On the right are two programs that both do the same thing. If you care, **rand()** returns a pseudorandom number between 0 and 2^{31}. When I time **p1** on one computer, the following takes three seconds:

```
UNIX> echo 10000 100000 | ./p1
```

- **Question 1**: Roughly how many seconds will the following take?

  ```
  UNIX> echo 10000 400000 | ./p1
  ```

- **Question 2**: Roughly how many seconds will the following take?

  ```
  UNIX> echo 30000 700000 | ./p1
  ```

I move to a different computer, and the following takes three seconds:

```
UNIX> echo 10000 100000 | ./p2
```

- **Question 3**: Roughly how many seconds will the following take?

  ```
  UNIX> echo 10000 400000 | ./p2
  ```

- **Question 4**: Roughly how many seconds will the following take?

  ```
  UNIX> echo 30000 700000 | ./p2
  ```

p1.cpp, compiled to **p1**.

```
#include <vector>
#include <iostream>
using namespace std;

int main()
{
  int vs, ns;
  vector <int> v;
  int matches, i, j, n;

  cin >> vs >> ns;

  for (i = 0; i < vs; i++) {
    v.push_back(rand()%100000);
  }

  matches = 0;
  for (i = 0; i < ns; i++) {
    n = rand()%100000;
    for (j = 0; j < v.size(); j++) {
      if (v[j] == n) matches++;
    }
  }
  cout << matches << endl;
  return 0;
}
```

p2.cpp, compiled to **p2**.

```
#include <vector>
#include <iostream>
using namespace std;

int main(int argc, char **argv)
{
  int vs, ns;
  vector < vector <int> > v;
  int matches, i, j, n;

  cin >> vs >> ns;
  v.resize(vs);

  for (i = 0; i < vs; i++) {
    n = rand()%100000;
    v[n%vs].push_back(n);
  }

  matches = 0;
  for (i = 0; i < ns; i++) {
    n = rand()%100000;
    for (j = 0; j < v[n%vs].size(); j++) {
      if (v[n%vs][j] == n) matches++;
    }
  }
  cout << matches << endl;
  return 0;
}
```

Question 1

The second **for** loop now runs 400,000 iterations instead of 100,000. It is doing the same amount of work in each iteration (looping through 10,000) numbers. So this will take four times as long. The answer is 12.

Question 2

The second **for** loop now runs 700,000 iterations instead of 100,000. It is traversing a vector that is three times the size as before, so each iteration will take three times as long. The whole program will thus take 21 times as long. The answer is 63.

We don't need to factor in the time for the first loop -- it is going to be *so* much faster than the second loop that we don't need to care about it.

Question 3

Once again, the second **for** loop now runs 400,000 iterations instead of 100,000. It is doing the same amount of work in each iteration (looking for numbers in a hash table that uses separate chaining). As with Question 2, it will take four times as long. The answer is 12. You'll note that the computer running this program must be a lot slower than the computer in Questions 1 and 2.

Question 4

We triple the size of our hash table, but its load factor is still 1. So the inner loop of the second **for** loop is doing the same amount of work as in Question 3. Thus, the slowdown is only 7. The answer is 21. The difference between the two loops will be less than in Questions 1 and 2, but the first loop is fast enough to be considered in the noise, compared to the second loop.