Problem 1. (40 points)

Let n be an integer. Determine whether n is a perfect square or not by giving an algorithm whose worst case running time is $O(\lg n)$. Integer n is a perfect square if there exists an integer x such that $n = x^2$.

Generalize this algorithm to determine whether n is a perfect power in time $O(\lg^2 n)$. Integer n is a perfect power if there exist integers x, y such that $n = x^y$.

Hint. $n = x^y$ means that $\lg n = y \lg x$. How big are $y, \lg x$? Think of an elementary method introduced in CS 114.

Problem 2. (40 points)

Suppose that we insert n keys into a hash table of size m using open addressing and uniform hashing. Let $p(n,m)$ be the probability that no collisions occur. Show that $p(n,m) \leq exp(-n(n-1)/(2m))$. Argue that when n exceed $\sqrt{m}$, the probability of avoiding collisions goes rapidly to zero.

Hint: Use induction... Also use $exp(x) \geq 1 + x$ for any real x. Note that $exp(x) = e^x$.

Problem 3. (40 points)

The Fibonacci sequence is given by the following recurrence $F_{n+1} = F_n + F_{n-1}$ for $n \geq 1$ and $F_0 = F_1 = 1$.

(a) Show how to compute F_n in $O(n)$ time.

(b) Given an $n \times n$ matrix A show how you can find A^n in $O(n^3 \lg n)$ time.

(c) Can you improve the obvious time bound in (a)? In particular prove that F_n can be computed in $O(\lg n)$ time. Hint: You may need to use the result of part (b), i.e. formulate the F_n as a matrix problem. The discussion on page 902 and 903 (Problem section at the end of the Chapter on Number-Theoretic Algorithms may offer you some insight).

The Fibonacci sequence is given by the following recurrence $F_n = F_{n-1} + F_{n-2}$ for $n \geq 1$ and $F_0 = 0, F_1 = 1$. It is easy to compute F_n in $O(n)$ time with an iterative algorithm. Show how one can compute F_n in $O(\lg n)$ time. Pages 901/902 may offer some assistance but note that the problem there is in some other context.

Problem 4. (40 points)

Consider two sets A and B each having n integers in the range from 0 to $10n$. We wish to compute the Cartesian sum of A and B defined by

$$C = \{x + y : x \in A \ \ and \ \ y \in B\}$$

Note that the integers in C are in the range from 0 to 20n. We want to find the elements of C and the number of times each element of C is realized as a sum of elements in A and B. Show that if the product of two degree bound n polynomials can be computed in $O(n \lg n)$ time, then this problem can also be solved in $O(n \lg n)$ time. **Hint.** Represent A and B as polynomials of degree at most $10n$.

Problem 5. (40 points)

You are given six polynomials $f_1, \ldots f_6$ of degrees $1, 2, 3, 1, 4, 5$ respectively. We are interested in finding the product $f = f_1 f_2 f_3 f_4 f_5 f_6$. Assume that the cost of multiplying two polynomials of degree a and b is $a \cdot b$. Find a schedule for multiplying the six polynomials that is of the lowest possible total cost.

Problem 6. (50 points)

You are given six polynomials $f_1, \ldots f_6$ of degrees $1, 2, 3, 1, 4, 5$ respectively. We are interested in finding the product $f = f_1 f_2 f_3 f_4 f_5 f_6$. Assume that the cost of multiplying two polynomials of degree a and b is $a + b$ (note the difference from the previous problem) i.e. it is proportional to the space required to store the product which is a polynomial of degree $a + b$.

Find a schedule for multiplying the six polynomials that is of the lowest possible total cost for this non-traiditional definition of a cost function.

Example. If you have three polynomials g_1, g_2, g_3 of degrees $1, 2, 3$ respectively and you first compute $g_2 g_3$ and then the multiply the result by g_1, the cost of the first multiplication is 5 ($= 2 + 3$) and the cost of the second multiplication is 6 since you multiply the result, a degree 5 polynomial to a degree one polynomial. Total cost is $5 + 6 = 11$. Is this the best you can do for these three polynomials?

Problem P1. (100 points)

Implement hashing by chaining and hashing by open-addressing. Implement (approximate interface) the following functions.

```
int HashFunction(key k)
or
int HashFunction(key k, probe i) // Hash function takes key k as input returns 0..m-1
/* For open addressing (Oa) implement
 * h(k,i) as  (h(k) % m + i**2 + i ) % m
 */


HashChainCreate(table T, int m); // Create a hash table/Initialize
HashChainEmpty (table T, int m); //Check if Table is empty
HashChainFull  (table T, int m); // or full
HashChainInsert(table T, key k, int m);
HashChainDelete(table T, key k, int m);
HashChainSearch(table T, key k, int m);
```

```
and
   HashOaCreate(table T, int m); // Create a hash table/Initialize
   HashOaEmpty (table T, int m); //Check if Table is empty
   HashOaFull  (table T, int m); // or full; this is different from overflow.
   HashOaInsert(table T, key k, int m);
   HashOaDelete(table T, key k, int m);
   HashOaSearch(table T, key k, int m);

   HashTable(type  t, table T, operation o, key k);
   ProcessHash(file file-name)
```

The end result is the implementation of HashTable, a function that can implement both types hash tables (eg. if t is equal to 0 then it means chaining, and 1 open-addressing with quadratic probing as defined above). An operation o can be defined in a single line with two arguments. The first being the operation (10 for Insertion, 11 for Deletion, and 12 for search) and the second the key value involved (assume integers keys.

I will test your code, through the command line, by typing in `Your program should support such an interface.`

```
 1      <<<<mean open-addressing
10 1
10 10
10 20
10 8
10 7
12 10
11 20
11 8
10 30
12 25
```

`12 10 returns the index of the hash table containing key 10, but 12 25 returns -1 (key not found) .`

Problem P2. (100 points)

Implement Shamir's secrete sharing scheme.

```
ShamirCreate(secret k, parties p, reconstruct  r, file-out file-name)
  // Returns  a file-name that contains one per-line the individual secrets
  // assigned to each of the p parties. file-name thus has p lines one for each party

secret ShamirReconstruct(parties p, reconstruct  r, file-in  file-name)
  // Uses file-name with at least r lines but no more than p to reconstruct
  // the secret s that is returned.

  // Details are left to you for implementation.
```

The interface will be through the command-line. A

```
% ./ShamirCreate k p r out-file
```

will call the corresponding function and generate some output in file out-file. (Note that `ShamirCreate` is not only a function name but also a program name.)

A `ShamirReconstruct p r my-file` will use my-file (containing lines of out-file) to return in the standard output the secret.

The catch of this Problem: Numbers can grow very big! The secret is a positive 32-bit integer `int` or `unsigned int`.