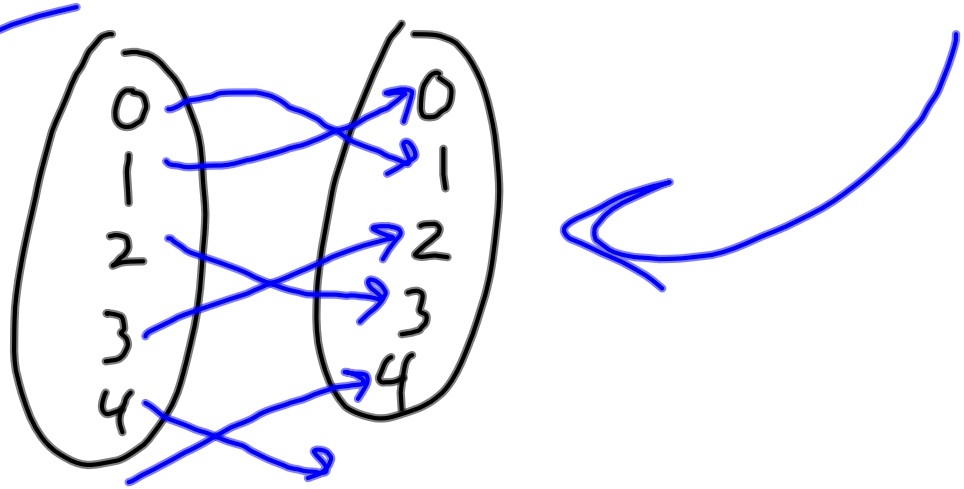
Today:

- converting bases
- fun w/ sums

$$f(n) = \left\lceil \frac{n}{2} \right\rceil$$



$$f(n) = \begin{cases} n+1 & \text{if } n \text{ even} \\ n-1 & \text{if } n \text{ odd} \end{cases}$$

Converting between bases:

stoi: string-to-int

Number	int
Numeral	String
Digit	char

stoi("5473") = 5473

↑ Integer.parseInt()

= 5000 + 400 + 70 + 3

= $5 \cdot 10^3 + 4 \cdot 10^2 + 7 \cdot 10^1 + 3 \cdot 10^0$

dtoi: digit to int

```
int stoi(String s) {
  int ssf = 0;
  for(int i=0; i<s.length(); ++i) {
    ssf += dtoi(s.charAt(s.length()-i-1))
           * Math.pow(10, i)
  }
  return ssf;
}
```

$stoi_2("1101") = 1\cdot8 + 1\cdot4 + 0\cdot2 + 1\cdot1 = 13$

$[1101]_2 = 13$

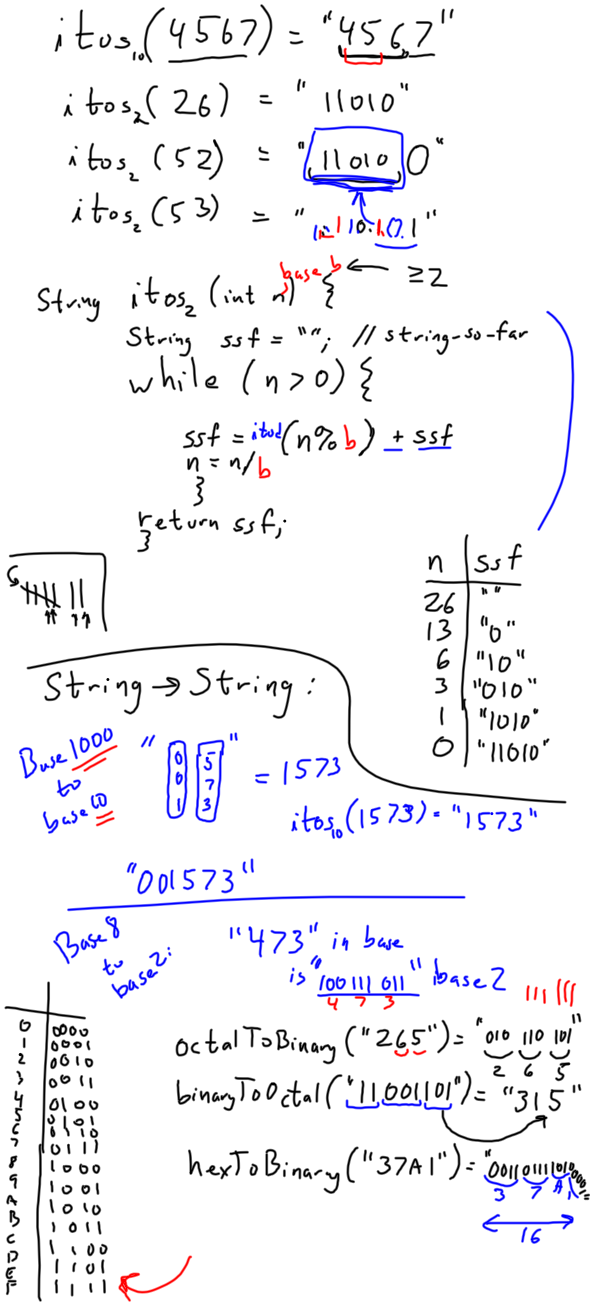
$stoi_2("11010") = 1\cdot16 + 1\cdot8 + 0\cdot4 + 1\cdot2 + 1\cdot0 = 26$

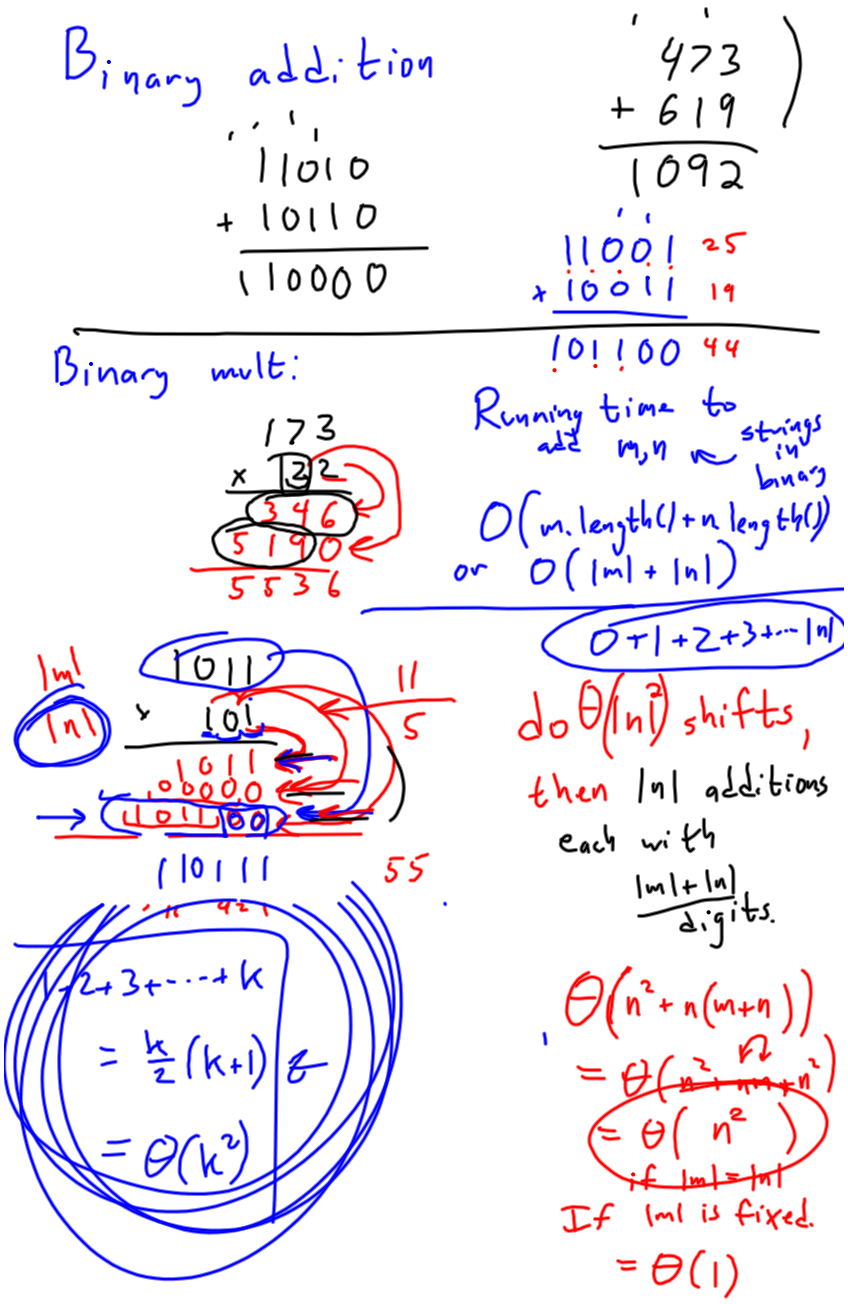
$stoi_{16}("2A1") = 2\cdot256 + 10\cdot16 + 1\cdot1 = 544$

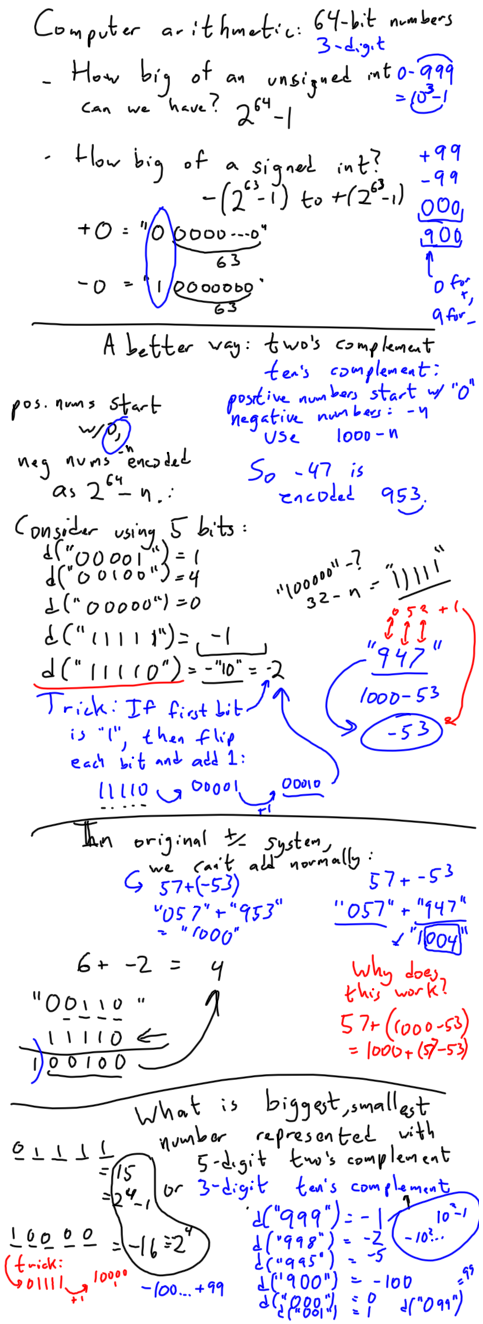
$[II]_{Roman} = 2$

$stoi_2("110100") = 52$

"There are 10 types of people — those who know binary, and those who don't"


itos10(4567) = "4567"
itos2(26) = "11010"
itos2(52) = "11010" 0
itos2(53) = "11010" 1
String itos2 (int n) { ← base b ≥2
String ssf = ""; // string-so-far
while (n > 0) {
ssf = itod(n%b) + ssf
n = n/b
}
return ssf;
}
n | ssf
26 | ""
13 | "0"
6 | "10"
3 | "010"
1 | "1010"
0 | "11010"
String → String :
Base 1000 to base 10
"001573" = 1573
itos10(1573) = "1573"
"001573"
Base 8 to base 2:
"473" in base is "100 111 011" base2
octalToBinary("265") = "010 110 101"
binaryToOctal("11,001,101") = "315"
hexToBinary("37A1") = "0011 0111 1010 0001"
16


Binary addition
11010
+ 10110
110000
473
+ 619
1092
11001 25
+ 10011 19
101100 44
Binary mult:
173
x 32
346
5190
5536
Running time to add m,n ~ strings in binary
O(m.length() + n.length())
or O(|m| + |n|)
0+1+2+3+...|n|
|m|
|n|
1011
x 101
1011
00000
1011100
110111
11
5
55
do Θ(|n|²) shifts,
then |n| additions
each with
|m|+|n| digits.
1+2+3+...+k
= k/2 (k+1)
= Θ(k²)
Θ(n² + n(m+n))
= Θ(n² + mn + n²)
= Θ(n²)
if |m|=|n|
If |m| is fixed.
= Θ(1)


Computer arithmetic: 64-bit numbers
3-digit
- How big of an unsigned int can we have? $2^{64}-1$
0-999 $=10^3-1$
- How big of a signed int? $-(2^{63}-1)$ to $+(2^{63}-1)$
+0 = "0 0000...0" 63
-0 = "1 0000000" 63
+99
-99
000
900
0 for +, 9 for -
A better way: two's complement
ten's complement:
positive numbers start w/ "0"
negative numbers: -n
Use 1000-n
pos. nums start w/ 0
neg nums encoded as $2^{64}-n$.:
So -47 is encoded 953.
Consider using 5 bits:
d("00001") = 1
d("00100") = 4
d("00000") = 0
d("11111") = -1
d("11110") = -"10" = -2
Trick: If first bit is "1", then flip each bit and add 1:
11110 00001 00010
"100000" - ? = "11111"
32 - n = "11111"
"947"
1000-53
-53
Then original ± system, we can't add normally:
57+(-53)
"057"+"953"
= "1000"
57+-53
"057"+"947"
"1004"
Why does this work?
57+(1000-53)
= 1000+(57-53)
6+-2 = 4
"00110"
11110
100100
What is biggest, smallest number represented with 5-digit two's complement or 3-digit ten's complement
01111 =15 $=2^4-1$
10000 =-16 $=2^4$
(trick: 01111 +1 10000)
-100...+99
d("999") = -1
d("998") = -2
d("995") = -5
d("900") = -100
d("000") = 0
d("001") = 1
10^2-1
-10^2...
d("099")