Unless stated otherwise, variables are integers.

1. One way to add k bits is to increase the sum by 1 each time a bit is 1. How long does this method take? Hint: what is the runtime of the following algorithm?

```
def bincount(k):  # k >= 0
  for j in range(k+1):
    print bin(j)

output from bincount(9):
0b0                             this algorithm prints numbers
0b1                             from 0 to k in binary.
0b10
0b11                            to add k bits, in the worst case
0b100                           you would have written (in memory) the
0b101                           value of the sum each time it changed,
0b110                           so you would have written the numbers
0b111                           0, 1, ..., k   (in binary)
0b1000
0b1001
```

So (unless you are doing something really clever), adding k bits takes $O(f(t))$ time, where $f(t) = \sum_{j=0}^{t} \lg(j)$ time. It is easy to see that $f(t)$ is in $\Theta(t \lg t)$. So adding k numbers takes $\Theta(k \lg k)$ time.

2. Consider binary multiplication using the school algorithm. Assume each of the two inputs has k bits.

(i) What is the runtime if the rows are added one at a time? i.e. add the first two rows, then add that sum to the next row, etc.

$k - 1$ additions, each addition involves 2 numbers, each with at most $2k - 1$ bits, so takes $\Theta(k)$ time, so total of $\Theta(k^2)$ time.

(ii) What is the runtime if the rows are added column by column?

$\Theta(k)$ columns, each with at most k bits (plus the carry bits). By previous question, adding each column takes $\Theta(k \lg k)$ time, so total $\Theta(k^2 \lg k)$ time. So (i) is more efficient.

3.
```
def d2b(n): #n >= 0
  if (n==0):
      return '0'
  if (n==1):
    return '1'
  return d2b(n/2)+d2b(n%2)
```

(i) Trace with $n = 37$.

Recursion tree below. Also: you can implement in python, add print statements, and run.

```
37 (returns 100101)
18 (returns 10010)     1 (returns 1)
9  (returns 1001)      0 (returns 0)
4  (returns 100)       1 (returns 1)
2  (returns 10)        0 (returns 0)
1  (returns 1)         0 (returns 0)
```

(ii) What does the algorithm do?

Decimal to binary.

(iii) Give runtime, as a function of n.

Let $k = \lg n$, i.e. the number of bits in n.

Recursion tree has depth $\Theta(k)$. Dividing n by 2 can be done in $\Theta(k)$ time. (why?) Computing $n \bmod 2$ can be done in $O(k)$ time. So total time takes $\Theta(k^2) = \Theta((\lg n)^2)$. Note: if we use ordinary division, which takes $\Theta(k^2)$ time, then the total is $\Theta(k^3) = \Theta((\lg n)^3)$).

(iv) Prove correctness.

Argue by induction on n. Base cases: when $n \leq 1$.

4. Trace Al Khwarizmi's multiplication with inputs 907 658.

Left as an exercise. Notice: if you multiply 658 907, you add one less number.