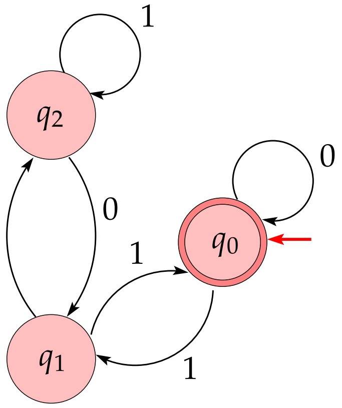
Proposta de solució al problema 1

(a) Una possible solució:



(b) Les respostes:

	(1)	(2)	(3)	(4)	(5)
CERT	X	X	X	X	
FALS					X

Proposta de solució al problema 2

(a) Escollim un índex i que apunti al mínim del vector.

En general, si i és l'índex tal que reemplacem $a[i]$ per $-a[i]$, definim S_i com la suma dels elements del vector després del reemplaçament. Tenim que $S_i = (-a[i] + \sum_{k=0}^{n-1} a[k]) + (-a[i]) = -2a[i] + \sum_{k=0}^{n-1} a[k]$. Per tant, donats dos índexos i, j, tenim que

$$S_i \geq S_j \Leftrightarrow -2a[i] + \sum_{k=0}^{n-1} a[k] \geq -2a[j] + \sum_{k=0}^{n-1} a[k] \Leftrightarrow -2a[i] \geq -2a[j] \Leftrightarrow a[i] \leq a[j]$$

Per tant, si i apunta al mínim del vector, per tot j es compleix $S_i \geq S_j$.

(b) Una possible solució:

```
int max_sum(const vector<int>& a, int k) {
  priority_queue<int, vector<int>, greater<int>> pq;
  for (int ai : a)
    pq.push(ai);
  for (int j = 0; j < k; ++j) {
    int ai = pq.top();
    pq.pop();
    pq.push(-ai);
  }
```

```
    int sum = 0;
    while (not pq.empty()) {
      sum += pq.top();
      pq.pop();
    }
    return sum;
  }
```

Aquesta funció té cost $\Theta((n+k)\log n)$. Tot i que ja es dóna per bona, encara es podria fer més eficient creant el heap en temps lineal en lloc de fer

```
    priority_queue<int, vector<int>, greater<int>> pq;
    for (int ai : a)
      pq.push(ai);
```

Llavors tindria cost $\Theta(n + k \log n)$.

Proposta de solució al problema 3

(a) Una possible solució:

```
  int n, W;
  vector<int> v, w;

  int opt1() {
    vector<vector<int>> m(n+1, vector<int>(W+1, 0));
    for (int i = 1; i <= n; ++i) {
      for (int j = 0; j < w[i-1]; ++j)
        m[i][j] = m[i-1][j];
      for (int j = w[i-1]; j <= W; ++j)
        m[i][j] = max(v[i-1] + m[i-1][j-w[i-1]], m[i-1][j]);
    }
    return m[n][W];
  }

  int main() {
    cin >> n >> W;
    v = w = vector<int>(n);
    for (int& x : v) cin >> x;
    for (int& x : w) cin >> x;
    cout << opt1() << endl;
  }
```

(b) El cost de construir la matriu és $\Theta(n \cdot W)$. Per altra banda, el bucle extern fa n voltes, cadascuna de les quals té cost $\Theta(W)$. Per tant, el cost de *opt1* és $\Theta(n \cdot W + n \cdot W) = \Theta(n \cdot W)$.

(c) La recurrència és:

$$c(i,V) = \begin{cases} 0 & \text{si } V \leq 0 \\ w_{i-1} + c(i-1, V - v_{i-1}) & \text{si } V > \sum_{k=0}^{i-2} v_k \\ \min(w_{i-1} + c(i-1, V - v_{i-1}), c(i-1, V)) & \text{altrament} \end{cases}$$

(d) Una possible solució:

```
int n, W;
vector<int> v, w, s;
vector<vector<int>> cc;

int c(int i, int V) {
  if (V ≤ 0) return 0;
  int& res = cc[i][V];
  if (res ≠ −1) return res;
  if (V > s[i−1]) return res =      w[i−1] + c(i−1, V − v[i−1]);
  else            return res = min(w[i−1] + c(i−1, V − v[i−1]), c(i−1, V));
}

int opt2() {
  s = vector<int>(n+1, 0);
  for (int k = 1; k ≤ n; ++k) s[k] = s[k−1] + v[k−1];
  int S = s.back();
  cc = vector<vector<int>>(n+1, vector<int>(S+1, −1));
  int V = 0;
  while (V ≤ S and c(n, V) ≤ W) ++V;
  return V−1;
}

int main() {
  cin >> n >> W;
  v = w = vector<int>(n);
  for (int& x : v) cin >> x;
  for (int& x : w) cin >> x;
  cout << opt2() << endl;
}
```

(e) El cost de construir i omplir el vector s és $\Theta(n)$. Per altra banda, el cost de construir la matriu cc és $\Theta(n \cdot S)$. El bucle final, sense comptar el cost de les crides $c(n, V)$, fa com a molt S voltes, cadascuna de cost constant. Per tant el cost és $O(S)$. Per últim, el cost acumulat de totes les crides a $c(n, V)$ és $O(n \cdot S)$, ja que hi ha $O(n \cdot S)$ subproblemes, i calcular-ne un només requereix cost constant. En total doncs el cost és $\Theta(n) + \Theta(n \cdot S) + O(S) + O(n \cdot S) = \Theta(n \cdot S)$.

(f) En els apartats anteriors hem vist que *opt1* té cost $\Theta(n \cdot W)$ (independentment de S) i que *opt2* té cost $\Theta(n \cdot S)$ (independentment de W). Per tant:

- Si W és petit i S és gran, escollirem el programa amb *opt1*.
- Si W és gran i S és petit, escollirem el programa amb *opt2*.

Proposta de solució al problema 4

Una possible solució:

```
int n, m, p;
vector<int> a, b, u;
vector<vector<int>> s;

bool can_place_at (int k, int i, int j) {
  if (i + a[k] > n or j + b[k] > m) return false;
  for (int r = i; r < i + a[k]; ++r)
    for (int c = j; c < j + b[k]; ++c)
      if (s[r][c] ≠ −1) return false;
  return true;
}

void fill (int k, int i, int j, int v) {
  for (int r = i; r < i + a[k]; ++r)
    for (int c = j; c < j + b[k]; ++c)
      s[r][c] = v;
}

bool bt(int i, int j, int left) {
  if (n*m − (i*m + j) < left) return false;
  if (i == n) {
    for (int r = 0; r < n; ++r) {
      for (int c = 0; c < m; ++c)
        cout << ' ' << s[r][c];
      cout << endl;
    }
    return true;
  }
  if (j == m) return bt(i+1, 0, left);
  if (s[i][j] ≠ −1) return bt(i, j+1, left);
  for (int k = 0; k < p; ++k) {
    if (not u[k] and can_place_at (k, i, j)) {
      u[k] = true;
      fill (k, i, j, k);
      if (bt(i, j+1, left − a[k]*b[k])) return true;
      fill (k, i, j, −1);
      u[k] = false;
    }
  }
  return bt(i, j+1, left);
}

int main() {
  cin >> n >> m >> p;
```

```
  a = b = vector<int>(p);
  int left = 0;
  for (int k = 0; k < p; ++k) {
    cin >> a[k] >> b[k];
    left += a[k] * b[k];
  }
  s = vector<vector<int>>(n, vector<int>(m, -1));
  u = vector<int>(p, false);
  if (not bt(0, 0, left)) cout << "No solution" << endl;
}
```