Proposed solution to problem 1

(a) $\Theta(n^{\log 7})$

(b) The master theorem for subtractive recurrences states that given a recurrence $T(n) = aT(n-c) + \Theta(n^k)$ with $a, c > 0$ and $k \geq 0$ then

$$T(n) = \begin{cases} \Theta(n^k) & \text{if } a < 1, \\ \Theta(n^{k+1}) & \text{if } a = 1, \\ \Theta(a^{\frac{n}{c}}) & \text{if } a > 1. \end{cases}$$

(c) $\Theta(n)$

Proposed solution to problem 2

(a) The cost of the program is determined by the sum of the costs of two groups of instructions:

- **A.** The instructions that are executed at each iteration of the loop: the evaluation of the condition $i \leq n$, the call $f(i)$, the evaluation of the condition $i == p$ and the increment i++.
- **B.** The instructions that are executed when the condition of the **if** is true: the call $g(n)$ and the increment $p *= 2$.

We count separately the contribution to the total cost of the two groups of instructions:

- **A.** If the cost of $f(m)$ is $\Theta(m)$, then at the i-th iteration the cost of **A** is $\Theta(i)$. So in total the contribution to the cost is $\sum_{i=1}^{n} \Theta(i) = \Theta(\sum_{i=1}^{n} i) = \Theta(n^2)$.
- **B.** If the cost of $g(m)$ is $\Theta(m)$, then every time that the condition of the **if** is true the cost of **B** is $\Theta(n)$. Since this happens $\Theta(\log n)$ times, the contribution to the total cost is $\Theta(n \log n)$.

Hence in total the cost of $h(n)$ is $\Theta(n^2) + \Theta(n \log n) = \Theta(n^2)$.

(b) Let us consider the same two groups of instructions of the previous exercise, and again we count separately their contribution to the total cost:

- **A.** If the cost of $f(m)$ is $\Theta(1)$, then the cost of **A** at each iteration is $\Theta(1)$. Since $\Theta(n)$ are performed, the cost in total is $\Theta(n)$.
- **B.** If the cost of $g(m)$ is $\Theta(m^2)$, then every time that the condition of the **if** is true the cost of **B** is $\Theta(n^2)$. Since this happens $\Theta(\log n)$ times, the cost is $\Theta(n^2 \log n)$.

Hence in total the cost of $h(n)$ is $\Theta(n) + \Theta(n^2 \log n) = \Theta(n^2 \log n)$.

Proposed solution to problem 3

(a) The i-th row uses $i+1$ cells ($0 \leq i < n$). In total, the consumed space is

$$\sum_{i=0}^{n-1}(i+1) = \sum_{j=1}^{n} j = \Theta(n^2).$$

(b) A possible solution:

```
int p(int x) { return x*(x+1)/2;}

int mystery(int k, int l, int r) {
  if (l+1 == r) return l;
  int m = (l+r)/2;
  if (p(m) ≤ k) return mystery(k, m, r);
  else          return mystery(k, l, m);
}

pair<int,int> row_column(int n, int k) {
  if (k < 0 or k ≥ p(n)) return {−1,−1};
  int i = mystery(k, 0, n);
  return {i, k − p(i)};
}
```

(c) Let $C(N)$ be the cost in the worst case of a call to the function *mystery*(k, l, r), where $N = r - l + 1$. The recurrence that describes $C(N)$ is

$$C(N) = C(N/2) + \Theta(1),$$

since one makes one recursive call over an interval of half the size, and operations that take constant time. By using the master theorem of divisive recurrences, the solution to the recurrence is $C(N) = \Theta(\log N)$.

Therefore, the cost in the worst case of a call to function *row_column*(n, k) is $\Theta(\log n)$.

(d) The index i of the searched row is the natural number $0 \leq i < n$ such that $p(i) \leq k < p(i+1)$. We can compute it by solving the second degree equation $p(x) = k$ and taking $i = \lfloor x \rfloor$:

```
int p(int x) { return x*(x+1)/2;}

pair<int,int> row_column(int n, int k) {
  if (k < 0 or k ≥ p(n)) return {−1,−1};
  int i(floor ((sqrt (1.+8*k) − 1)/2));
  return {i, k − p(i)};
}
```

Proposed solution to problem 4

(a) Let us show it by contradiction. Let us assume that i is such that $0 \leq i < n-1$ i $f_A(i+1) < f_A(i)$. Let us take $k = i+1$, $j = f_A(i+1)$ and $l = f_A(i)$, so that $0 \leq i < k < n$ and $0 \leq j < l < n$. Since $j = f_A(k)$, we have that $A_{k,j} \leq A_{k,l}$. And as $l = f_A(i)$, we have that $A_{i,l} \leq A_{i,j}$; in fact, since $j < l$, it must be $A_{i,l} < A_{i,j}$. Thus, by adding we have that $A_{i,l} + A_{k,j} < A_{i,j} + A_{k,l}$. This contradicts that A is a Monge matrix.

(b) For each even i, one can compute the column where the leftmost minimum of the i-th row of A appears as follows. From the recursive call over B_1, we have the column j_1 where the leftmost minimum of the i-th row of A appears between columns 0 and $\frac{n}{2} - 1$. Similarly, from the recursive call over B_2, we have the column j_2 where the leftmost minimum of the i-th row of A appears between columns $\frac{n}{2}$ and $n-1$. Hence, $f_A(i) = j_1$ if $A_{i,j_1} \leq A_{i,j_2}$, and $f_A(i) = j_2$ otherwise.

For each odd i, one determines $f_A(i)$ by examining the columns between $f_A(i-1)$ and $f_A(i+1)$ (defining $f_A(n) = n-1$ for notational convenience), and choosing the index where the leftmost minimum appears. This has cost $\Theta(f_A(i+1) - f_A(i-1) + 1)$. In total:

$$\sum_{i=1, i \text{ odd}}^{n-1} \Theta(f_A(i+1) - f_A(i-1) + 1) = \Theta((n-1) - f_A(0) + \frac{n}{2}) = \Theta(n)$$

since $0 \leq f_A(0) < n$.

(c) Let $C(n)$ be the cost of the proposed algorithm for computing the function f_A for all rows of a Monge matrix A of size $n \times n$.

Apart from the two recursive calls in step (2) over matrices of size $\frac{n}{2} \times \frac{n}{2}$, steps (1) and (3) require time $\Theta(n)$ in total. So the recurrence that describes the cost is

$$C(n) = 2C(n/2) + \Theta(n).$$

By using the master theorem of divisive recurrences, the solution to the recurrence is $C(n) = \Theta(n \log n)$.