Munster Programming Training
Cycle 2
Lecture 2

Milan De Cauwer & Andrea Visentin

Account

MPT 2-3-4 2016-17			
Student First Name	Student Surname		Username
Aleksei	Ivanov		ai3
Andrew	Nash		an12
Benjamin	Provan-Bessell		bpb2
David	Byron		db28
Julia	Sheehan		js37
Katrin	Birk		kb26
Michael	Condon		mc59
Oisin	Cronin		oc8
Paul	Gunnarsson		pg6
Rachel	Cullen		rc19
Melina	Neilson		mn16
Éabha	McMahon		em22
Jonathan	Hanley		cycle 3 jh22
Conor	Holden		cycle 3 ch35
Bradley	Rees		cycle 3 br3
Diarmuid	O'Donoghue		cycle 3 dod5
Emily	Ray		cycle 3 er7
Conor	Bradley		cycle 3 cb18
Tadhg	Kearney		cycle 3 tk9
Caelum	Forder		cycle2 cf22

Password:
November

Remember to change it

Last lecture

- We introduce what an algorithm is
- We saw how a problem can be solved with different algorithms
- Not all the algorithms are equivalent

Recall on Big O

Big O notation is used in Computer Science to describe the performance or complexity of an algorithm.

Big O specifically describes the worst-case scenario, and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

It is always a function of the problem size.

Problem size n

Usually we call the problem size n.

n can be:

- length of the array
- number of edge in a graph
- number of product to produce

In different occasions n can assume different meaning.

Constant – O(1)

O(1) describes an algorithm that will always execute in the same time (or space) regardless of the size of the input data set.

Example:

```
>>> 5 + 3
8
>>> k = k + 1
```

Logaritmic – O(log N)

It’s early to understand O(log N). But consider that is slower than O(1) and faster than O(N).

Linear – O(N)

O(N) describes an algorithm whose performance will grow linearly and in direct proportion to the size of the input data set.

```
>>>for x in range(0, n + 1):
>>>       print "We're on time %d" % (x)
```

Quadratic– O(N^2)

O(N^2) represents an algorithm whose performance is directly proportional to the square of the size of the input data set.

This is common with algorithms that involve nested iterations over the data set.

```
>>>for x in range(1, n):
>>>     for y in range(1, n):
>>>          print '%d * %d = %d' % (x, y, x*y)
```

Exponential– $O(2^N)$

$O(2^N)$ denotes an algorithm whose growth doubles with each additon to the input data set.

The growth curve of an $O(2^N)$ function is exponential - starting off very shallow, then rising meteorically.

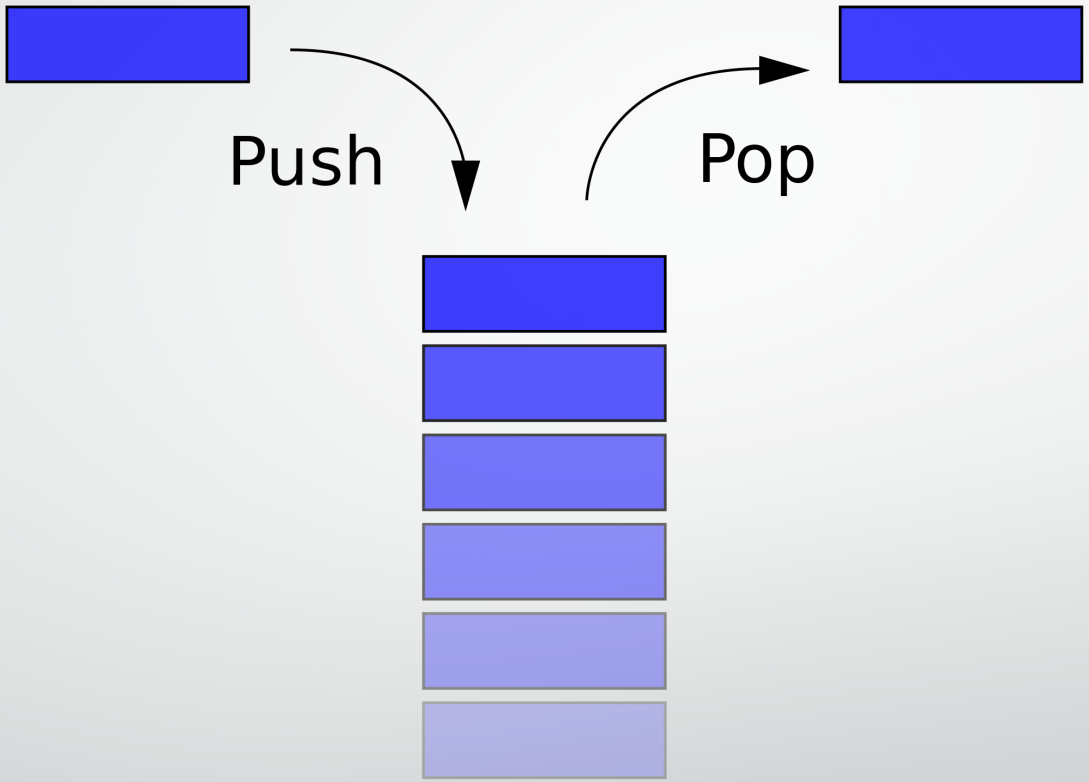
Example

http://bigocheatsheet.com/

Data Structure

It’s important to choose the right one for your problem.

Queue

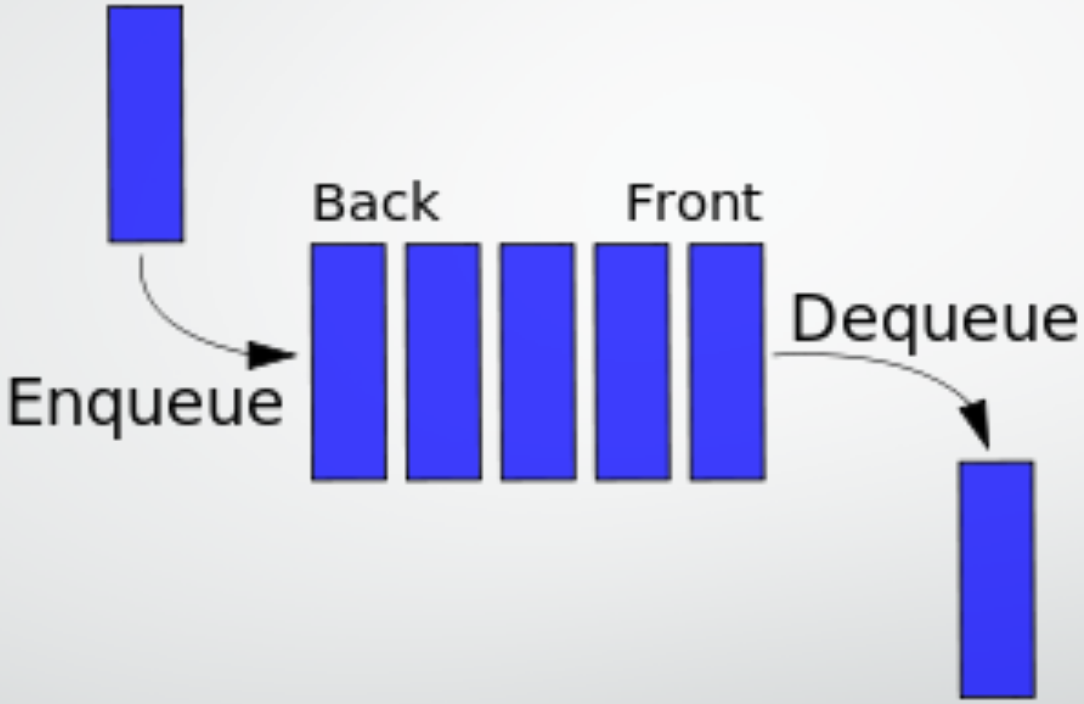


Order: Last in, first out (LIFO)

Stack Example

```
>>> myStack = []
>>>
>>> myStack.append(3)
>>> myStack.append(3)
>>>
>>> myStack
[3 , 5]
 >>> myStack.pop()
5
>>> myStack
[3 ]
```

Queue



Order: First in, first out (LIFO)

Stack Example

```
>>> from collections import deque
>>> myQueue = deque([1, 2, 3])
>>> myQueue.append(4)
>>> myQueue.append(5)
>>> myQueue.popleft()
1
>>> myQueue.popleft()
2
>>> myQueue
deque([3, 4, 5])
```

Exercise Stack and Queue

Your task in this exercise is to implement both the stack and queue operations described in the lecture. Your program should read in a list of integers and insert each one into both a stack and a queue. It should then print out the contents of the stack and then the contents of the queue.

Input

The first line of input contains N, the number of integers. $0 < N < 10000$. The next line of input then contains N integers.

Output

Your program should output the contents of the stack in last-in first-out order, followed immediately by a new line. Then the contents of the queue in first-in first-out order.