COMP1819
Algorithms and Data Structures

Lecture 03: Stacks,

and Queues

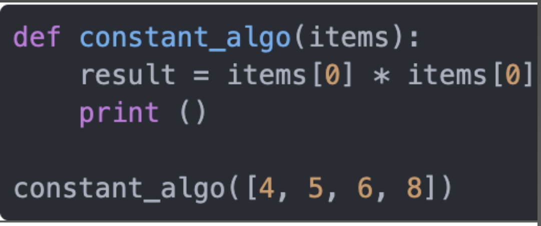
Dr. Tuan Vuong

02/02/2021

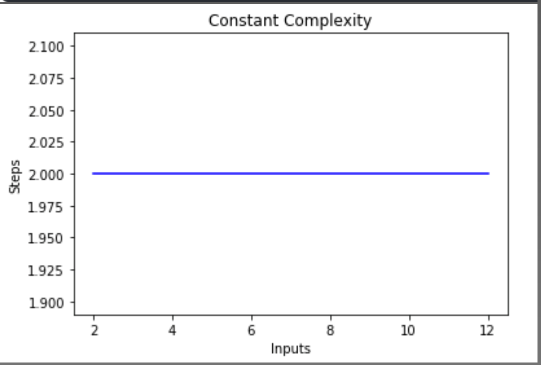
Content

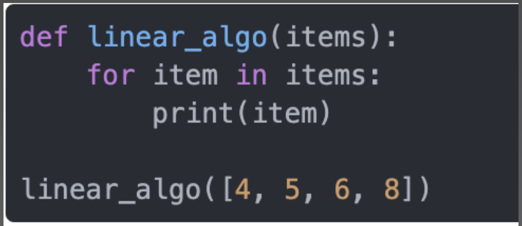
- Revision on Algorithm Analysis
- Lab 02 Discussion (Prime numbers)
- Stacks
- Queues
- Deques
- Reinforcement

Algorithm Analysis – Big O notation (Complexity in term of n – input size)

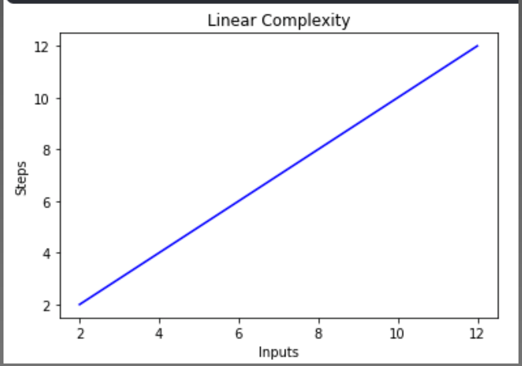


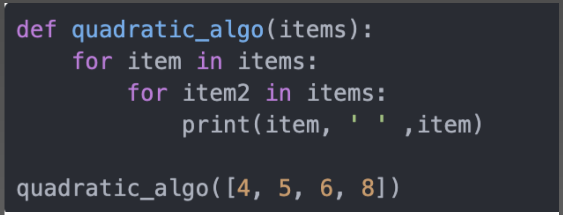
```python
def constant_algo(items):
    result = items[0] * items[0]
    print ()

constant_algo([4, 5, 6, 8])
```



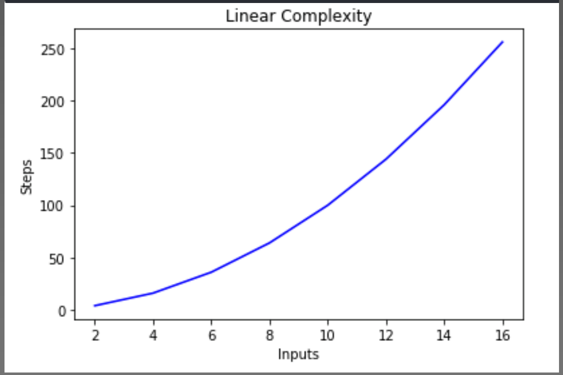


```python
def linear_algo(items):
    for item in items:
        print(item)

linear_algo([4, 5, 6, 8])
```



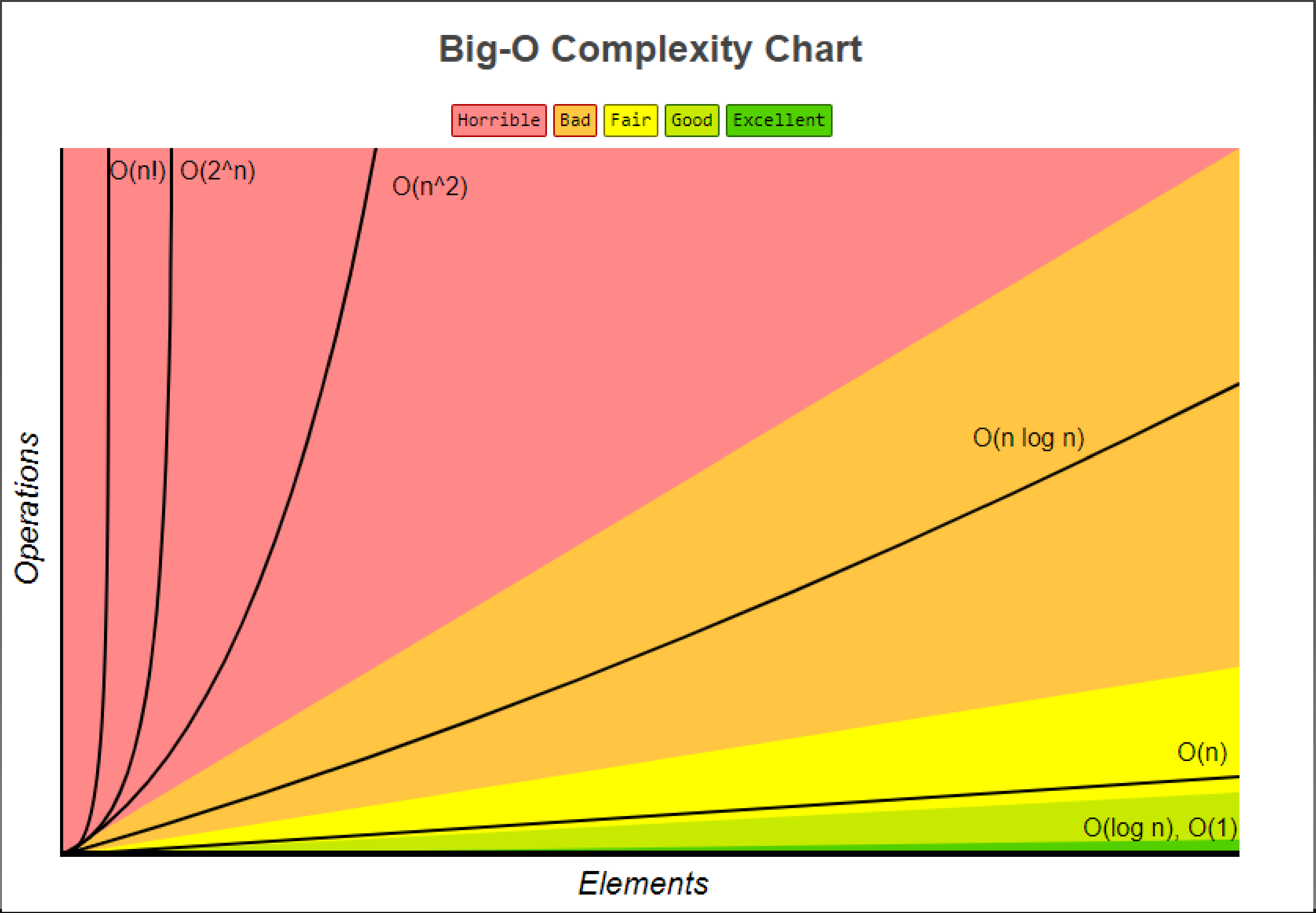


```python
def quadratic_algo(items):
    for item in items:
        for item2 in items:
            print(item, ' ' ,item)

quadratic_algo([4, 5, 6, 8])
```



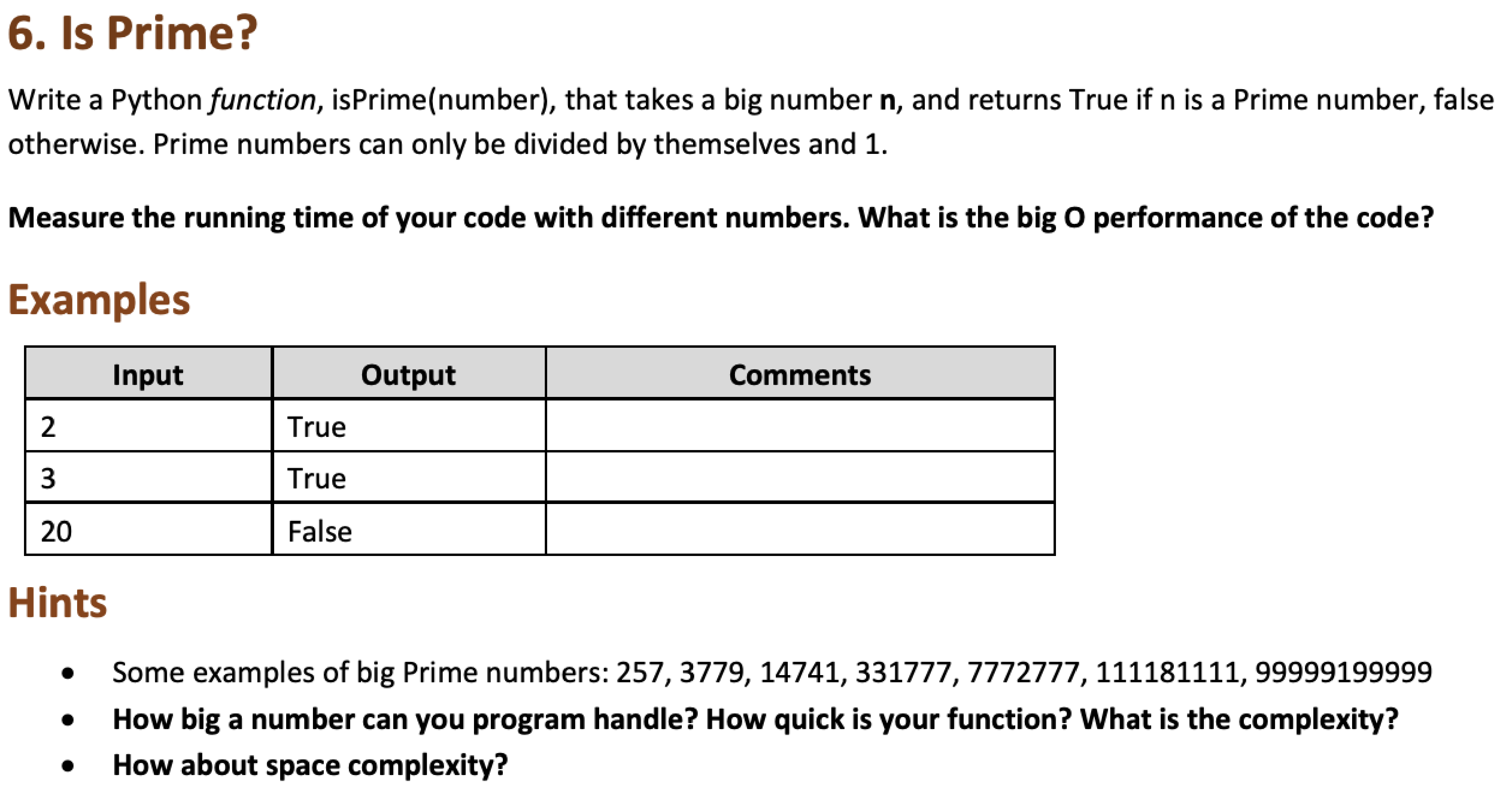
Common rules:

- Constants can be omitted: O(100) -> O(1), O(3n) -> O(n), O(7n^2) -> O(n^2)
- Smaller terms can be omitted: O(200+3n) -> O(n), O(4n+7n^2) -> O(n^2)



How about Lab 02 isPrime?

Lab 02



6. Is Prime?

Write a Python *function*, isPrime(number), that takes a big number **n**, and returns True if n is a Prime number, false otherwise. Prime numbers can only be divided by themselves and 1.

Measure the running time of your code with different numbers. What is the big O performance of the code?

Examples

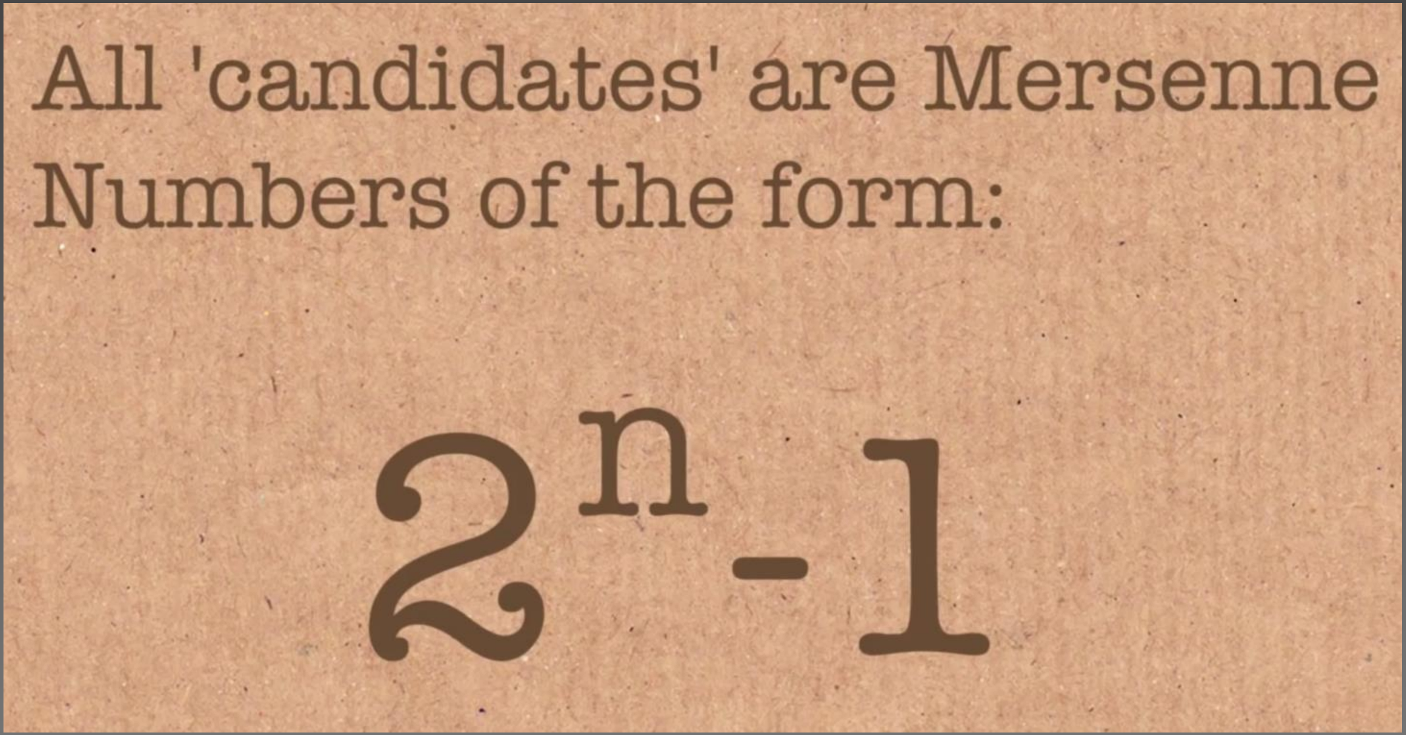
Input	Output	Comments
2	True	
3	True	
20	False	

Hints

- Some examples of big Prime numbers: 257, 3779, 14741, 331777, 7772777, 111181111, 99999199999
- **How big a number can you program handle? How quick is your function? What is the complexity?**
- **How about space complexity?**

Prime numbers have ONLY 2 factors. Hence, we can loop from 1 to n to count numbers of factors: `len([f for f in range(1,n+1) if n%f==0])`
Which is the biggest prime number we found?

Largest Known Prime number

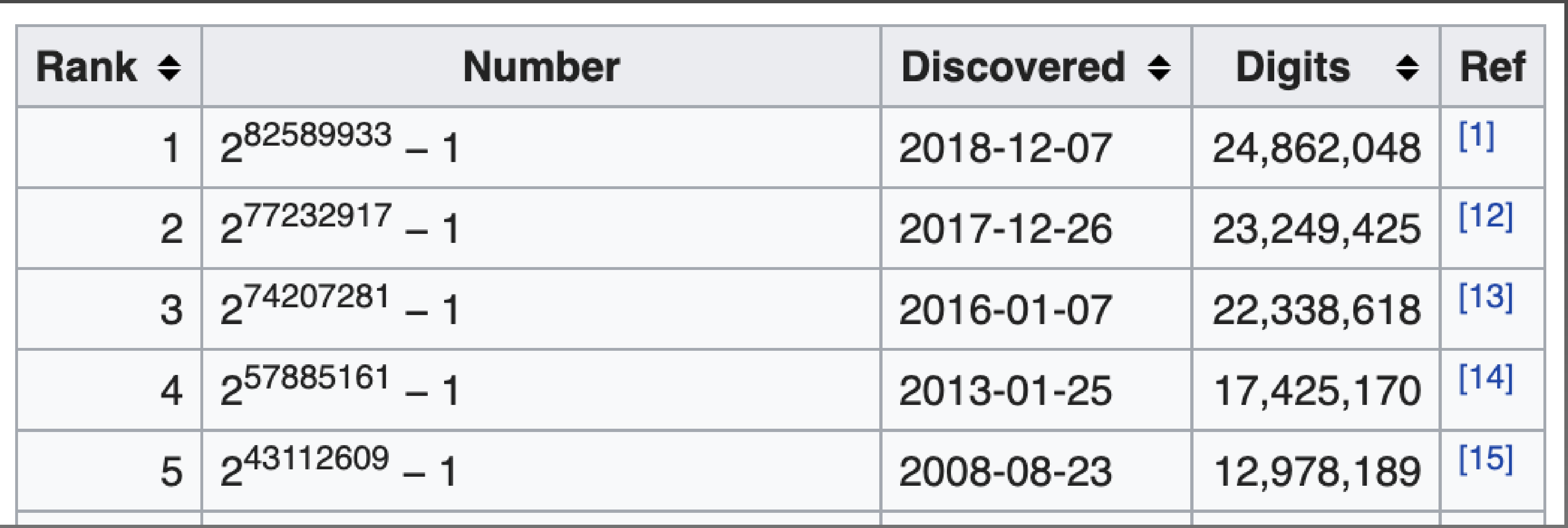


All 'candidates' are Mersenne Numbers of the form:

$$2^n-1$$

Some prime numbers: 2, 3, 5, 7, 11, 13, 17, 19, …
n = 2 -> Prime = 3?
n = 3 -> Prime = 7?
n = 4 -> Prime = 15?
Which one is the largest known? How many digits?

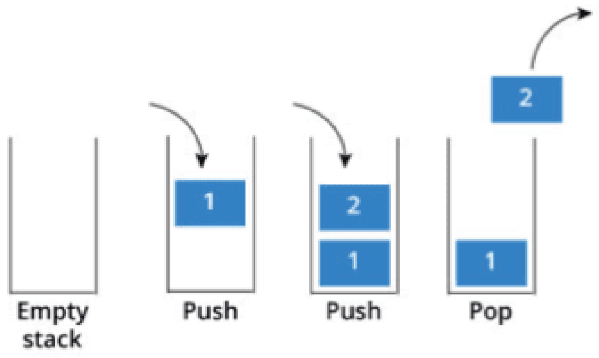
Largest Known Prime number



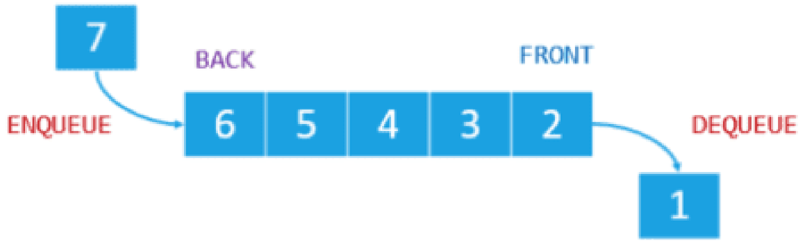
Rank	Number	Discovered	Digits	Ref
1	$2^{82589933} - 1$	2018-12-07	24,862,048	[1]
2	$2^{77232917} - 1$	2017-12-26	23,249,425	[12]
3	$2^{74207281} - 1$	2016-01-07	22,338,618	[13]
4	$2^{57885161} - 1$	2013-01-25	17,425,170	[14]
5	$2^{43112609} - 1$	2008-08-23	12,978,189	[15]

https://www.youtube.com/watch?v=tlpYjrbujG0

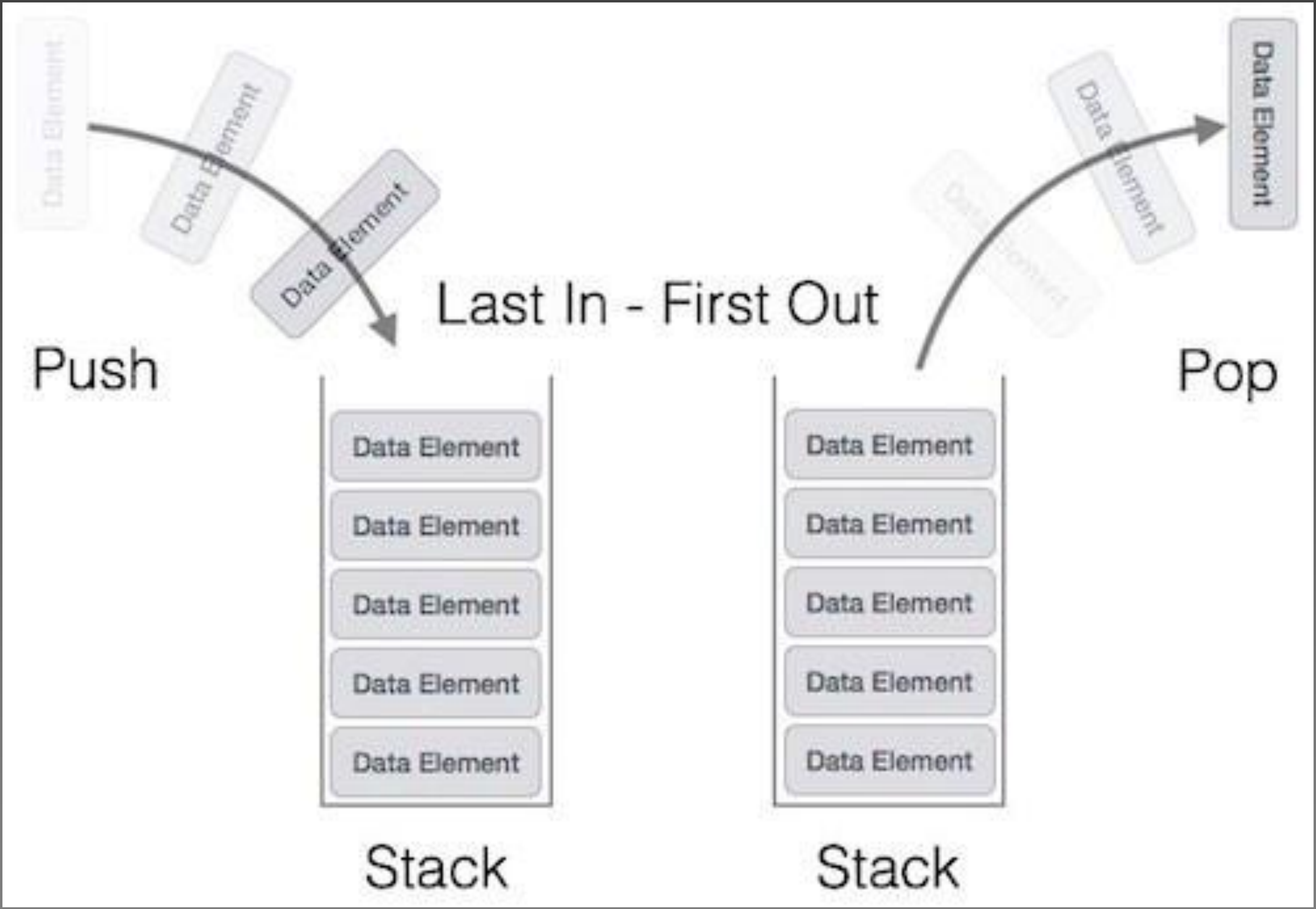
Data Structure Basics



Stack



Queue

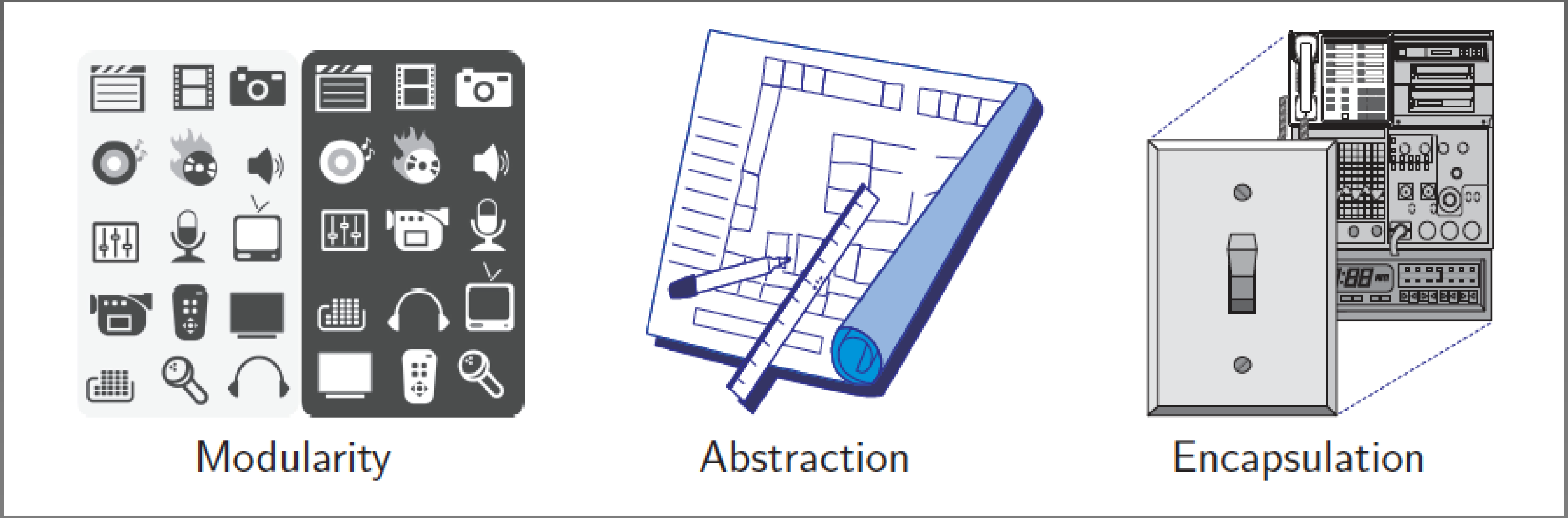

Last In - First Out
Push
Pop
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Data Element
Stack
Stack

Object-Oriented Design Principles

Modularity

Abstraction

Encapsulation



Abstract Data Types (ADTs)

- An abstract data type (ADT) is an abstraction of a data structure
- An ADT specifies:
  - Data stored
  - Operations on the data
  - Error conditions associated with operations

- Example: ADT modeling a simple stock trading system
  - The data stored are buy/sell orders
  - The operations supported are
    - order buy(stock, shares, price)
    - order sell(stock, shares, price)
    - void cancel(order)
  - Error conditions:
    - Buy/sell a nonexistent stock
    - Cancel a nonexistent order

Example

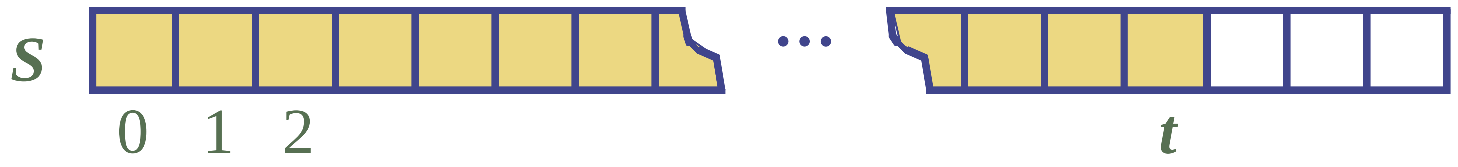
Operation	Return Value	Stack Contents
S.push(5)	–	[5]
S.push(3)	–	[5, 3]
len(S)	2	[5, 3]
S.pop()	3	[5]
S.is_empty()	False	[5]
S.pop()	5	[ ]
S.is_empty()	True	[ ]
S.pop()	"error"	[ ]
S.push(7)	–	[7]
S.push(9)	–	[7, 9]
S.top()	9	[7, 9]
S.push(4)	–	[7, 9, 4]
len(S)	3	[7, 9, 4]
S.pop()	4	[7, 9]
S.push(6)	–	[7, 9, 6]
S.push(8)	–	[7, 9, 6, 8]
S.pop()	8	[7, 9, 6]

Applications of Stacks

- Direct applications
  - Page-visited history in a Web browser
  - Undo sequence in a text editor
  - Chain of method calls in a language that supports recursion
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

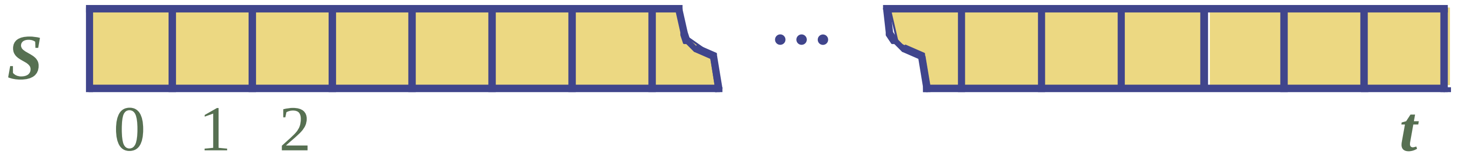
Array-based Stack

- A simple way of implementing the Stack ADT uses an array
- We add elements from left to right
- A variable keeps track of the index of the top element



Array-based Stack (cont.)

- The array storing the stack elements may become full
- A push operation will then need to grow the array and copy all the elements over.



Performance and Limitations

- Performance
  - Let n be the number of elements in the stack
  - The space used is $O(n)$
  - Each operation runs in time $O(1)$ (amortized in the case of a push)

Array-based Stack in Python

```python
class ArrayStack:
  """LIFO Stack implementation using a Python list as underlying storage."""

  def __init__(self):
    """Create an empty stack."""
    self._data = [ ]                      # nonpublic list instance

  def __len__(self):
    """Return the number of elements in the stack."""
    return len(self._data)

  def is_empty(self):
    """Return True if the stack is empty."""
    return len(self._data) == 0

  def push(self, e):
    """Add element e to the top of the stack."""
    self._data.append(e)                  # new item stored at end of list

  def top(self):
    """Return (but do not remove) the element at the top of the stack.

    Raise Empty exception if the stack is empty.
    """
    if self.is_empty():
      raise Empty('Stack is empty')
    return self._data[−1]                 # the last item in the list

  def pop(self):
    """Remove and return the element from the top of the stack (i.e., LIFO).

    Raise Empty exception if the stack is empty.
    """
    if self.is_empty():
      raise Empty('Stack is empty')
    return self._data.pop( )              # remove last item from list
```

Parentheses Matching

- Each "(", "{", or "[" must be paired with a matching ")", "}", or "["
  - correct: ( )(( )){([( )])}
  - correct: ((( )(( )){([( )])}
  - incorrect:)(( )){([( )])}
  - incorrect: ({[ ])}
  - incorrect: (

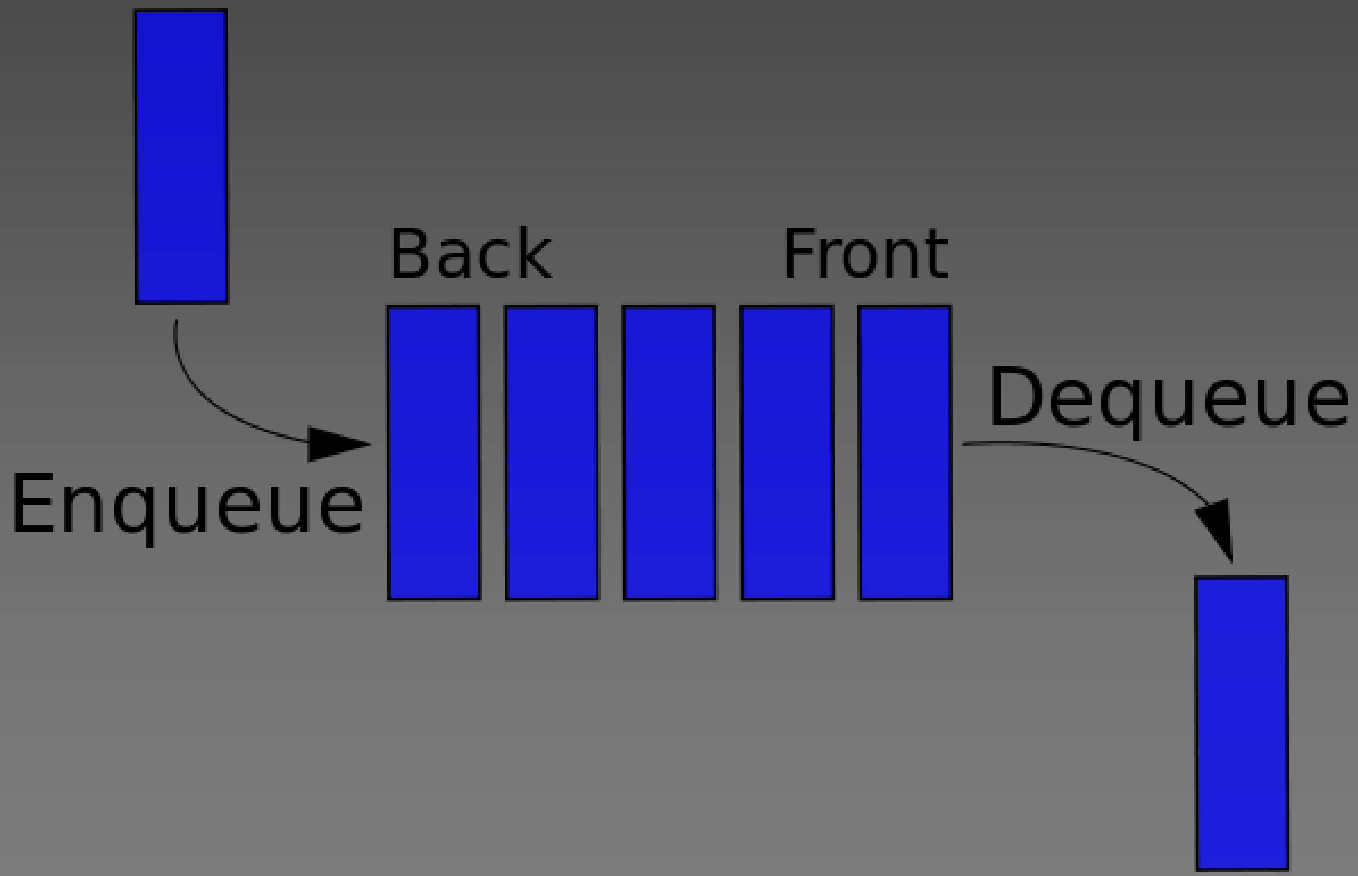
Parentheses Matching Algorithm

```
Algorithm ParenMatch(X,n):
Input: An array X of n tokens, each of which is either a grouping symbol, a
variable, an arithmetic operator, or a number
Output: true if and only if all the grouping symbols in X match
Let S be an empty stack
for i=0 to n-1 do
    if X[i] is an opening grouping symbol then
        S.push(X[i])
    else if X[i] is a closing grouping symbol then
        if S.is_empty() then
            return false {nothing to match with}
        if S.pop() does not match the type of X[i] then
            return false {wrong type}
if S.isEmpty() then
    return true {every symbol matched}
else return false {some symbols were never matched}
```

Parentheses Matching in Python

```
def is_matched(expr):
  """Return True if all delimiters are properly match; False otherwise."""
  lefty = '({['                                   # opening delimiters
  righty = ')}]'                                  # respective closing delims
  S = ArrayStack()
  for c in expr:
    if c in lefty:
      S.push(c)                                   # push left delimiter on stack
    elif c in righty:
      if S.is_empty():
        return False                              # nothing to match with
      if righty.index(c) != lefty.index(S.pop()):
        return False                              # mismatched
  return S.is_empty()                             # were all symbols matched?
```

Queue



The Queue ADT

- The Queue ADT stores arbitrary objects
- Insertions and deletions follow the first-in first-out scheme
- Insertions are at the rear of the queue and removals are at the front of the queue
- Main queue operations:
  - enqueue(object): inserts an element at the end of the queue
  - object dequeue(): removes and returns the element at the front of the queue
- Auxiliary queue operations:
  - object first(): returns the element at the front without removing it
  - integer len(): returns the number of elements stored
  - boolean is_empty(): indicates whether no elements are stored
- Exceptions
  - Attempting the execution of dequeue or front on an empty queue throws an EmptyQueueException

Example

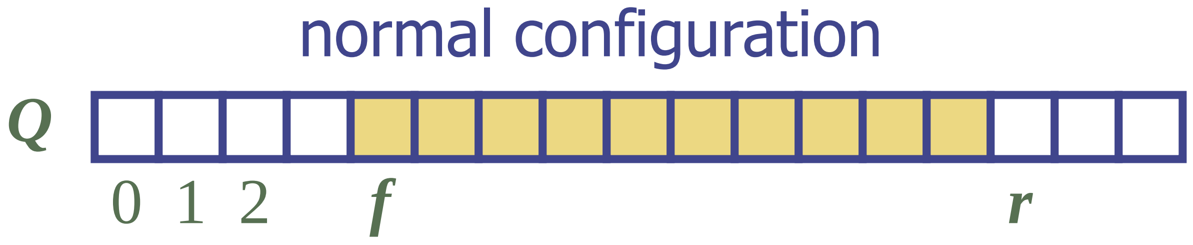
Operation	Return Value	first $\leftarrow Q \leftarrow$ last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue( )	5	[3]
Q.is_empty( )	False	[3]
Q.dequeue( )	3	[ ]
Q.is_empty( )	True	[ ]
Q.dequeue( )	“error”	[ ]
Q.enqueue(7)	–	[7]
Q.enqueue(9)	–	[7, 9]
Q.first( )	7	[7, 9]
Q.enqueue(4)	–	[7, 9, 4]
len(Q)	3	[7, 9, 4]
Q.dequeue( )	7	[9, 4]

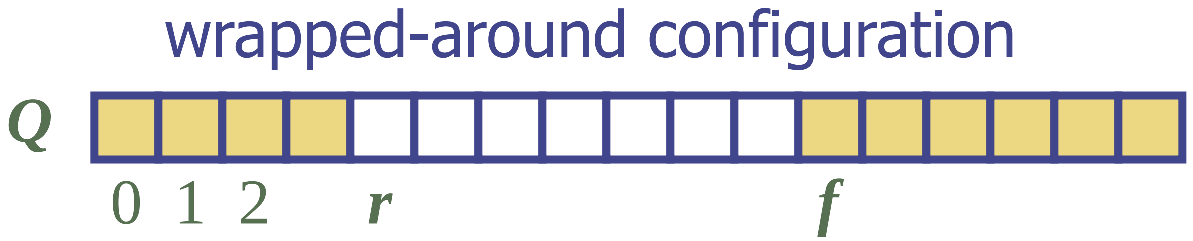
Applications of Queues

- Direct applications
  - Waiting lists, bureaucracy
  - Access to shared resources (e.g., printer)
  - Multiprogramming
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

Array-based Queue

- Use an array of size N in a circular fashion
- Two variables keep track of the front and rear
  - f index of the front element
  - r index immediately past the rear element
- Array location r is kept empty

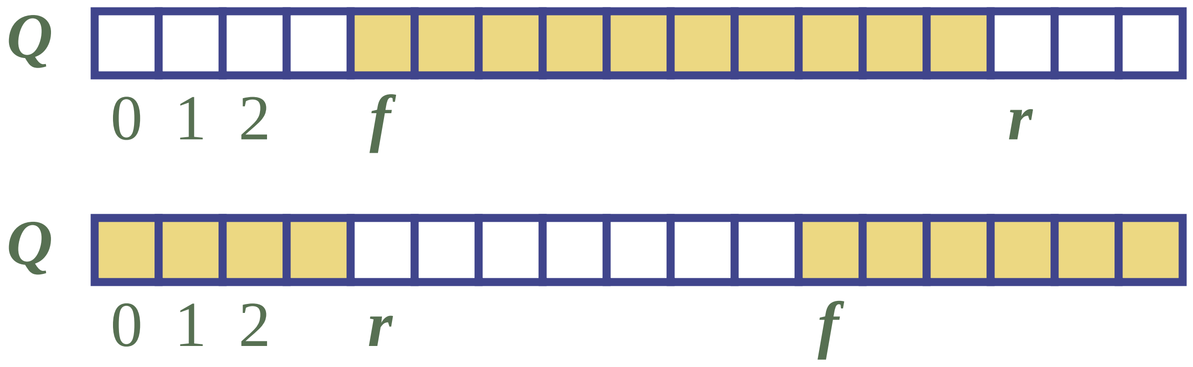




Queue Operations

- We use the modulo operator (remainder of division)

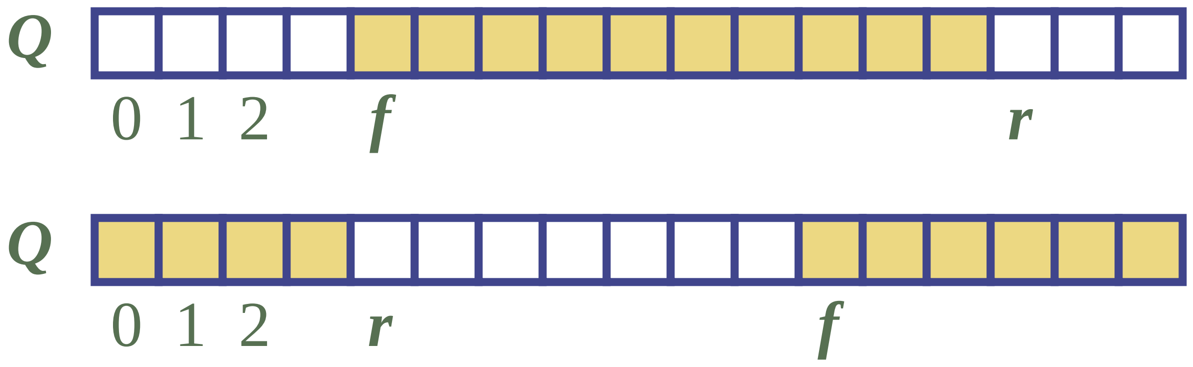
```
Algorithm size()
    return $(N - f + r) \bmod N$

Algorithm isEmpty()
    return $(f = r)$
```



Queue Operations (cont.)

- Operation enqueue throws an exception if the array is full
- This exception is implementation-dependent

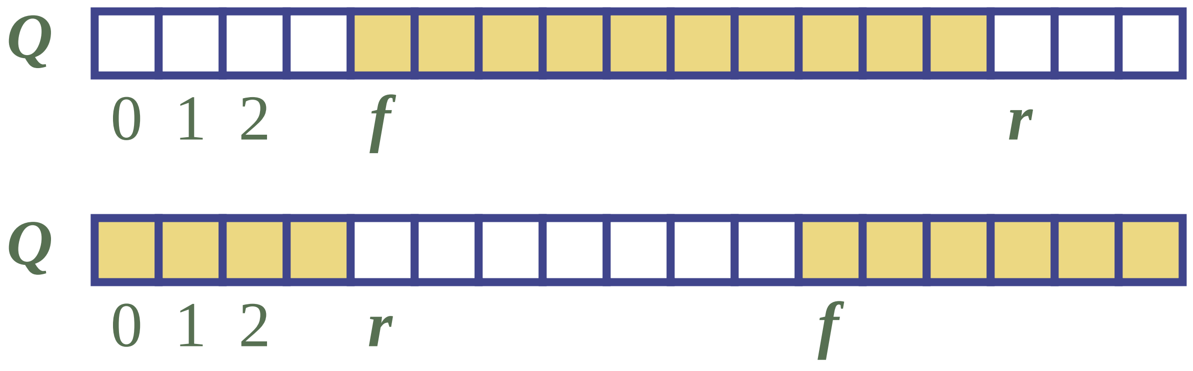
```
Algorithm enqueue(o)
  if size() = N − 1 then
    throw FullQueueException
  else
    Q[r] ← o
    r ← (r + 1) mod N
```



Queue Operations (cont.)

- Operation dequeue throws an exception if the queue is empty
- This exception is specified in the queue ADT

```
Algorithm dequeue()
  if isEmpty() then
    throw EmptyQueueException
  else
    $o \leftarrow Q[f]$
    $f \leftarrow (f + 1) \bmod N$
    return $o$
```



Queue in Python

- Use the following three instance variables:
  - _data: is a reference to a list instance with a fixed capacity.
  - _size: is an integer representing the current number of elements stored in the queue (as opposed to the length of the data list).
  - _front: is an integer that represents the index within data of the first element of the queue (assuming the queue is not empty).

Queue in Python, Beginning

```
class ArrayQueue:
  """FIFO queue implementation using a Python list as underlying storage."""
  DEFAULT_CAPACITY = 10          # moderate capacity for all new queues

  def __init__(self):
    """Create an empty queue."""
    self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
    self._size = 0
    self._front = 0

  def __len__(self):
    """Return the number of elements in the queue."""
    return self._size

  def is_empty(self):
    """Return True if the queue is empty."""
    return self._size == 0

  def first(self):
    """Return (but do not remove) the element at the front of the queue.

    Raise Empty exception if the queue is empty.
    """
    if self.is_empty():
      raise Empty('Queue is empty')
    return self._data[self._front]

  def dequeue(self):
    """Remove and return the first element of the queue (i.e., FIFO).

    Raise Empty exception if the queue is empty.
    """
    if self.is_empty():
      raise Empty('Queue is empty')
    answer = self._data[self._front]
    self._data[self._front] = None                  # help garbage collection
    self._front = (self._front + 1) % len(self._data)
    self._size -= 1
    return answer
```

Queue in Python, Continued

```
def enqueue(self, e):
  """Add an element to the back of queue."""
  if self._size == len(self._data):
    self._resize(2 * len(self.data))              # double the array size
  avail = (self._front + self._size) % len(self._data)
  self._data[avail] = e
  self._size += 1

def _resize(self, cap):                           # we assume cap >= len(self)
  """Resize to a new list of capacity >= len(self)."""
  old = self._data                                # keep track of existing list
  self._data = [None] * cap                       # allocate list with new capacity
  walk = self._front
  for k in range(self._size):                     # only consider existing elements
    self._data[k] = old[walk]                     # intentionally shift indices
    walk = (1 + walk) % len(old)                  # use old size as modulus
  self._front = 0                                 # front has been realigned
```

Reinforcement

Discussion Questions - Stack

What values are returned during the following series of stack operations, if executed upon an initially empty stack? push(5), push(3), pop(), push(2), push(8), pop(), pop(), push(9), push(1), pop(), push(7), push(6), pop(), pop(), push(4), pop(), pop().

Discussion Questions - Queue

What values are returned during the following sequence of queue operations, if executed on an initially empty queue? enqueue(5), enqueue(3), dequeue(), enqueue(2), enqueue(8), dequeue(), dequeue(), enqueue(9), enqueue(1), dequeue(), enqueue(7), enqueue(6), dequeue(), dequeue(), enqueue(4), dequeue(), dequeue().

Quick overview

- Stacks are simple data structures that maintain a LIFO, last-in first-out, ordering.
- The fundamental operations for a stack are push, pop, and is_empty.
- Queues are simple data structures that maintain a FIFO, first-in first-out, ordering.
- The fundamental operations for a queue are enqueue, dequeue, and is_empty.

Extra reading

- Dequeue
- Lists
- Prefix, infix, and postfix expressions

Next week