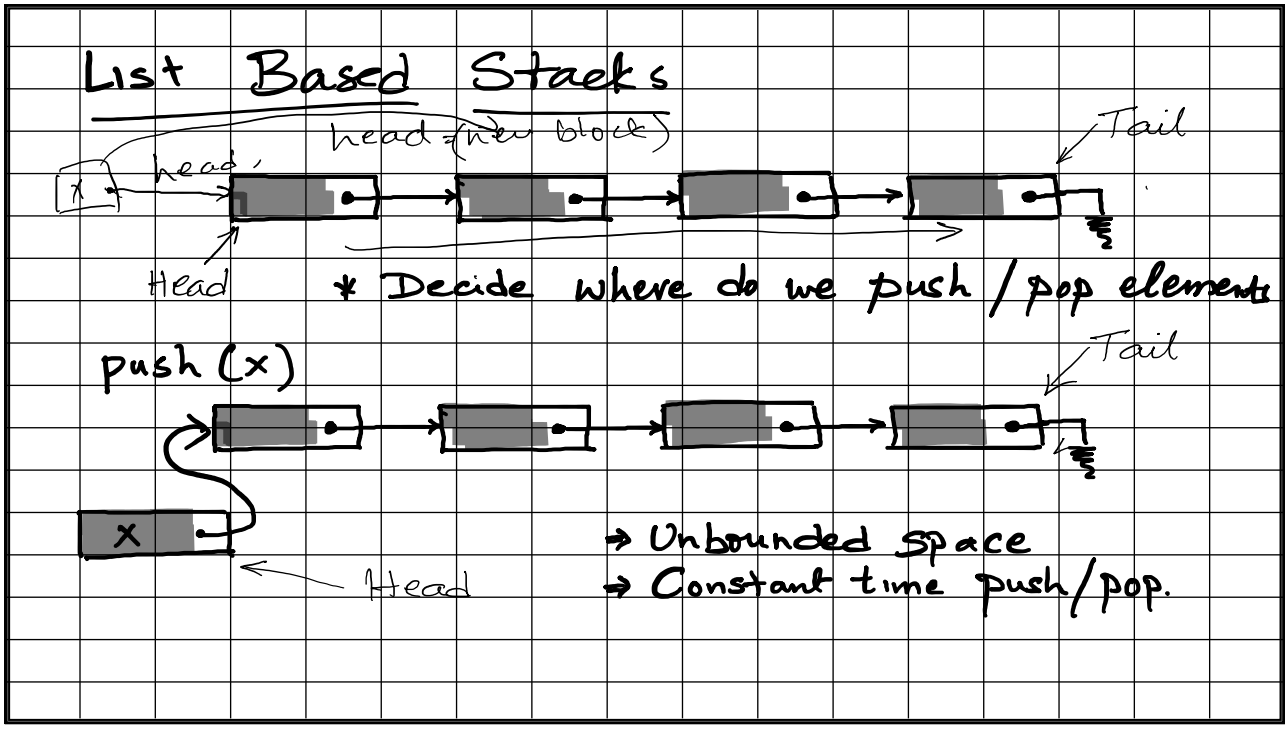
O(1)

```
push(X) {
    if (t == S.length-1)
    then
        throw Exception;
    t ← t+1;
    S[t] ← X;
}
```

O(1)

* Maximum size of the Stack is defined apriori cannot be (easily) changed.

14


List Based Stacks
head (new block)
head
Tail
Head
* Decide where do we push / pop elements
push (x)
Tail
x
Head
→ Unbounded space
→ Constant time push/pop.

15