Lab 1
Analysis and thoughts
The algorithm

Given: an integer
Return: true iff integer is odd, else return false

function isOdd(n: integer)
  if (n % 2) equals 1
    return true
  else
    return false

• Manipulatable?
• Data transform?
Implementations

Go

```go
package main

import "fmt"

func isOdd(v int) bool {
    return v%2 == 1
}

func main() {
    for i := 0; i < 5; i++ {
        fmt.Printf("%d  %v\n", i, isOdd(i))
    }
}
```

Java

```java
public class V0 {
    public static boolean isOdd(int i) {
        return i % 2 != 0;
    }

    public static void main(String[] args) {
        V0 v0 = new V0();
        for (int i = 0; i < 5; i++) {
            System.out.format("%d %b\n", i, isOdd(i));
        }
    }
}
```

Python

```python
def isOdd(x):
    return x % 2 == 1

for k in range(5):
    print(k, isOdd(k))
```
Results

Go

0..5
0 false
1 true
2 false
3 true
4 false

-5..-5
-5 false
-4 false
-3 false
-2 false
-1 false
0 false
1 true
2 false
3 true
4 false

Java

0..5
0 false
1 true
2 false
3 true
4 false

-5..-5
-5 false
-4 false
-3 false
-2 false
-1 false
0 false
1 true
2 false
3 true
4 false

Python

0..5
0 False
1 True
2 False
3 True
4 False

-5..-5
-5 True
-4 False
-3 True
-2 False
-1 True
0 False
1 True
2 False
3 True
4 False
• Results
  • Agreement on positive numbers
  • Java and Go differ from Python
    • Python seems correct
• Hypotheses
  • Java and Go reject concept of modulus on negative numbers
  • Java and Go implement modulus differently from Python
• H1: The "undefined" hypothesis.
• Test: if % undefined for negative integers in Java, then a runtime exception would occur.
• Rejected: no such exception occurs
• H2: % is different in Python and Go/Java for negative numbers
  • Test: Write a program to show x%5 for the numbers -10..0 (hard to see with %2)

<table>
<thead>
<tr>
<th>Go</th>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10 0</td>
<td>-10 0</td>
<td>-10 0</td>
</tr>
<tr>
<td>-9 -4</td>
<td>-9 -4</td>
<td>-9 1</td>
</tr>
<tr>
<td>-8 -3</td>
<td>-8 -3</td>
<td>-8 2</td>
</tr>
<tr>
<td>-7 -2</td>
<td>-7 -2</td>
<td>-7 3</td>
</tr>
<tr>
<td>-6 -1</td>
<td>-6 -1</td>
<td>-6 4</td>
</tr>
<tr>
<td>-5 0</td>
<td>-5 0</td>
<td>-5 0</td>
</tr>
<tr>
<td>-4 -4</td>
<td>-4 -4</td>
<td>-4 1</td>
</tr>
<tr>
<td>-3 -3</td>
<td>-3 -3</td>
<td>-3 2</td>
</tr>
<tr>
<td>-2 -2</td>
<td>-2 -2</td>
<td>-2 3</td>
</tr>
<tr>
<td>-1 -1</td>
<td>-1 -1</td>
<td>-1 4</td>
</tr>
</tbody>
</table>

• Conclusion: Python is Different!!!
Modulus ==? Modulus

- **Knuth**: \( \text{mod}(a, n) = a - n \times \text{floor}(a / n) \)

- **Floor?**

<table>
<thead>
<tr>
<th>Go</th>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10</td>
<td>-10</td>
<td>-10</td>
</tr>
<tr>
<td>-9</td>
<td>-9</td>
<td>-9</td>
</tr>
<tr>
<td>-8</td>
<td>-8</td>
<td>-8</td>
</tr>
<tr>
<td>-7</td>
<td>-7</td>
<td>-7</td>
</tr>
<tr>
<td>-6</td>
<td>-6</td>
<td>-6</td>
</tr>
<tr>
<td>-5</td>
<td>-5</td>
<td>-5</td>
</tr>
<tr>
<td>-4</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>-3</td>
<td>-3</td>
<td>-3</td>
</tr>
<tr>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Conclusions

• Go uses a different definition of floor than Java or Python
• Java and Python use same definition of floor
• Go and Python are consistent with Knuth given their different definitions of floor
• Java is not consistent with Knuth! (but is consistent with Go)
Fixed Algorithm

instead of checking for 1, check for 0

Given: an integer

Return: true iff integer is odd, else return false

function isOdd(n: integer)

if (n % 2) equals 0
    return false
else
    return true
Part 2: Bitwise Operators

- Bitwise Operators are binary operations that work on the binary representation of a number.
- There are two main operators (same in Go, Java and Python)
  - & the result has a 1 in a bit position if both numbers have a 1 in that position
  - | the result has a 1 in a bit position if either number has a 1 in that position

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Decimal</th>
<th>Binary</th>
<th>Decimal</th>
<th>Binary</th>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>b10001</td>
<td>17</td>
<td>b10001</td>
<td>23</td>
<td>b10111</td>
<td>28</td>
<td>b11100</td>
</tr>
<tr>
<td>3</td>
<td>b00011</td>
<td>3</td>
<td>b00011</td>
<td>1</td>
<td>b00001</td>
<td>1</td>
<td>b00001</td>
</tr>
<tr>
<td>1</td>
<td>b00001</td>
<td>19</td>
<td>b10011</td>
<td>1</td>
<td>b00001</td>
<td>0</td>
<td>b00000</td>
</tr>
</tbody>
</table>
Binary Negative Numbers

- Most / all PLs used "2s complement" to represent signed integers
- Use the leftmost bit to indicate sign
- Negative number take the complement of positive numbers
- Importantly, odd negative numbers have a 1 in the rightmost bit, just like positive
- So, for positive and negative can determine oddness as "x&1==1"

<table>
<thead>
<tr>
<th></th>
<th>Positive</th>
<th>Negative</th>
<th>1's Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b00000001</td>
<td>-1</td>
<td>b11111111</td>
</tr>
<tr>
<td>2</td>
<td>b00000010</td>
<td>-2</td>
<td>b11111110</td>
</tr>
<tr>
<td>3</td>
<td>b00000011</td>
<td>-3</td>
<td>b11111101</td>
</tr>
<tr>
<td>4</td>
<td>b00000100</td>
<td>-4</td>
<td>b11111010</td>
</tr>
</tbody>
</table>
Modulus using Bitwise Operator

Given: an integer
Return: true iff integer is odd, else return false

function isOdd(n: integer)
if (n & 1) == 1
  return true
else
  return false
Part 3: Which is better?

• Idea, do a LOT of modulus operations and time how long that takes
Part 2: Timing Modulus

• Use the Unix "time" function

• For example:
  javac M.java
time java M

time go run M.go

time python M.py

• Problems??
public class V1 {
    public static boolean isOddBit(int i) {
        return (i & 1) == 1;
    }
    public static boolean isOdd(int i) {
        return (i % 2) != 0;
    }
    public static void timer() {
        long st = System.nanoTime();
        int odd = 0;
        for (int j = 0; j < 10000000; j++) {
            for (int i = -100; i < 100; i++) {
                if (isOddBit(i))
                    odd++;
            }
        }
        long en = System.nanoTime();
        System.out.println((en - st) / 1000000.0);
    }
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++)
            timer();
    }
}

• Timing:
  • As close to the problem as you can get
  • Read system clock before you start
  • Read system clock when you finish
  • subtract
• Why do the odd calculation $2 \times 10^8$ times?
• Why do bit and mod as a pair?
• Sequentially or in totally separate programs?
• Other problems??
public class V1 {
    public static boolean isOddBit(int i) {
        return (i & 1) == 1;
    }
    public static boolean isOdd(int i) {
        return (i % 2) != 0;
    }
    public static void timer() {
        long st = System.nanoTime();
        int odd = 0;
        for (int j = 0; j < 1000000; j++) {
            for (int i = -100; i < 100; i++) {
                if (isOddBit(i))
                    odd++;
            }
        }
        long en = System.nanoTime();
        System.out.println((en - st) / 1000000.0 + " ");
    }
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++)
            timer();
    }
}
func timeit(ff func(int)bool, ss string) {
    start := time.Now()
    odd:=0
    for i:=0; i<1000000; i++ {
        for j:=-100; j<=100; j++ {
            if ff(j) {
                odd++
            }
        }
    }
    duration := time.Since(start)
    fmt.Printf("%v %v ", ss, duration)
}

Python

st = time.time()
odd=0
for i in range(1000000):
    for j in range(-100, 100):
        if isOddBit(i):
            oddd += 1
et = time.time()
elapsed_time = et - st
print('Execution time (bit):', elapsed_time, 'seconds')

Java

```java
long st = System.nanoTime();
int odd = 0;
for (int j = 0; j < 10000000; j++) {
    for (int i = -100; i < 100; i++) {
        if (isOddBit(i))
            odd++;
    }
}
long en = System.nanoTime();
System.out.print((en - st) / 1000000.0 + "; ");
```
All trials in a language are done using a single unix command.

Tentative conclusions:

- Python is a lot slower.
- Bitwise odd is slower than modulus odd (at least for Python).
- Need a better look at Java and Go.

Each trial is 1,000,000 runs over range -100..100.
Same data as previous chart, just eliminating python

Tentative conclusions:

- bitwise odd is slower than modulus odd -- at least for Java
- Java has something weird at startup
  - Java oddness happens at every startup!
    - (data not shown)
Laptops do lots of things
As a result, times are not consistent
Odd Times for Go

<table>
<thead>
<tr>
<th>Description</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac -- Mod</td>
<td>12.06</td>
</tr>
<tr>
<td>Mac -- Bit</td>
<td>13.57</td>
</tr>
<tr>
<td>Comet -- Mod</td>
<td>1.01</td>
</tr>
<tr>
<td>Comet -- Bit</td>
<td>1.19</td>
</tr>
</tbody>
</table>

each trial is 1,000,000 runs over range -100..100
Winston on Presentations

Pick

- Time
- The room
  - Shape matters (Park 227, Park 338)
  - A happy place
Practice

• Pick your location
• AV issues
• Lights on
• Chat up early arrivers
The talk

• Be Happy
• VSN-C
  • Start with Vision
  • Steps
  • News
• Finish with Contributions
Contributions == Conclusions

- No "thank you"
- No collaborators
  - if needed, do early
"you have too many slides and all of them have too many words"

Winston
Do not read
No cute clip art
Avoid bullet lists
Use big fonts

(use even bigger fonts)
Progress bars -- maybe

"page 1 or 12"?
Um like er... you know
Bellow!
( use a mic, practice)

Monotone

Pockets
• Given an array, $A$ of $n$ integers arranged in ascending order, and an integer $x$:

$$\text{search}(A, n, x) = \begin{cases} 
  i, & \text{such that } A[i] = n \\
  -1, & \text{otherwise}
\end{cases}$$
How long to sort 10 million numbers?

Computer A

Speed: $10^{10}$ instructions/sec
Running $O(n^2)$ sort
Requires $2n^2$ instructions

How long will it take?

Computer B

Speed: $10^{7}$ instructions/sec
Running $O(n \log n)$ sort
Requires $50n \log n$ instructions

How long will it take?
How long to sort 10 million numbers?

Computer A

Speed: $10^{10}$ instructions/sec
Running $O(n^2)$ sort
Requires $2n^2$ instructions

\[
\frac{2 \times (10^7)^2}{10^{10}} \approx 20,000s
\]

~5.5 hours

Computer B

Speed: $10^7$ instructions/sec
Running $O(n \log n)$ sort
Requires $50n \log n$ instructions

How long will it take?
How long to sort 10 million numbers?

Computer A

Speed: $10^{10}$ instructions/sec  
Running $O(n^2)$ sort  
Requires $2n^2$ instructions  

\[
\frac{2 \times (10^7)^2}{10^{10}} \approx 20,000 \text{s}
\]

\~5.5 hours

Computer B

Speed: $10^7$ instructions/sec  
Running $O(n \log n)$ sort  
Requires $50 \ n \ \log n$ instructions  

\[
\frac{50 \times 10^7 \times \log 10^7}{10^7} \approx 1163 \text{s}
\]

under 20 minutes!
How long to sort 10 million numbers?

Computer A

Speed: $10^{10}$ instructions/sec
Running $O(n^2)$ sort
Requires $2n^2$ instructions

$$\frac{2 \times (10^7)^2}{10^{10}} \approx 20,000s$$

If running $50 \ n \ log \ n$ program: < 2s!!

Computer B

Speed: $10^7$ instructions/sec
Running $O(n \ log \ n)$ sort
Requires $50 \ n \ log \ n$ instructions

$$\frac{50 \times 10^7 \times log10^7}{10^7} \approx 1163s$$

under 20 minutes!
P = NP?
vertex cover of a graph is a set of vertices that includes at least one endpoint of every edge.
Vertex Cover Algorithm

• Find the minimum vertex cover of a graph
• We will discuss graph representations, just make something up for now
NP-Complete

• NP = Non-deterministic Polynomial
• in NP == Solution is verifiable in P time
• problem is provably equivalent to other NP complete problems
xkcd??

- More on xkcd.com
def algorithmDevelopment(problemSpec):
    correct = false
    while not correct or not fastEnough(runningTime):
        algorithm = deviseAlgorithm(problemSpec)
        correct = analyzeCorrectness(algorithm)
        runningTime = analyzeEfficiency(algorithm)
    return algorithm
Algorithm for Program Development

def programDevelopment(algorithm, testSuite):
    language = pickLanguage(algorithm)
    program = code(algorithm, program)
    do:
        check = false
        while not check:
            program = debug(program)
            check = verifyProgram(program, testSuite)
        performance = measure(performance)
        while not acceptable(performance)
GOOD ALGORITHMS ARE YUGE!
An algorithm to consider

• Given two lists of integers
• call these A and B
• Find: $\min(\text{abs}(A[i]-B[j]))$