6.2

More Stacks and More Queues

Icebreaker

Lesson Plan

- [10] Icebreaker
- [25] Advanced Techniques

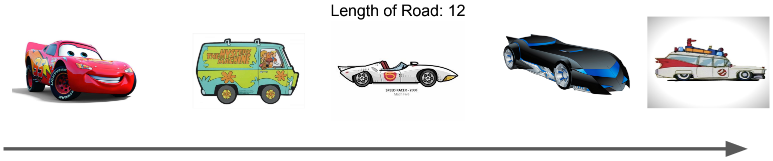
- [60]: More Practice

Why don't we see more queue problems?

Checking Against the Top of the Stack

Car Fleets

Length of Road: 12



starting position: 0
speed: 1

starting position: 3
speed: 3

starting position: 5
speed: 1

starting position: 8
speed: 4

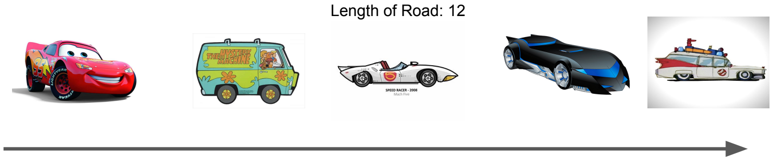
starting position: 10
speed: 2

This is a one lane road; cars cannot pass each other.
When a car catches up to a car in front of it, it forms a "fleet" - it moves at the same speed of the car in front of it, it is treated as if it's at the same position.

How many fleets reach the destination?

Car Fleets

Length of Road: 12



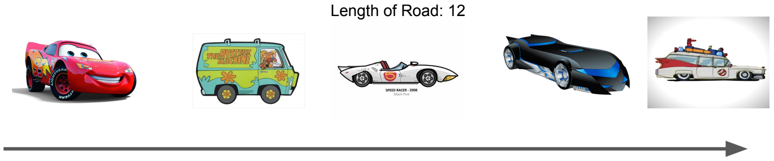
starting position: 0	starting position: 3	starting position: 5	starting position: 8	starting position: 10
speed: 1	speed: 3	speed: 1	speed: 4	speed: 2

The Mystery Machine (second car) will definitely catch up to Mach 5 (the third car). It's starting just a little bit behind but moving so much faster!

How can we solve this more generally?

Car Fleets

Length of Road: 12



starting position: 0	starting position: 3	starting position: 5	starting position: 8	starting position: 10
speed: 1	speed: 3	speed: 1	speed: 4	speed: 2
time: 12	time: 3	time: 7	time: 1	time: 1

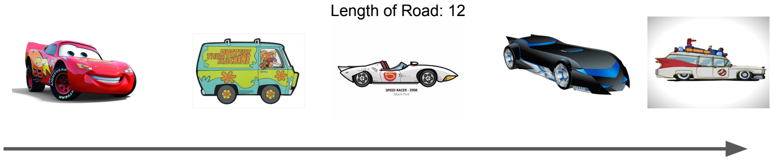
For each car, figure out how much time it needs to reach the destination!

time = distance / speed

If this math doesn't make sense, *DON'T WORRY* - we can pretend we started with the time instead!

Car Fleets

Length of Road: 12

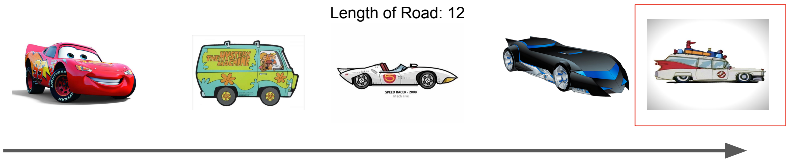


time: 12	time: 3	time: 7	time: 1	time: 1

Reduce the problem to only the information we need: the car order and the time it needs!

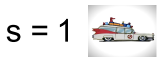
Car Fleets

Length of Road: 12



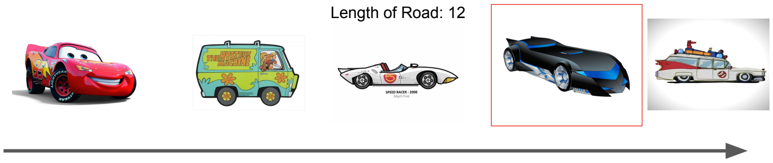
time: 12	time: 3	time: 7	time: 1	time: 1

Push the *LAST* car onto the stack



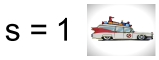
Car Fleets

Length of Road: 12



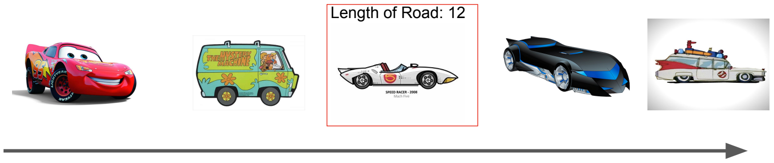
time: 12	time: 3	time: 7	time: 1	time: 1

Look at the next car and compare it to the top of the stack:
- If it needs MORE time than s[-1], it doesn't catch up;
  - Push it onto s as a new fleet!
- Else, it DOES catch up; it joins s[-1] so we can ignore it!



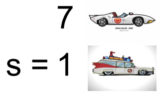
Car Fleets

Length of Road: 12



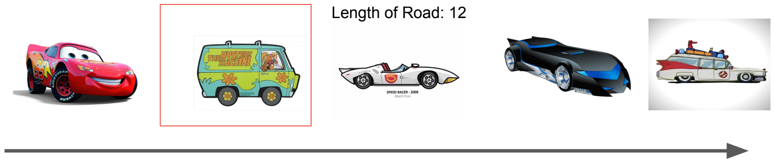
time: 12	time: 3	time: 7	time: 1	time: 1

Look at the next car and compare it to the top of the stack:
- If it needs MORE time than s[-1], it doesn't catch up;
  - Push it onto s as a new fleet!
- Else, it DOES catch up; it joins s[-1] so we can ignore it!



Car Fleets

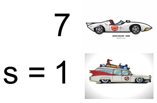
Length of Road: 12



time: 12 time: 3 time: 7 time: 1 time: 1

Look at the next car and compare it to the top of the stack:

- If it needs MORE time than s[-1], it doesn't catch up;
    - Push it onto s as a new fleet!
- Else, it DOES catch up; it joins s[-1] so we can ignore it!

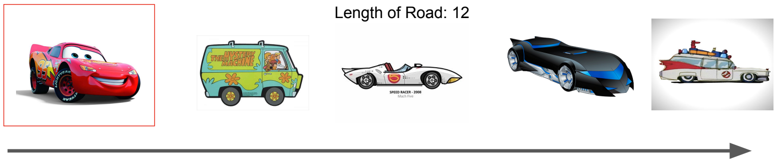


7

s = 1

Car Fleets

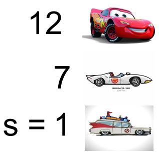
Length of Road: 12



time: 12 time: 3 time: 7 time: 1 time: 1

Look at the next car and compare it to the top of the stack:

- If it needs MORE time than s[-1], it doesn't catch up;
    - Push it onto s as a new fleet!
- Else, it DOES catch up; it joins s[-1] so we can ignore it!



12

7

s = 1

What if our inputs weren't in order to begin with?

Sort them!

Sometimes doing O(nlogn) work at the beginning to sort saves us a lot of trouble!

Next Larger

Next Larger

Given a list lst:
[2, 1, 7, 4, 5, 9, 6, 8, 3]

Return a list res where res[i] is the *first greater value* that comes after lst[i] in lst, or -1 if there is none.

[7, 7, 9, 5, 9, -1, 8, -1, -1]

Next Larger

0 1 2 3 4 5 6 7 8
[2, 1, 7, 4, 5, 9, 6, 8, 3]

0 1 2 3 4 5 6 7 8
[-1, -1, -1, -1, -1, -1, -1, -1, -1]

Push the first *INDEX* onto the stack

(we often start by 'seeding' the stack with the initial value)

s= 0 2

Next Larger

```
0 1 2 3 4 5 6 7 8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0   1   2   3   4   5   6   7   8
[-1, -1, -1, -1, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
s= 0 2
```

Next Larger

```
0 1 2 3 4 5 6 7 8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0   1   2   3   4   5   6   7   8
[-1, -1, -1, -1, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
   1 1
s= 0 2
```

Next Larger

```
0 1 2 3 4 5 6 7 8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0   1   2   3   4   5   6   7   8
[-1, -1, -1, -1, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
   1 1
s= 0 2
```

Next Larger

```
0 1 2 3 4 5 6 7 8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0   1  2   3   4   5   6   7   8
[-1, 7, -1, -1, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
   1 1    1 < 7, so res[1] = 7
s= 0 2
```

Next Larger

```
0 1 2 3 4 5 6 7 8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
0  1  2   3   4   5   6   7   8
[7, 7, -1, -1, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
s= 0 2    2 < 7, so res[0] = 7
```

Next Larger

```
0 1 2 3 4 5 6 7 8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
0  1  2   3   4   5   6   7   8
[7, 7, -1, -1, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
s=
```

Next Larger

```
 0  1  2  3  4  5  6  7  8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0  1  2   3   4   5   6   7   8
[7, 7, -1, -1, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
s= 2 7
```

Finally push this!

Next Larger

```
 0  1  2  3  4  5  6  7  8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0  1  2   3   4   5   6   7   8
[7, 7, -1, -1, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
s= 2 7
```

Next Larger

```
 0  1  2  3  4  5  6  7  8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0  1  2   3   4   5   6   7   8
[7, 7, -1, -1, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
   3 4
s= 2 7
```

Next Larger

```
 0  1  2  3  4  5  6  7  8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0  1  2   3   4   5   6   7   8
[7, 7, -1, -1, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
   3 4
s= 2 7
```

Next Larger

```
 0  1  2  3  4  5  6  7  8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0  1  2   3  4   5   6   7   8
[7, 7, -1, 5, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
   3 4
s= 2 7
```

4 < 5, so res[3] = 5

Next Larger

```
 0  1  2  3  4  5  6  7  8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0  1  2   3  4   5   6   7   8
[7, 7, -1, 5, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
s= 2 7
```

7 > 5, so stop and push!

Next Larger

```
 0  1  2  3  4  5  6  7  8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0  1  2  3   4   5   6   7   8
[7, 7, 9, 5, -1, -1, -1, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

```
   4 5
s= 2 7
```

Next Larger

```
 0  1  2  3  4  5  6  7  8
[2, 1, 7, 4, 5, 9, 6, 8, 3]
```

```
 0  1  2  3  4   5  6   7   8
[7, 7, 9, 5, 9, -1, 8, -1, -1]
```

```
For each subsequent index i
Repeat this process:
    Look at the top element of the stack (index j)
    If the value at index j is bigger, stop
    If the value at index j is smaller:
        res[j] = lst[i] !!!
        pop!!!

Then push i onto the stack!
```

“Recursion”

Recursion By Iteration

Claim: Any problem that can be done recursively can also be done iteratively.

We can use a stack to mimic the recursion!

Recursion: Making Change

Remember the recursive making change problem:

Given a number n, can we make change for it using only 7, 11, and 19 cent coins?

Recursion: Making Change

n = 25

Push n onto the stack

s= 25

Recursion: Making Change

```
while s:
    curr = s.pop()  25
    if curr == 0:
        return True
    if curr < 0:
        continue
    s.append(curr - 19)
    s.append(curr - 11)
    s.append(curr - 6)
return False
```

```
    19
    14
s=  6
```

Recursion: Making Change

```
while s:
    curr = s.pop()  19
    if curr == 0:
        return True
    if curr < 0:
        continue
    s.append(curr - 19)
    s.append(curr - 11)
    s.append(curr - 6)
return False
```

```
    13
    8
    0   Eventually we'll pop this and return True!
    14
s=  6
```

Stack vs. Queue

This would work with a Queue too!

It's the difference between Depth-First Search and Breadth-First Search, which we'll learn all about later on.

Practice Problems - More Stacks [repl.it]