CONCEPTS OF PROGRAMMING

MPCS 50101
AUTUMN 2019
SESSION 12

THE UNIVERSITY OF CHICAGO

EXAM REVIEW

EXAM REVIEW

PROBLEM 1

```
T a)  True and True
F b)  False and True
F c)  1 == 1 and 2 == 1
T d)  "chicago" == 'chicago'
T e)  1 == 1 or 2 != 1
T f)  True and 1 == 1
F g)  False and 0 != 0
T h)  True or 1 == 1
T i)  "dogs" == "dog" + "s"
F j)  1 != 0 and 2 == 1
T k)  "dog" != "cat"
T l)  1 == int("1")
T m)  not (True and False)
F n)  not (1 == 1 and 0 != 1)
F o)  not (10 == 1 or 1000 == 1000)
F p)  not (1 != 10 or 3 == 4)
T q)  not ("cat" == "cat" and "cat" == "purr")
T r)  1 == 1 and (not ("dog" == 1 or 1 == 0))
F s)  "exam" == "easy" and (not (3 == 4 or 3 == 3))
F t)  3 == 3 and (not ("coding" == "coding" or "Python" == "fun"))
F u)  isinstance(0,str)
T v)  isinstance("int",str)
```

EXAM REVIEW

PROBLEM 2

```
i = 1
while i < 5:
    j = 0
    while j < i:
        print(i % 2)
        j = j + 1
    i = i + 1
    print("")
```

DESCRIBE?

EXAM REVIEW

PROBLEM 2

4 ITERATIONS

NUMBERS IN GROUP

```
i = 1
while i < 5:
    j = 0
    while j < i:
        print(i % 2)
        j = j + 1
    i = i + 1
    print("")
```

BLANK LINE

ALTERNATING 1 AND 0

```
1

0
0

1
1
1

0
0
0
0
```

EXAM REVIEW

PROBLEM 3

```
greeting = "hello world"
print(greeting[:5])

> hello
```

EXAM REVIEW

PROBLEM 3

```
try:
    dog = int("dog")
except:
    print("A")
else:
    print("B")
```

```
> A
```

EXAM REVIEW

PROBLEM 3

```
length = 3
width = 2.0
print(length / width)

> 1.5
```

EXAM REVIEW

PROBLEM 3

```
numbers = [1,2,3,4,5]
print(sum(numbers))

> 15


print(min(numbers[1:]))

> 2
```

EXAM REVIEW

PROBLEM 3

```
favorite_numbers = {"Sally": 12, "Linus": 14,
                    "Charlie": 5}
print(favorite_numbers["Linus"])
> 14

for something in favorite_numbers:
    print(something)

> Sally
> Linus
> Charlie
```

EXAM REVIEW

PROBLEM 3

```
def my_function(n):
    if n < 0:
        print(':)')
    else:
        print("%s" % n)
        my_function(n-1)

my_function(5)
```

```
5
4
3
2
1
0
:)
```

EXAM REVIEW

PROBLEM 4

```
# Problem 4.
#
# Write a currency conversion program that allows the users to input money in
# United States Dollars and have it converted to Japenes Yen.
#
# Prompt the user to enter a dollar amount in USD. Using the input, convert the
# money to Yen and display the result. The exchange rate is 1 Japanese Yen equals
# 0.0092 USD.
#
# Print out the both the USD and Yen values. Use the exact format presented below
# when printing the screen.
#
#      United States Dollars: 92.44
#      Japanese Yen: 1000.00
#
# Validate the user input to ensure that the calculations can be completed.  If
# not, print a message to the user that indicates that there was a problem
# with the input and end the program.
```

```
def validate_input(usd):
     #
     return valid_usd


def convert(usd):
    #
    return yen

def main():
    usd  = input("Enter a dollar amount?")
    valid_usd = validate_input(usd)
    yen = convert(valid_usd)
    print("Japanese Yen: 92.00")


if __name__ == '__main__':
    main()
```

```python
def validate_input(usd):
     #
      return valid_usd


def convert(usd):
    #
    return yen

def main():
    usd  = input("Enter a dollar amount?")
    valid_usd = validate_input(usd)
   yen = convert(valid_usd)
   print("Japanese Yen: 92.00")


if __name__ == '__main__':
    main()
```

FUNCTIONS TAKE INPUT AND RETURN VALUE

EACH MAIN TASK HAS A FUNCTION

ALL INPUT AND OUTPUT FOR PROGRAM IS DONE IN MAIN()

EXAM REVIEW

PROBLEM 6

```
# Problem 6.
#
# One of the best ways to improve as a Scrabble player is to memorize all the 2
# and 3 letter words allowed during game play.  It should be noted that many
# of the world's best Scrabble players believe it is a waste of time (and brain
# power) to learn the definitions.
#
# Given a list of all the 3 letter Scrabble words (3_letter_words.txt) write a
# program to sort the list by the highest scoring words.
#
# The file is in the following format, where the 3 letter word is following by
# a pipe, "|", and then the definition:
#
#  AAH | to exclaim in delight
#  AAL | East Indian shrub
#  AAS | [aa] (rough, cindery lava)
#
# You should create a module named `words` that will handle all the functions
# related to reading, scoring and sorting the list.
```

EXAM REVIEW

PROBLEM 6

```
def read_file(file_name):
    """Read in a file and return a list of all the words"""
    try:
        file_handle = open(file_name, 'r')
    except IOError:
        exit("File not found!")

    words = list()
    for line in file_handle:
        # Word is only first 3 characteristic
        line = line[0:3]
        # Remove all blanks
        line = line.strip()
        line = line.lower()
        if line != "":
            words.append(line)
    return words
```

EXAM REVIEW

PROBLEM 6

```
def score_word(word):
    tile_score = {"a": 1, "c": 3, "b": 3, "e": 1, "d": 2, "g": 2,"f": 4,
        "i": 1, "h": 4, "k": 5, "j": 8, "m": 3, "l": 1, "o": 1, "n": 1,
        "q": 10, "p": 3, "s": 1, "r": 1, "u": 1, "t": 1, "w": 4, "v": 4,
        "y": 4, "x": 8, "z": 10}
    score = 0
    for letter in word:
        score += tile_score[letter]
    return score
```

EXAM REVIEW

PROBLEM 6

```
if __name__ == '__main__':

    # Read in file
    three_letter_words = read_file('3letter_words.txt')

    # Score words
    scored_words = dict()
    for word in three_letter_words:
        score = score_word(word)
        scored_words[word] = score
```

EXAM REVIEW

PROBLEM 6

```
# Track score
    by_score = dict()
    biggest_score = -1
    for word in scored_words:
        score = scored_words[word]
        if biggest_score < score:
            biggest_score = score
        if by_score.has_key(score):
            by_score[score].append(word)
        else:
            by_score[score] = list()
    print "----- By Score ---"
    print by_score[biggest_score]
```

EXAM REVIEW

PROBLEM 6

```
# Brute force

    top_score_words = list()
    biggest_score = max(scored_words.values())
    print "Big",biggest_score
    word_counter = 0

    while word_counter < 20:
        for word in scored_words:
            if scored_words[word] == biggest_score:
                top_score_words.append(word)
                print word,scored_words[word],word_counter
                word_counter += 1
                if word_counter == 20:
                    break
        biggest_score -= 1
```

EXAM REVIEW

PROBLEM 6

```
# Pythonista

for word in sorted(scored_words,key=scored_words.get,reverse=True)[0:20]:
        top_score_words.append(word)
        print("%s -> %d" % (word, scored_words[word]))
```

EXAM REVIEW

PROBLEM 6

```
# Print to file

  fh = open("3letter_words_sorted.txt",'w')
  for word in top_score_words:
      fh.write(word + "\n")
  fh.close()
```

EXAM REVIEW

PROBLEM 7

```
# Problem 7. Bonus Point
#
# What do you get if you divide the circumference
# of a pumpkin by its diameter?
#
```

EXAM REVIEW

PROBLEM 6

```
# Problem 7. Bonus Point
#
# What do you get if you divide the circumference
# of a pumpkin by its diameter?
#
```

Pumpkin Pi

INHERITANCE

INHERITANCE

- One way in which OOP promotes cod reuse is through "inheritance"
- Inheritance is process where a child derives the attributes and methods from a parent class

```
class Parent:
  # attributes
  # methods

class Child(Parent):
  # parents attributes
  # parents methods

  # childs attribues
  # childs methods
```

INHERITANCE

- This can be useful to model real-world relationship and hierarchies

```
Parent: Animals
Child: Dog, Cat, Rabbit

Parent: Person
Child: Student, Teacher
```

INHERITANCE

- Can also be overkill
- Think about the benefits/ disadvantage when designing your classes

```
Parent: Item
Child:  Car, Bread, Skate
```

INHERITANCE

- Bad design

```python
class Car(object):
    """A car for sale by dealership.

    Attributes:
        wheels: An integer representing the number of wheels the car has.
        miles: The integral number of miles driven on the car.
        make: The make of the car as a string.
        model: The model of the car as a string.
        year: The integral year the car was built.
    """

    def __init__(self, wheels, miles, make, model, year):
        """Return a new Car object."""
        self.wheels = wheels
        self.miles = miles
        self.make = make
        self.model = model
        self.year = year

    def __str__(self):
        return "%s %s %d" % (self.make, self.model, self.year)


class Truck(object):
    """A truck for sale by dealership.

    Attributes:
        wheels: An integer representing the number of wheels the truck has.
        miles: The integral number of miles driven on the truck.
        make: The make of the truck as a string.
        model: The model of the truck as a string.
        year: The integral year the truck was built.
```

INHERITANCE

- Working smarter, not harder
- A `Car` is a type of `Vehicle and inherits all its properties
- Add/change only properties that are different

```python
class Vehicle(object):
    """A vehicle for sale by dealership.

    Attributes:
        wheels: An integer representing the number of wheels the car has.
        miles: The integral number of miles driven on the car.
        make: The make of the car as a string.
        model: The model of the car as a string.
        year: The integral year the car was built.
    """
    wheels = 0

    def __init__(self, miles, make, model, year):
        """Return a new Vehicle object."""
        self.miles = miles
        self.make = make
        self.model = model
        self.year = year

    def __str__(self):
        return "%s %s %d" % (self.make, self.model, self.year)


class Car(Vehicle):
    """A Car class"""
    wheels = 4


class Truck(Vehicle):
    """A Truck class"""
    wheels = 4


class Motorcyle(Vehicle):
    """A Motorcyle class"""
    wheels = 2

    def __str__(self):
        """Override the parent implementation of __str__"""
        return "Motorcyles Rule - %s %s %d" % (self.make, self.model, self.year)


t = Truck(0, 'Honda', 'Accord', 2014)
c = Car(2343, 'Toyota', 'Sienna', 2008)
m = Motorcyle(10, 'Indian', "Chieftain",2016)

print t
print c
print m
```

INHERITANCE

```
class Vehicle(object):
    """Base class"""
    wheels = 0

class Car(Vehicle):
    """A Car class"""
    wheels = 4


class Truck(Vehicle):
    """A Truck class"""
    wheels = 4


class Motorcyle(Vehicle):
    """A Motorcyle class"""
    wheels = 2
```

BASECLASS (SUPERCLASS)

SUBCLASSES (DERIVED CLASS)

INHERITANCE

- Not uncommon to declare a base class that will never be used directly
  - Abstract class

```
class Vehicle(object):
    """Base class"""
    wheels = 0

class Car(Vehicle):
    """A Car class"""
    wheels = 4


class Truck(Vehicle):
    """A Truck class"""
    wheels = 4


class Motorcyle(Vehicle):
    """A Motorcyle class"""
    wheels = 2
```

NEVER JUST A VEHICLE

INHERITANCE

```
class Vehicle(object):
    """Base class"""
    wheels = 0

class Car(Vehicle):
    """A Car class"""
    wheels = 4


class Truck(Vehicle):
    """A Truck class"""
    wheels = 4


class Motorcycle(Vehicle):
    """A Motorcycle class"""
    wheels = 2
```

SEE HOW CLASS VARIABLES CAN BE USEFUL

EVERY INSTANCE OF A CLASS OF TRUCK WILL HAVE 4 WHEELS

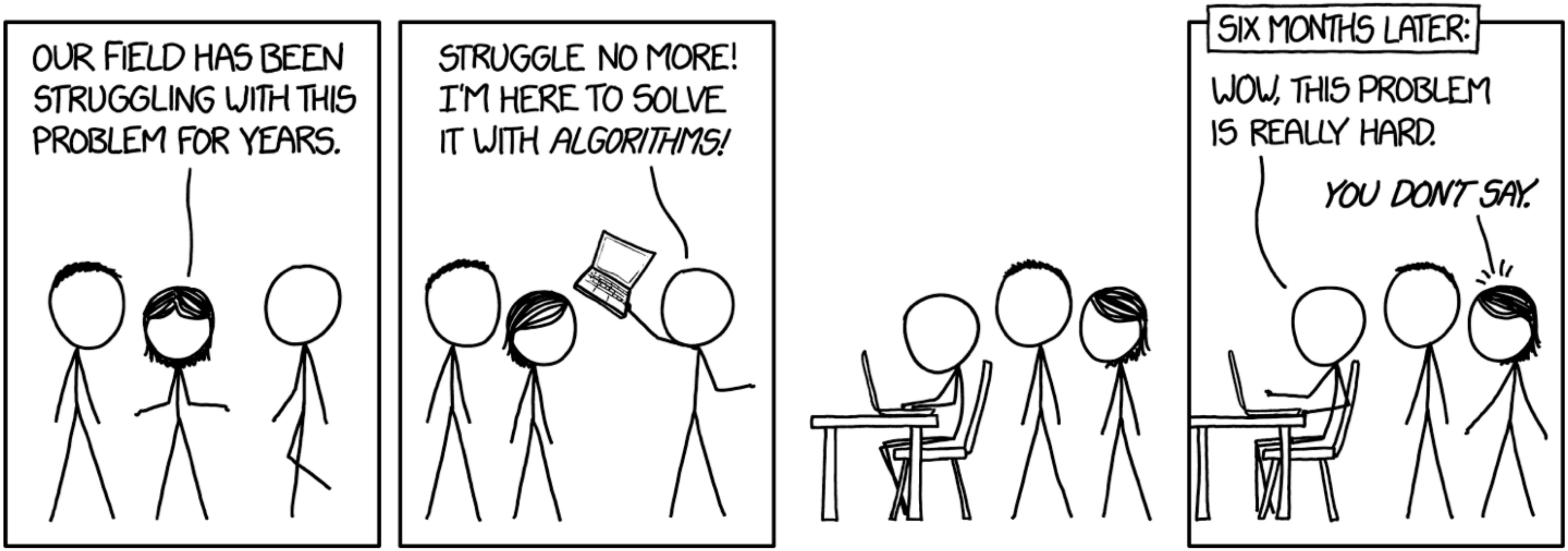
INHERITANCE

- Multiple inheritance exists, but is difficult to get right
- Philosophical differences about the utility

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class SchoolMember(Person):
    pass

class Teacher(SchoolMember):
    pass

class Student(SchoolMember):
    pass

class StudentTeacher(Teacher,Student):
    pass
```

ALGORITHM ANALYSIS

ALGORITHM ANALYSIS



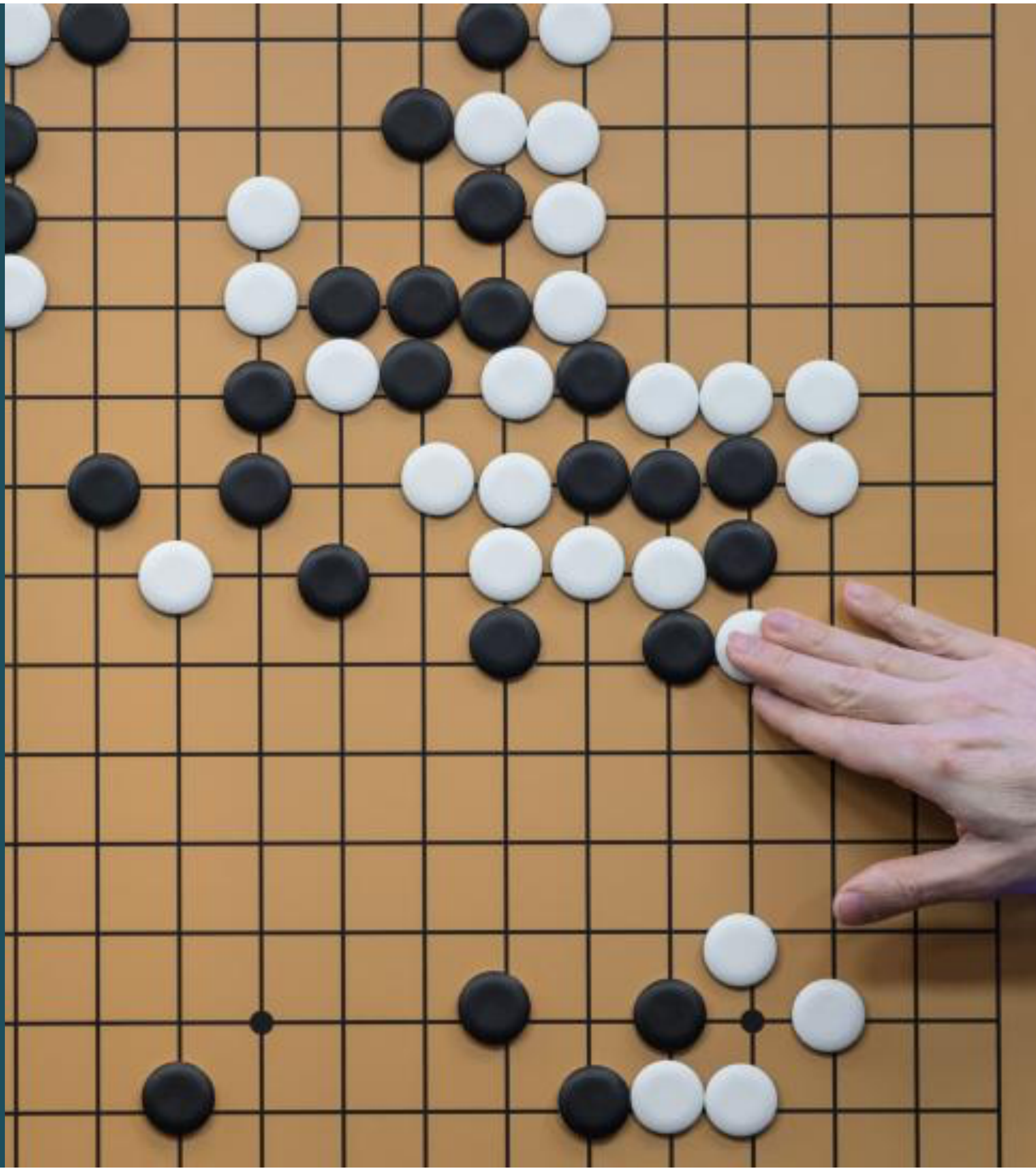
ALGORITHM ANALYSIS

- Algorithm is step-by-step instructions for solving a problem
  - Making a PB&J sandwich
  - Driving directions
  - Sorting
  - Sum a list of numbers

```python
# Algorithm to sum a range of
# numbers from 0 to n
def sum_list(n):
    sum = 0
    for i in range(1,n+1):
        sum = sum + i
    return sum


print(sum_list(10))
```

ALGORITHM ANALYSIS

- An algorithm is NOT a program
- **Program** is <u>an implementation of an algorithm</u> in a language that can be interpreted and executed by a computer



ALGORITHM ANALYSIS

- A program may use <u>many</u> algorithms to solve a given problem

```
def process_file(file):?
 # Read a file
 # sort the file
 # identify duplicates
 # write to a file
```

ALGORITHM ANALYSIS

- A "robust" algorithm should
  - Solve the problem for all sets of inputs and outputs
  - Handle all error conditions
  - Be deterministic (same output for same set of inputs)

ALGORITHM ANALYSIS

- Attributes and considerations for algorithms
  - Correctness - It provides a correct solution to the problem

ALGORITHM ANALYSIS

- Attributes and considerations for algorithms
  - Efficiency
    - Time: How long does it take to solve the problem?
    - Space: How much memory is needed?
    - Benchmarking vs. Analysis: Real world performance

ALGORITHM ANALYSIS

- Attributes and considerations for algorithms
  - Ease of understanding
    - Program maintenance

ALGORITHM ANALYSIS

- Attributes and considerations for algorithms
  - Elegance 😇

ALGORITHM ANALYSIS

- For a given problem, there are many different algorithms to solve the same problem
- Which one is the "best"?
- How do we measure this?
  - Running time? Machine dependent!
  - Number of steps?

Contents [hide]

ALGORITHM ANALYSIS

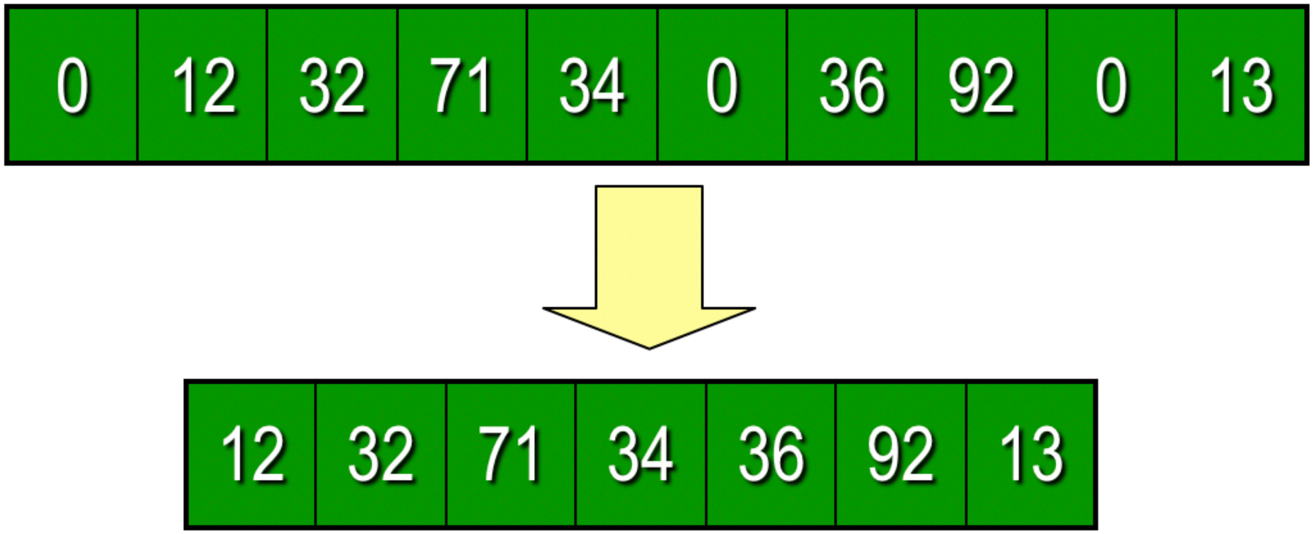
- Algorithm analysis is concerned with comparing algorithms based upon the amount of computing resources that each algorithm uses
  - Time, money, humans, …
- Provide a way to compare algorithms to make educated decisions

Contents [hide]

EXAMPLE: THE DATA CLEANUP PROBLEM

DATA CLEANUP PROBLEM

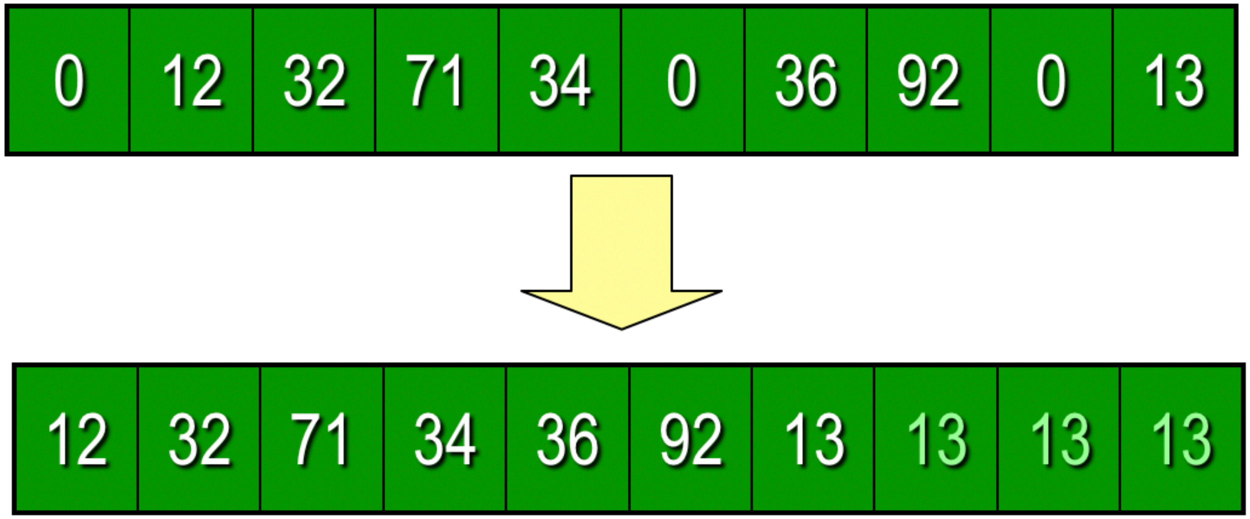
- Remove "0" from a list of numbers



DATA CLEANUP PROBLEM

SHUFFLE LEFT ALGORITHM

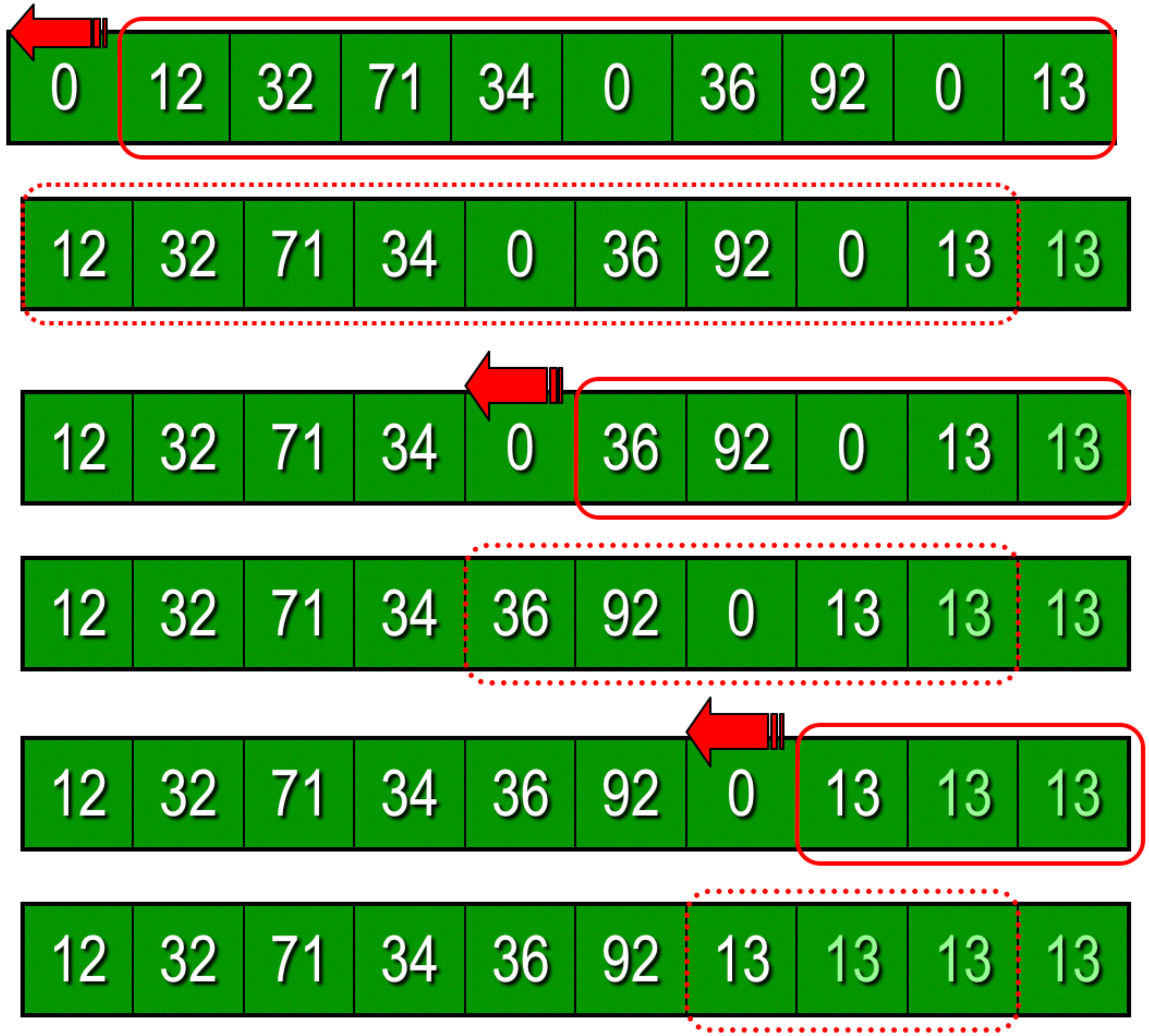
- Scan the list from left to right
- Whenever we encounter a 0 element we copy ("shuffle") the rest of the list one position left



DATA CLEANUP PROBLEM

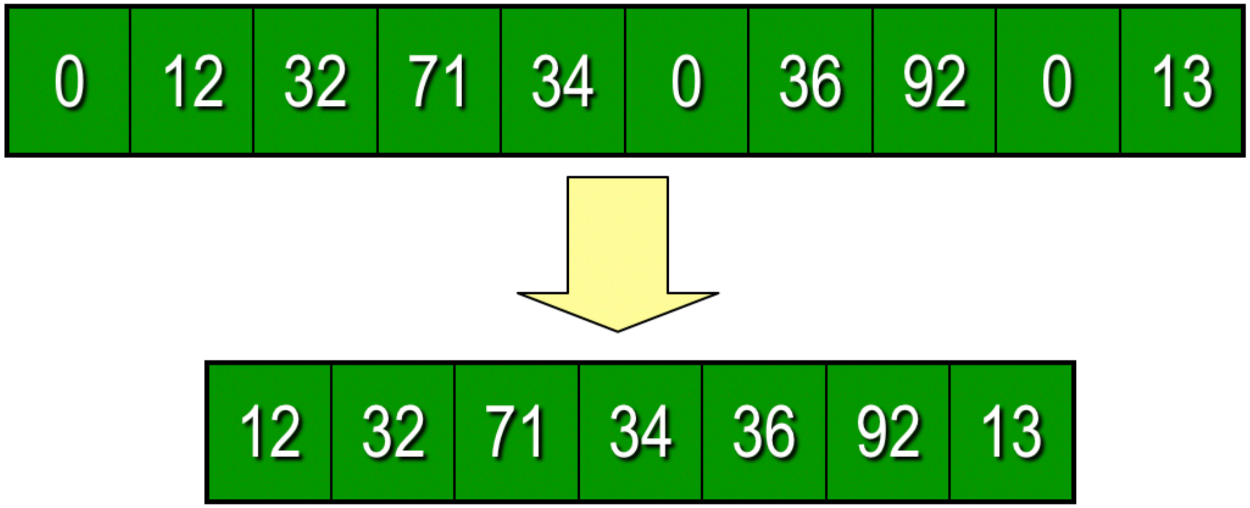
SHUFFLE LEFT ALGORITHM

- Scan the list from left to right
- Whenever we encounter a 0 element we copy ("shuffle") the rest of the list one position left



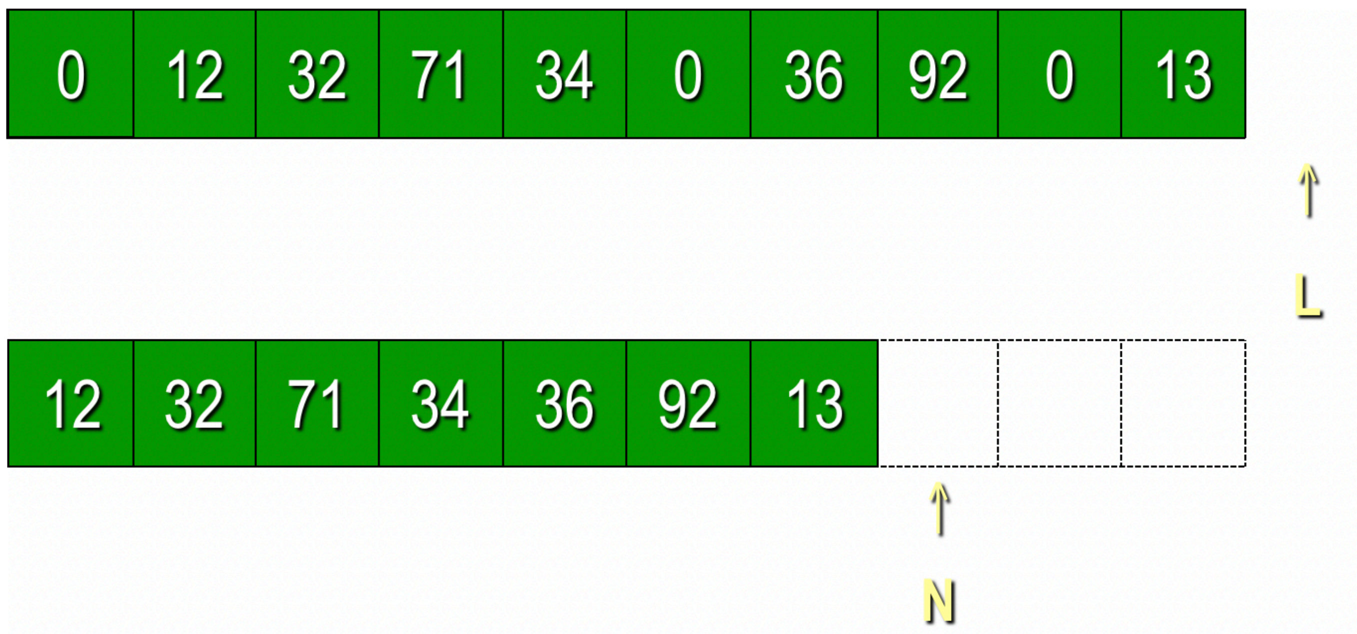
DATA CLEANUP PROBLEM
COPY-OVER ALGORITHM

- Scan the list from left to right
- Whenever we encounter a nonzero element we copy it over to a new list



DATA CLEANUP PROBLEM
COPY-OVER ALGORITHM

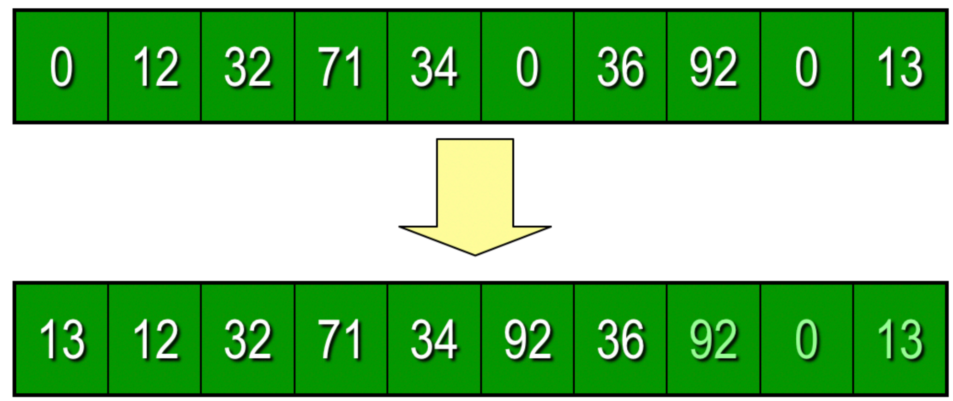
- Scan the list from left to right
- Whenever we encounter a nonzero element we copy it over to a new list



DATA CLEANUP PROBLEM

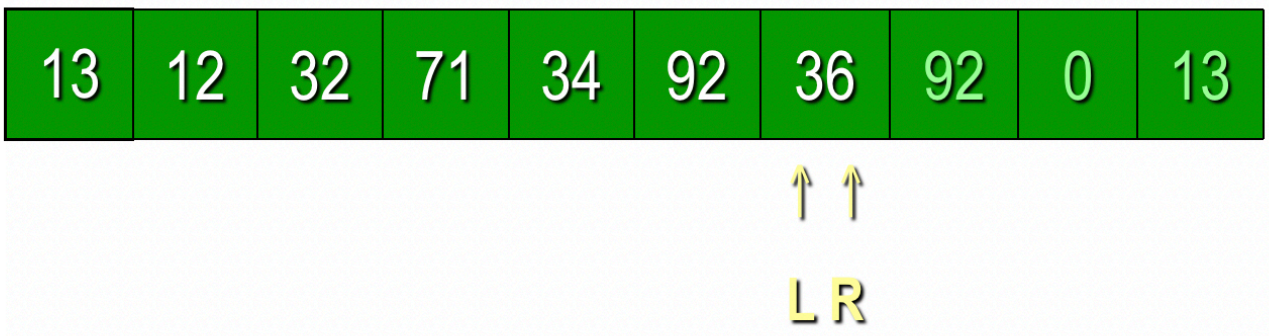
CONVERGING-POINTERS ALGORITHM

- Scan the list from both left (L) and right (R)
- Whenever L encounters a 0 element, the element at location R is copied to location L, then R reduced



DATA CLEANUP PROBLEM

CONVERGING-POINTERS ALGORITHM



- Scan the list from both left (L) and right (R)
- Whenever L encounters a 0 element, the element at location R is copied to location L, then R reduced

DATA CLEANUP PROBLEM

- Which one is the most space efficient?
- Which one is the most time efficient?

DATA CLEANUP PROBLEM

for i in 0...100:	Space	Time
Shuffle Left	Constant	Go Through List Many Times
Copy Over	Double	Go Through List Once
Converging Pointers	Constant	Go Through List Once

MEASURING RESOURCES

MEASURING EFFICIENCY

- Metric to measure time efficiency of algorithms
  - How long does it take to solve the problem?
    - Depends on machine speed
  - How many steps does the algorithm execute?
    - Need to count all steps

MEASURING EFFICIENCY

- How many "fundamental steps" does the algorithm execute?
- Depends on size and type of input, interested in knowing
  - Best-case, Worst-case, Average-case behavior

MEASURING RESOURCES

SEQUENTIAL SEARCH ALGORITHM

Find a name in array of strings

```
1. Get values for Name, N1,…, Nn
2. Set the value i to 1 and set the value of Found to NO
3. Repeat steps 4 through 7 until Found = YES or i > n
4.      If Name = Ni then
5.            Print the value of N
6.            Set the value of Found to YES
        Else
7.            Add 1 to the value of i
8. If Found = NO then print "Sorry, name not in directory"
9. Stop
```

MEASURING RESOURCES

SEQUENTIAL SEARCH ALGORITHM

Find a name in array of strings

```
def sequential_search(name, names):
    found = False
    i = 0
    n = len(name)
    while found is False or i > n:
        if name == names[i]:
            found = True
            print "Found"
        i = i + 1
    if found == False:
        print "Sorry, the name is not there"


names = ["pebbles", "lola", "chloe"]
sequential_search("lola", names)
```

MEASURING EFFICIENCY

SEQUENTIAL SEARCH ALGORITHM

- How many steps does the algorithm execute?
  - Steps 2, 5, 6, 8 and 9 are executed at most once.
  - Steps 3, 4, and 7 depends on input size.
- Worst case:
  - Step 3, 4, and 7 are executed at most n-times.
- Best case:
  - Step 3 and 4 are executed only once.
- Average case:
  - Step 3, 4 are executed approximately (n/2)-times.

1. Get values for *Name*, $N_1, \ldots, N_n$, $T_1, \ldots, T_n$
2. Set the value *i* to 1and set the value of *Found* to NO
3. Repeat steps 4 through 7 until *Found* = YES or $i > n$
4.   If *Name* = N_i then
5.   Print the value of T_i
6.   Set the value of *Found* to YES

       Else
7.   Add 1 to the value of *i*
8. If Found = NO then print "Sorry, name not in directory"
9. Stop

MEASURING EFFICIENCY

SEQUENTIAL SEARCH ALGORITHM

- How many steps does the algorithm execute?
  - Steps 2, 5, 6, 8 and 9 are executed at most once.
  - Steps 3, 4, and 7 depends on input size.
- Worst case:
  - Step 3, 4, and 7 are executed at most n-times.
- Best case:
  - Step 3 and 4 are executed only once.
- Average case:
  - Step 3, 4 are executed approximately (n/2)-times.

1. Get values for *Name*, $N_1, \ldots, N_n$, $T_1, \ldots, T_n$
2. Set the value i to 1land set the value of *Found* to NO
3. Repeat steps 4 through 7 until *Found* = YES or $i > n$
4.   If *Name* = N_i then
5.   Print the value of T_i
6.   Set the value of *Found* to YES

       Else
7.   Add 1 to the value of i
8. If Found = NO then print "Sorry, name not in directory"
9. Stop

MEASURING EFFICIENCY

SEQUENTIAL SEARCH ALGORITHM

- How many steps does the algorithm execute?
  - Steps 2, 5, 6, 8 and 9 are executed at most once.
  - Steps 3, 4, and 7 depends on input size.
- Worst case:
  - Step 3, 4, and 7 are executed at most n-times
- Best case:
  - Step 3 and 4 are executed only once
- Average case:
  - Step 3, 4 are executed approximately (n/2)-times

```
1. Get values for Name, N_1,…, N_n, T_1,…, T_n
2. Set the value i to 1and set the value of Found to NO
3. Repeat steps 4 through 7 until Found = YES or i > n
4.      If Name = N_i then
5.             Print the value of T_i
6.             Set the value of Found to YES
        Else
7.             Add 1 to the value of i
8. If Found = NO then print "Sorry, name not in directory"
9. Stop
```

THE NAME IS LAST

MEASURING EFFICIENCY

SEQUENTIAL SEARCH ALGORITHM

- How many steps does the algorithm execute?
  - Steps 2, 5, 6, 8 and 9 are executed at most once.
  - Steps 3, 4, and 7 depends on input size.
- Worst case:
  - Step 3, 4, and 7 are executed at most n-times
- Best case:
  - Step 3 and 4 are executed only once
- Average case:
  - Step 3, 4 are executed approximately (n/2)-times

```
1. Get values for Name, N1,…, Nn, T1,…, Tn
2. Set the value i to 1and set the value of Found to NO
3. Repeat steps 4 through 7 until Found = YES or i > n
4.      If Name = Ni then
5.             Print the value of Ti
6.             Set the value of Found to YES
        Else
7.             Add 1 to the value of i
8. If Found = NO then print "Sorry, name not in directory"
9. Stop
```

THE NAME IS FIRST

MEASURING EFFICIENCY

SEQUENTIAL SEARCH ALGORITHM

- How many steps does the algorithm execute?
  - Steps 2, 5, 6, 8 and 9 are executed at most once.
  - Steps 3, 4, and 7 depends on input size.
- Worst case:
  - Step 3, 4, and 7 are executed at most n-times
- Best case:
  - Step 3 and 4 are executed only once
- Average case:
  - Step 3, 4 are executed approximately (n/2)-times

```
1. Get values for Name, N1,…, Nn, T1,…, Tn
2. Set the value i to 1and set the value of Found to NO
3. Repeat steps 4 through 7 until Found = YES or i > n
4.        If Name = Ni then
5.                  Print the value of Ti
6.                  Set the value of Found to YES
          Else
7.                  Add 1 to the value of i
8. If Found = NO then print "Sorry, name not in directory"
9. Stop
```

THE NAME IS RANDOM

MEASURING RESOURCES

SEQUENTIAL SEARCH ALGORITHM

```
def sequential_search(name, names):
    found = False
    i = 0
    n = len(name)
    while found is False or i > n:
        if name == names[i]:
            found = True
            print "Found"
        i = i + 1
    if found == False:
        print "Sorry, the name is not there"


names = ["pebbles", "lola", "chloe"]
sequential_search("lola", names)
```

WHAT IF THIS WAS REALLY LONG LIST?

MEASURING EFFICIENCY

- Not interested in knowing the exact number of steps the algorithm performs
- Mainly interested in knowing how the number of steps grows with increased input size
- Why?
  - Given large enough input (∞), the algorithm with faster growth will execute more steps

MEASURING RESOURCES

- Algorithm: Sum of a range of integers
- The time grows proportional to the number of inputs

```
def sumOfN2(n):
    start = time.time()
    theSum = 0
    for i in range(1, n+1):
        theSum = theSum + i
    end = time.time()
    return theSum, end-start


for i in range(0, 1000000, 100000):
    sum, duration = sumOfN2(i)
    print("%d: Sum is %d required
%10.7f seconds" % (i, sum, duration))
```

MEASURING RESOURCES

```python
def sumOfN2(n):
    """Sum through iteration."""
    start = time.time()          # 1 time
    theSum = 0                   # 1 time
    for i in range(1, n+1):      # n times
        theSum = theSum + i      # 1 times
    end = time.time()            # 1 time
    return theSum, end-start     # 1 time
```

MEASURING RESOURCES

```
def sumOfN2(n):
    start = time.time()
    theSum = 0
    for i in range(1, n+1):
        theSum = theSum + i
    end = time.time()
    return theSum, end-start


for i in range(0, 1000000, 100000):
    sum, duration = sumOfN2(i)
    print("%d: Sum is %d required %10.7f seconds" % \
          (i, sum, duration))
```

```
0: Sum is 0 required  0.0000031 seconds
100000: Sum is 5000050000 required  0.0190489 seconds
200000: Sum is 20000100000 required  0.0326600 seconds
300000: Sum is 45000150000 required  0.0464170 seconds
400000: Sum is 80000200000 required  0.0582540 seconds
500000: Sum is 125000250000 required  0.0717301 seconds
600000: Sum is 180000300000 required  0.0905130 seconds
700000: Sum is 245000350000 required  0.1016700 seconds
800000: Sum is 320000400000 required  0.1161430 seconds
900000: Sum is 405000450000 required  0.1323390 seconds
```

MEASURING EFFICIENCY

- Algorithm: Sum of a range of integers
- The time is constant for any number of inputs

```
def sumOfN3(n):
    """With closed equation."""
    start = time.time()           # 1 time
    theSum = n * (n + 1) / 2      # 1 time
    end = time.time()             # 1 time
    return theSum, end-start      # 1 time
```

```
print("Sum is %d required %10.7f seconds" % sumOfN3(100))
print("Sum is %d required %10.7f seconds" % sumOfN3(10000))
print("Sum is %d required %10.7f seconds" %
sumOfN3(10000000))

# Sum is 5050 required  0.0000021 seconds
# Sum is 50005000 required  0.0000010 seconds
# Sum is 50000000500000000 required  0.0000000 seconds
```

MEASURING EFFICIENCY

- To compare algorithms we are concerned with their relative rate of growth
- Order of magnitude, O(f(n)) measures how the number of steps grows with input size n
  - For dominant part of the algorithm
- Referred to as "Big-O" notation

MEASURING EFFICIENCY

- Not interested in the exact number of steps, for example, algorithm where total steps, T(n)
  - T(n)
  - T(5n)
  - T(5n+345)
  - T(4500n+1000)

T(N) HAS AN ORDER OF MAGNITUDE F(N)

OR

O(N)

- For all the above algorithms, the total number of steps grows approx. proportionally with input size (given large enough n)

THE END

MPCS 50101
AUTUMN 2019
SESSION 12

THE UNIVERSITY OF CHICAGO