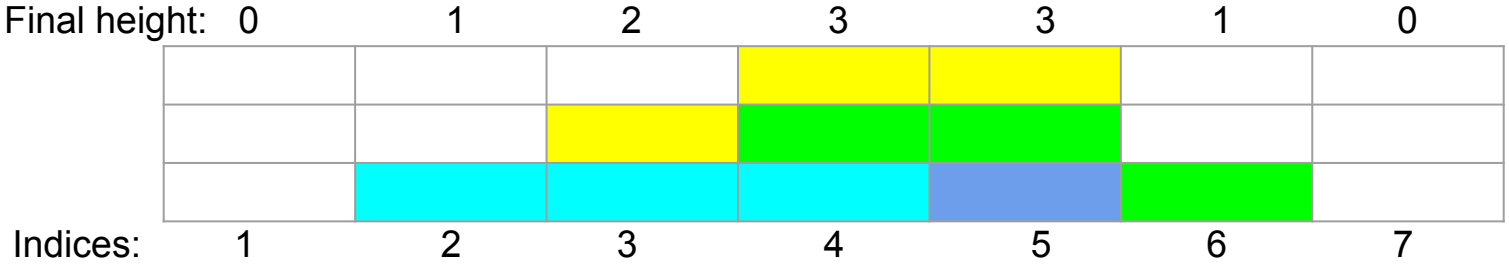
Topic 3: Difference Array, Stacks, and Balanced Parenthesis

Competitive Programming I (Fall 2019)
Ninghui Li (Purdue University)

USACO 2012 January Bronze 2: Haybale Stacking

Starts with N (1 <= N <= 1,000,000, N odd) empty stacks, Bessie is given a sequence of K instructions (1 <= K <= 25,000), each of the form "A B", meaning that Bessie should add one new haybale to the top of each stack in the range A..B. Compute the median height afterwards.

```
SAMPLE INPUT: N=7 K=4        5 5        2 4        4 6        3 5
SAMPLE OUTPUT:   1        (heights after updates 0,0,1,1,2,3,3)
```



Haybale Stacking: Using Difference Array

- Naive approach of maintaining the stack heights has complexity O(N K), which is not efficient enough.
- We need to take advantage of the fact that updates are ranges.
- Instead of directly maintaining the stack heights, we can maintain the differences of adjacent heights.
- Given an array A[0..n-1], its difference array is defined as
  - D[0] = A[0]
  - D[k] = A[k] - A[k-1] when 0<k<n
  - One range update on A (which update all elements in one continuous range by a constant) corresponds to 2 updates on D
- Given the difference array D, reconstructing A takes linear time

Haybale Stacking: In Difference Array

	1	2	3	4	5	6	7
5 5					+1	-1	
2 4		+1			-1		
4 6				+1			-1
3 5			+1			-1	

Final Diff. Array:	0	1	1	1	0	-2	-1
Final height:	0	1	2	3	3	1	0

Kattis: Disastrous Downtime (Required)

Given the starting times of N (1≤N≤100,000) requests, each of which lasts 1000 milliseconds, and K (1≤K≤100,000), the maximum number of requests a server can handle concurrently, compute the minimum number of servers needed to process all requests.

```
Input:    N=6 K=2    1000   1010   1500  1999  2000  2010   2999
Output:   2
Explanation: Request 1 starts at 1000 and ends at 2000.  At time
1999, there are 4 requests.
```

Disastrous Downtime using Difference Array

- We just need to keep track of the active number of requests at any time, which can be changed by two kinds of events: (1) starting of a request (+1 event), and (2) ending of a request (-1 event).
- Sorting events based on time, and we have a difference array.

Time	1000	1010	1500	1999	2000	2010	2500	2999	3000	3010	3999
Diff	+1	+1	+1	+1	-1,+1	-1,+1	-1	-1,+1	-1	-1	-1
Acc	1	2	3	4	4	4	3	3	2	1	0

Disastrous Downtime: The Code

```
int ans = 0, m=0;
int[] t = new int[2*N][2];  // N is number of requests
for (int i=0, j=0; i<N; i++) {
  int  ti = nextInt();   // arrival time of next req
  t[j][0] = ti;       t[j][1] = 1;  j++;   // start
  t[j][0] = ti+1000; t[j][1] = -1; j++;   // ending
}
sort2DInt(t);          // by both dimensions
for (int i=0; i<2*N; i++) {
  m += t[i][1];
  if (m > ans*K)   ans++;
}
return ans;
```

Prefix Sum Array and Difference Array

- Prefix sum array makes answering queries about ranges more efficient.
- Difference array makes updating ranges more efficient.
- Prefix sum array and difference array are dual to each other
- To make both efficient, we need to use fancier data structures (which will be covered in CP2)
  - Segment trees
  - Binary indexed trees (Fenwick trees)

LeetCode: Evaluate Reverse Polish Notation (Required)

Evaluate the value of an arithmetic expression in Reverse Polish Notation.
(Writing “2 3 +” instead “2+3”.) Valid operators are `+`, `-`, `*`, `/`. Each operand may be an integer or another expression.
Note:

- Division between two integers should truncate toward zero.
- The given RPN expression is always valid. That means the expression would always evaluate to a result and there won't be any divide by zero operation.

Evaluate Reverse Polish Notation: Examples

```
Input: ["4", "2", "1", "+", "-", "3", "*"]      Output: 3
Explanation: ((4 - (2 + 1)) * 3) = 3

Input: ["4", "13", "5", "/", "+"]               Output: 6
Explanation: (4 + (13 / 5)) = 6
```

Stacks

A *stack* is an *abstract data type* with two operators

- **push** adds an item to the top
- **pop** removes an item from the top

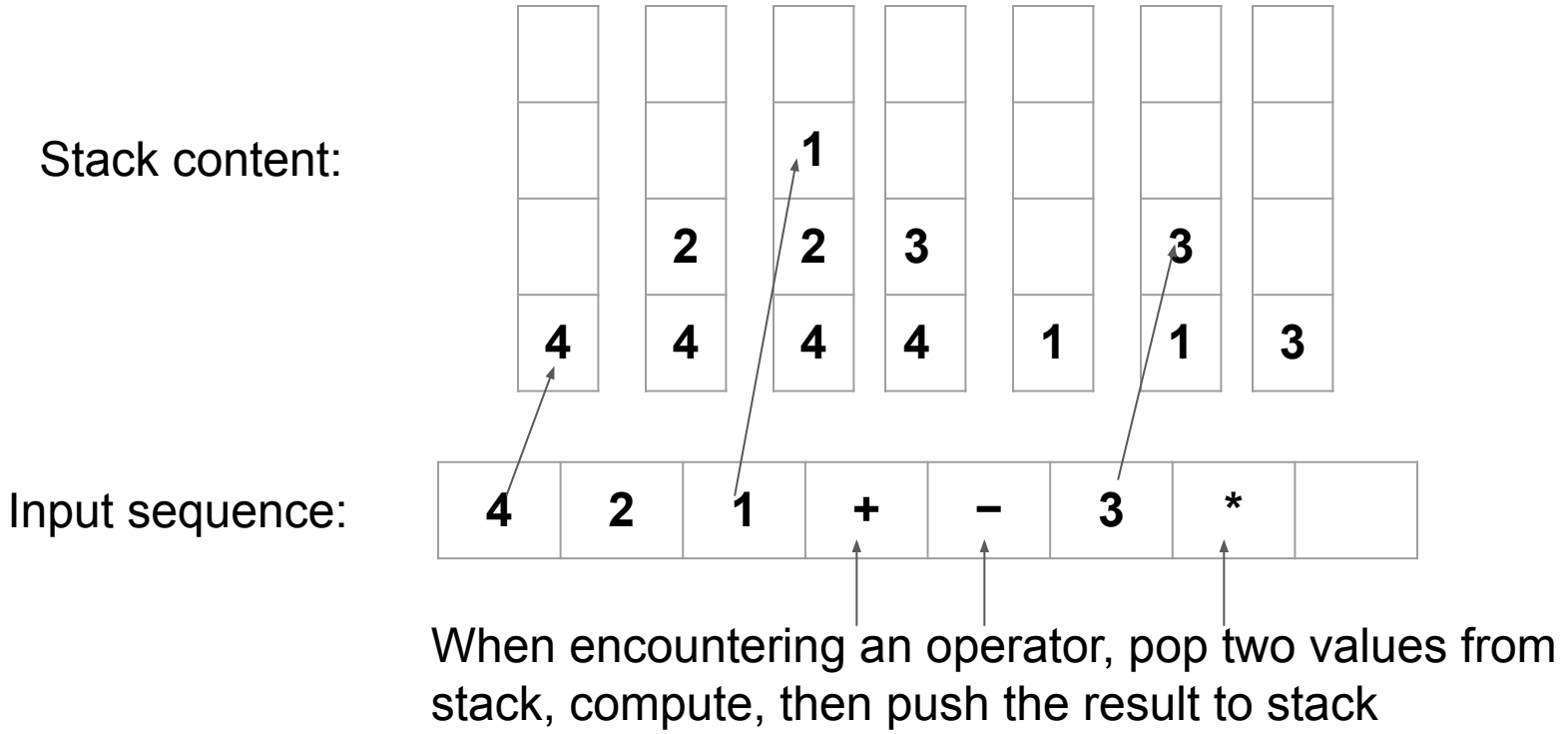
A stack is a LIFO (last in, first out) structure

- You take things off the top of the stack in the **reverse** order from which you put them on
- E.g., after push(1), push(2), push(3). The next pop() gives 3.

Understand stack is extremely important.

- Function calls are implemented (by compilers) using stacks

Evaluate Reverse Polish Notation: Stacks



Using Stack in Java

- Java has a Stack class, which is based on Vector class
- Both Stack and Vector are considered obsolete
- They were designed to allow multi-threaded access, but were done in an undesirable way, which is slow
- It is recommended to use ArrayDeque instead of Stack
- An added advantage is that ArrayDeque also support functionalities of a Queue

Java Deque Interface and ArrayDeque Class

- Deque stands for Double Ended Queue, and can support both Queue and Stack with intuitive APIs.
- To use as a Stack, push(e) = addFirst(e), pop() = removeFirst()

	First Element (Head)	Last Element (Tail)
Insert	void addFirst(E e)	void addLast(E e)
Remove	E removeFirst()	E removeLast()
Examine	E getFirst()	E getLast()

- Methods throws Exception when method fails. Adding fails when Deque is fixed size and full. Remove/get fails when empty.

LeetCode: Minimum Add to Make Parentheses Valid (Optional)

Given a string `S` of `'('` and `')'`, compute the minimum number of `'('` or `')'` that needs to be added so that the resulting parentheses string is valid.

Formally, a parentheses string is valid if and only if:

- It is the empty string, or
- It can be written as `AB` (`A` concatenated with `B`), where `A` and `B` are valid strings, or
- It can be written as `(A)`, where `A` is a valid string.

Example: `Input: "(()))("` `Output: 2`

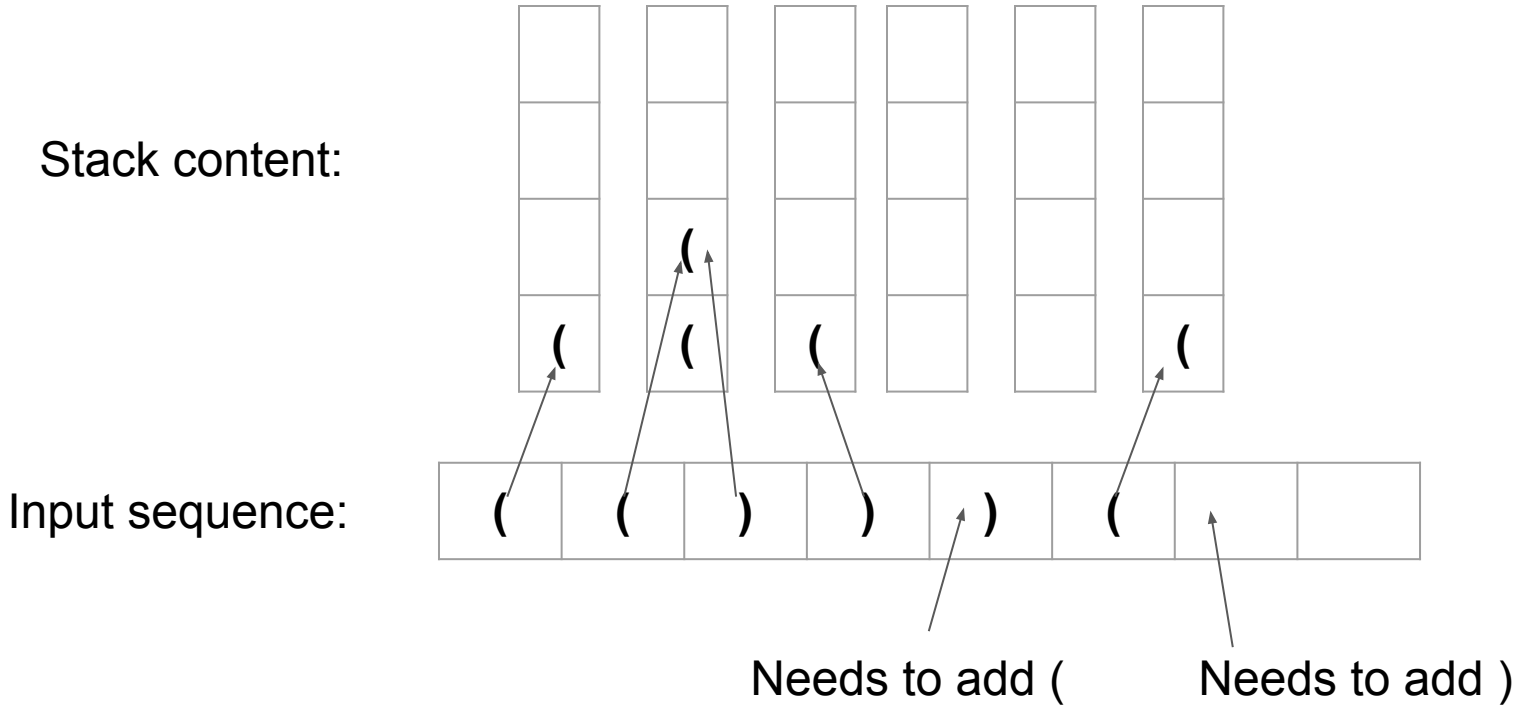
Make Parentheses Valid: Take One: Using Stacks

Using stacks to check for valid parentheses

- Whenever seeing a ‘(’, push to stack. Whenever seeing a ‘)’, try pop a ‘(‘.
- If stack always has ‘(‘ when needed, and is empty at end, then valid.

To computer minimal add, add ‘(’ whenever stack is empty when we pop, and add ‘)”s at the end to make stack empty.

Make Parentheses Valid: Stacks



Valid Parentheses Approach w/o Stacks

Formally, a parentheses string is valid if and only if:

- It is the empty string, or
- It can be written as `AB` (`A` concatenated with `B`), where `A` and `B` are valid strings, or
- It can be written as `(A)`, where `A` is a valid string.

Equivalent Condition:

- The string has the same number of ‘(’ and ‘)’
- No prefix of the string contains more ‘)’ than ‘(’

Valid Parenthses (Analysis)

View ‘(’ as +1, and ‘)’ as -1.

All prefix sums of valid parentheses must be non-negative.

This problem can be solved by maintaining a running prefix sum.

Note that when using a stack, the stack always contains some number of ‘(’’s. Its state can be captured by a single number, which is the prefix sum.

If more than one types of parentheses are used, e.g., [(){}], then stack is needed. (A number cannot capture all state information.)

LeetCode: Score of Parentheses (Optional)

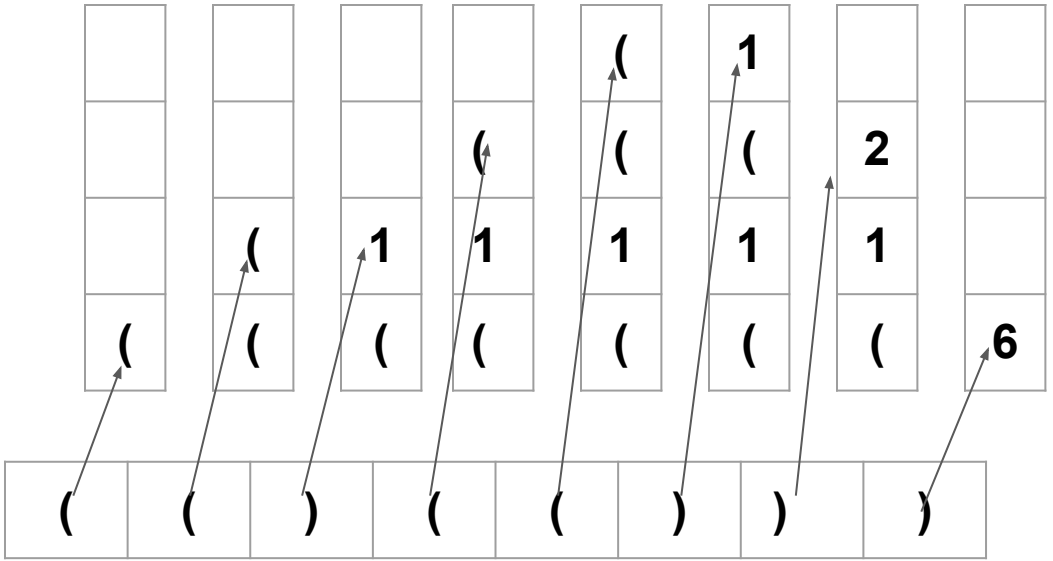
Given a balanced parentheses string `S`, compute the score of the string based on the following rule:

- `()` has score 1
- `AB` has score `A + B`, where A and B are balanced parentheses strings.
- `(A)` has score `2 * A`, where A is a balanced parentheses string.

```
Input: "(()(()))"
Output: 6
Explanation: "(()(()))" => "(1+(1))" => "(1+2)"  =>  "(3)"
```

Score of Parenthesis: Using Stacks

Stack content:



When seeing (, push to stack. When seeing), pop, if poped (, then push 1 to stack; else add up all numbers on stack until seeing (, multiple sum by 2, and push to stack.

Score of Parentheses: Without Using Stacks

Scoring rules

- `()` has score 1
- `AB` has score `A + B`, where A and B are balanced parentheses strings.
- `(A)` has score `2 * A`, where A is a balanced parentheses string.

Observations:

- Ultimately the score comes from direct pairs “()”
- The contribution of each “()” depends its depth
- For example, `in "( () ( () ) )".` There are two ()’s, the first contributed a score of 2, and the second one contributes a score of 4.
- One could keep track of the depth, and compute the score.