```
---------------------------------------------------------------
fib.cpp: different ways to compute the n'th fibonacci number
---------------------------------------------------------------

#include ...
using namespace std;

class fibonacci {
  public:
    long fib(int,int);  // public interface

  private:
    int fib1(int);      // straight recursion
    int fib2(int);      // recursion with cache
    int fib3(int);      // iteration with cache
    int fib4(int);      // iteration with state

    vector<int> cache;
};

long fibonacci::fib(int mode, int n) {
  if (mode == 2 || mode == 3)
    cache.assign(n+1, -1);

  long N = 0;
  switch (mode) {
    case 1: N = fib1(n); break;
    case 2: N = fib2(n); break;
    case 3: N = fib3(n); break;
    case 4: N = fib4(n); break;
  }

  return N;
}

int fibonacci::fib1(int n) {
  if (n <= 1) return n;

  return fib1(n-1) + fib1(n-2);
}

int fibonacci::fib2(int n) {
  if (cache[n] != -1) return cache[n];

  if (n <= 1) cache[n] = n;
  else        cache[n] = fib2(n-1) + fib2(n-2);

  return cache[n];
}

int fibonacci::fib3(int n) {
  if (n <= 1) return n;

  cache[0] = 0;
  cache[1] = 1;
  for (int i=2; i<=n; i++)
    cache[i] = cache[i-1] + cache[i-2];

  return cache[n];
}

int fibonacci::fib4(int n) {
  if (n <= 1) return n;

  int v[3] = { 0, 1, 1 };

  for (int i=3; i<=n; i++) {
    v[0] = v[1];
    v[1] = v[2];
    v[2] = v[0] + v[1];
  }

  return v[2];
}

int main(int argc, char *argv[]) {
  if (argc != 2) {
    printf("usage: %s n\n", argv[0]);
    return 0;
  }

  fibonacci F;
  int n = max(0, atoi(argv[1]));

  for (int i=1; i<=4; i++) {
    float T0 = omega(); long Fn = F.fib(i, n); float T1 = omega();
    printf("Fib%d %10ld %9.3f ms\n", i, Fn, 1000.0*(T1-T0));
  }
}

---------------------------------------------------------------
Hint: Don't try a large value for Fib1 unless you are not busy.
Hint: Fib4 can be translated into a recursion, say Fib5. How?
---------------------------------------------------------------
```

```
------------------------------------------------------------------
howmany.cpp: three ways to find minimum number different valued
             integers (aka "coins") needed to match a given sum
------------------------------------------------------------------

#include <...>
using namespace std;

class howmany {
  public:
    void set_mode(int n_mode) { mode = n_mode; }
    void push(int value) { v.push_back(value); }

    void solve(int);

    int get_N() { return N; }

  private:
    int solve1(int);   // straight recursion
    int solve2(int);   // recursion with cache
    int solve3(int);   // iteration with cache

    int mode;          // which solver to use
    int sum;           // sum to be produced
    int N;             // number solution coins

    vector<int> v;
    vector<int> cache;
};

int howmany::solve1(int s) {
  if (s == 0) return 0;

  int min = INT_MAX;

  for (int i=0; i<(int)v.size(); i++) {
    if (s >= v[i]) {
      int j = 1 + solve1(s-v[i]);
      if (0 < j && j < min)
        min = j;
    }
  }

  if (min == INT_MAX) min = -1;

  return min;
}

int howmany::solve2(int s) {
  if (cache[s] != -2) return cache[s];

  if (s == 0) { cache[s] = 0; return 0; }

  cache[s] = INT_MAX;

  for (int i=0; i<(int)v.size(); i++) {
    if (s >= v[i]) {
      int j = 1 + solve2(s-v[i]);
      if (0 < j && j < cache[s])
        cache[s] = j;
    }
  }

  if (cache[s] == INT_MAX) cache[s] = -1;

  return cache[s];
}

int howmany::solve3(int s) {
  cache[0] = 0;

  for (int k=1; k<=s; k++) {

    cache[k] = INT_MAX;

    for (int i=0; i<(int)v.size(); i++) {
      if (k >= v[i]) {
        int j = 1 + cache[k-v[i]];
        if (0 < j && j < cache[k])
          cache[k] = j;
      }
    }

    if (cache[k] == INT_MAX) cache[k] = -1;
  }

  return cache[s];
}
```

```
void howmany::solve(int n_sum) {
  sum = n_sum;
  N = 0;

  if (mode == 1) {
    N = solve1(sum);
  } else
  if (mode == 2) {
    cache.resize(sum+1, -2);
    N = solve2(sum);
  } else
  if (mode == 3) {
    cache.resize(sum+1, -1);
    N = solve3(sum);
  }
}

int main(int argc, char *argv[]) {
  if (argc < 4) {
    cerr << "usage: " << argv[0]
         << " -1|2|3 v1 v2 [.... vn]\n";
    return 0;
  }

  howmany H;

  H.set_mode(atoi(&argv[1][2]));

  for (int i=2; i<argc; i++)
    H.push(atoi(argv[i]));

  while (1) {
    int sum = 0;
    cout << "sum> ";
    cin >> sum;

    if (cin.eof())
      break;

    H.solve(sum);

    cout << H.get_N() << " coins needed\n";
  }

  cout << "\n";
}
```

Hint: howmany.cpp code can be modified to store link information that allows show() function to print how solution was produced

---

```
class howmany {
  ...
  public:
    void show(int);

  private:
    vector<int> link;
};

int howmany::solve3(int s) {
  cache[0] = 0;
  for (int k=1; k<=s; k++) {

    cache[k] = INT_MAX;

    for (int i=0; i<(int)v.size(); i++) {
      if (k >= v[i]) {
        int j = cache[k-v[i]] + 1;
        if (j != 0 && j < cache[k]) {
          cache[k] = j;
          link[k] = k-v[i];
        }
      }
    }

    if (cache[k] == INT_MAX) cache[k] = -1;
  }

  return cache[s];
}

void howmany::solve(int n_sum) {
  ...
  link.resize(sum+1, -1);
}

void howmany::show() {
  for (int k=sum; link[k] != -1; k=link[k]) {
    int value = k-link[k];
    cout << " " << value;
  }
  cout << "\n";
}
```