Debugging&Testing Guidelines

CS519-005 Algorithms, Fall 2016

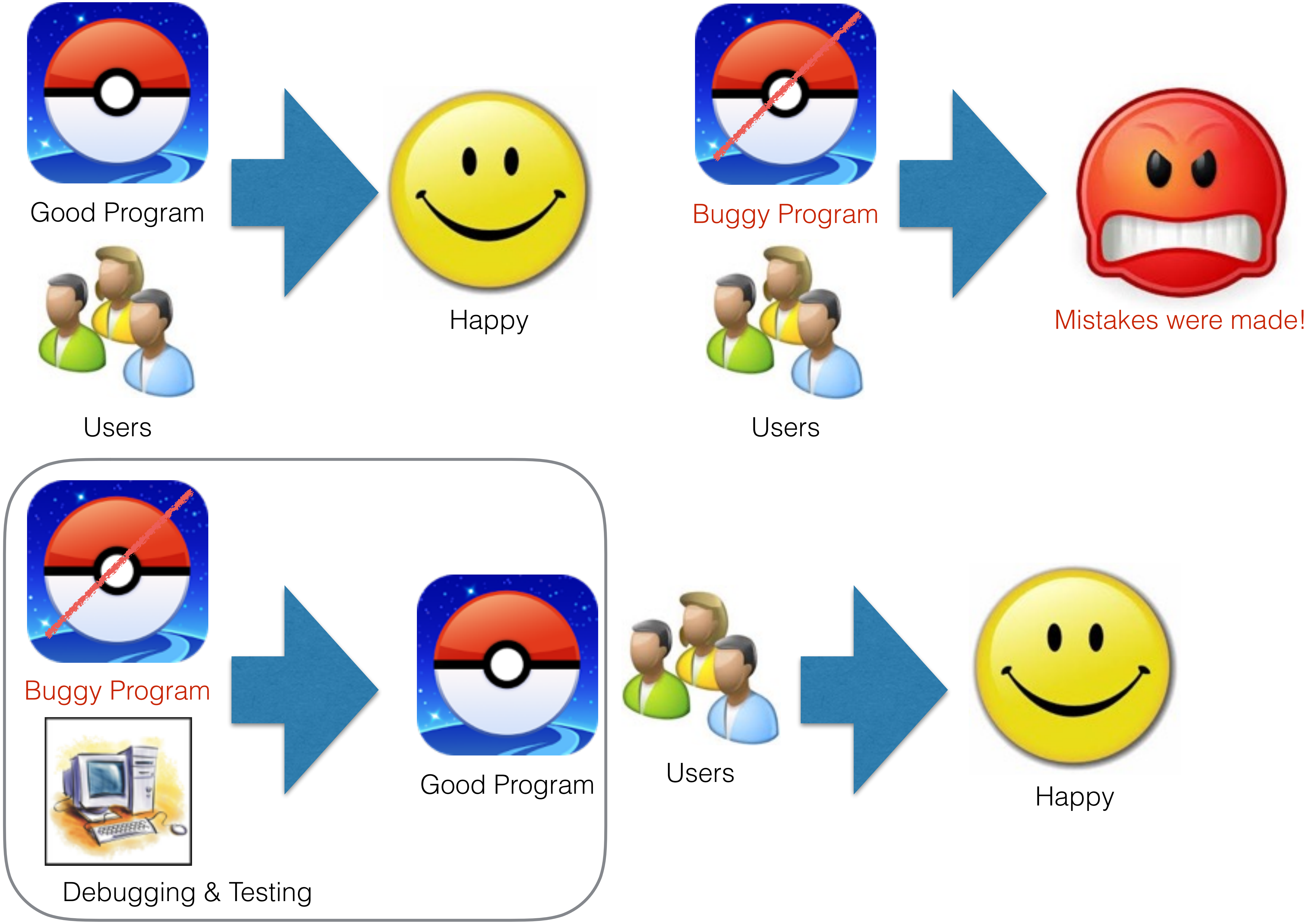
http://classes.engr.oregonstate.edu/eecs/fall2016/cs519-005/

Dezhong Deng

Oregon State University

dengdezhong816@gmail.com

Debugging/Testing is important!



Good Program

Users

Happy

Buggy Program

Users

Mistakes were made!

Buggy Program

Debugging & Testing

Good Program

Users

Happy

Try it without debugger

- Using IDE, or other debuggers can be great, but
  - debuggers like IDE run everything in real time
    - halt the program / one step after another
    - restart if you want to go back
    - much time-consuming
- Your brain explores multiple code paths at one time!
  - you always want a whole picture on 'how the current program is working'
  - faster and more efficient debugging/testing is needed!
- This guideline is focused on debugger-free debugging/testing!

Start with Print function

What is ‘range’?

```
sum = 0
print "range(10): ", range(10)
for i in range(10):
    sum += i
print sum
```

```
range(10):  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
45
```

How does ‘fibonacci’ work?

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        print "b =", b
        a, b = b, a+b
    return b
print fibonacci(5)
```

```
b = 1
b = 1
b = 2
b = 3
b = 5
8
```

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        print "b =", b
        a, b = b, a+b
    return b
print "result =", fibonacci(5)
```

What will happen?

```
result = b = 1
b = 1
b = 2
b = 3
b = 5
8
```

```
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        print "b =", b
        a, b = b, a+b
    return b
result = fibonacci(5)
print "result =", result
```

See how recursion works

```
import random
def qsort(a, level = 0):
    funcid = random.randint(0,10000)
    if a == []:
        return []
    else:
        print "%s ID:%d\t qsort(%s)" % \
            (" |" * level, funcid, str(a))
        pivot = a[0]
        left = [x for x in a if x < pivot]
        right = [x for x in a[1:] if x >= pivot]
        result = qsort(left, level+1) + [pivot] + \
                 qsort(right, level+1)
        print "%s ID:%d\t returns(%s)" % \
            (" |" * level, funcid, str(result))
        return result
```

```
>>> qsort([6,2,8,4,1,9,7])
 ID:4247  qsort([6, 2, 8, 4, 1, 9, 7])
 | ID:7905    qsort([2, 4, 1])
 | | ID:3447  qsort([1])
 | | ID:3447  returns([1])
 | | ID:2168  qsort([4])
 | | ID:2168  returns([4])
 | ID:7905    returns([1, 2, 4])
 | ID:7876    qsort([8, 9, 7])
 | | ID:6430  qsort([7])
 | | ID:6430  returns([7])
 | | ID:5464  qsort([9])
 | | ID:5464  returns([9])
 | ID:7876    returns([7, 8, 9])
 ID:4247  returns([1, 2, 4, 6, 7, 8, 9])
[1, 2, 4, 6, 7, 8, 9]
>>> python
```

What does ‘level’ mean?

Recursion depth!

Recursion depth

Print your recursion depth

```
from random import randint
max_depth = 0
def qselect(k, a, depth = 0):
    global max_depth
    if depth > max_depth:
        max_depth = depth
    if a == [] or k < 1 or k > len(a):
        return None
    else:
        pindex = randint(0, len(a)-1)
        a[0],a[pindex] = a[pindex],a[0]
        pivot = a[0]
        left = [x for x in a if x < pivot]
        right = [x for x in a[1:] if x >= pivot]
        lleft = len(left)
        return pivot if k == lleft+1 else \
            qselect(k, left, depth+1) if k <= lleft else \
            qselect(k-lleft-1, right, depth+1)

result = qselect(3, range(10000))
print max_depth
```

Or simply set recursion limit

```
from contextlib import contextmanager
from sys import getrecursionlimit
from sys import setrecursionlimit
@contextmanager
def recursionlimit(n=1000):
    rec_limit = getrecursionlimit()
    setrecursionlimit(n)
    yield
    setrecursionlimit(rec_limit)
```

Verifying your calculation

- Verifying a calculation can be hard in some questions
  - but it could be simple to write an equivalent approach!
    - although it might be slower, it could help you on verifying the calculation of your original program

```
Q: Calculate the number of inversions in a list.
   Must run in O(nlogn) time.
```

Simple O(n^2) method can be used of verifying O(n log(n)) method!

```
def num_inversions(lst):
    result = 0
    l = len(lst)
    for i in xrange(l):
        for j in xrange(i+1, l):
            if lst[i] > lst[j]:
                result += 1
    return result
```

```
>>> num_inversions([4, 1, 3, 2])
4
>>> num_inversions([2, 4, 1, 3])
3
>>> num_inversions([4, 3, 2, 1])
6
```

What about x+y=z problem in HW3?

Creating test cases

basic cases for a sort function

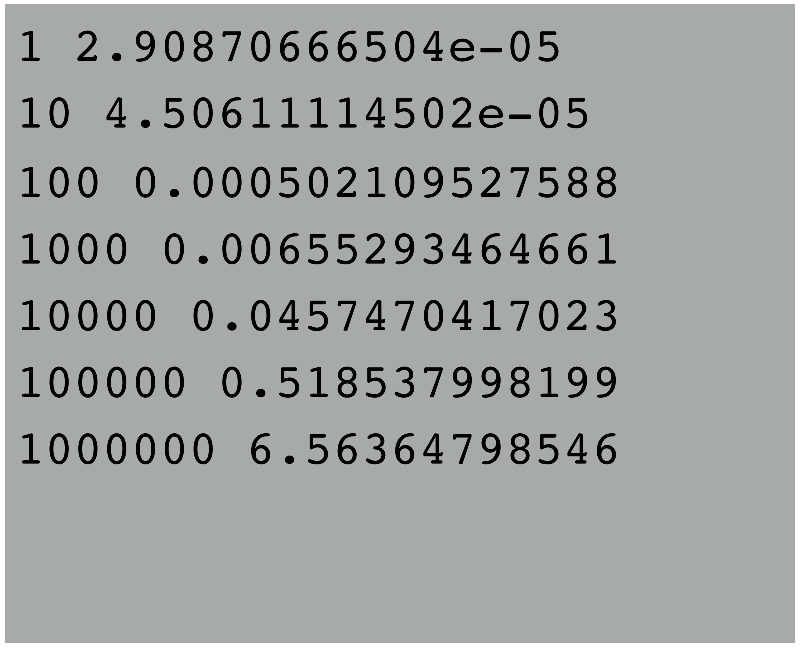
```
import random
print sort([])
print sort([1])

## rev_lst: reversed list
rev_lst = range(20)
rev_lst.reverse()
print sort(rev_lst)

## lst: unsorted list without duplicates
lst = range(20)
random.shuffle(lst)
print lst
print sort(lst)

## lst2: unsorted list with duplicates
def getrandomlist(n):
    return [random.choice(range(1000)) for _ in xrange(n)]
lst2 = getrandomlist(20)
print lst2
print sort(lst2)
```

- Most important part for debugging/testing
- Designing smart cases makes debugging/testing much more efficient!

Testing time complexity

O(n log(n)) time?

```
import time, random
n = 1
for _ in xrange(7):
    lst = range(n)
    random.shuffle(lst)
    starttime = time.time()
    result = mergesort(lst)
    print n, time.time()-starttime
    n *= 10
```



```
1 2.90870666504e-05
10 4.50611114502e-05
100 0.000502109527588
1000 0.00655293464661
10000 0.0457470417023
100000 0.518537998199
1000000 6.56364798546
```

Yes!

- Usually a laptop can do 10^7 to 10^8 operations in one second
    - and you can always test it
- Plot your result in log-scale to prove your thoughts!
    - gnuplot, python matplotlib, …

Thanks! Questions?

Dezhong Deng

Oregon State University

dengdezhong816@gmail.com