Proposed solution to problem 1

(a) Function g grows faster:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^{n^2}}{2^{2^n}} = \lim_{n\to\infty} 2^{\log(\frac{n^{n^2}}{2^{2^n}})} = 2^{\lim_{n\to\infty}\log(\frac{n^{n^2}}{2^{2^n}})} = 0$$

since

$$\lim_{n\to\infty} \log(\frac{n^{n^2}}{2^{2^n}}) = \lim_{n\to\infty} (\log(n^{n^2}) - \log(2^{2^n})) = \lim_{n\to\infty} (n^2 \log n - 2^n) = -\infty$$

(b) The cost of the algorithm in the average case is $\Theta(n^2)$:

$$\frac{1}{n^3} \cdot \Theta(n^4) + \frac{1}{n} \cdot \Theta(n^3) + (1 - \frac{1}{n^3} - \frac{1}{n}) \cdot \Theta(n) = \Theta(n) + \Theta(n^2) + \Theta(n) = \Theta(n^2)$$

(c) By using the Master Theorem of divisive recurrences we have that $a = 2$, $b = 4$, $\alpha = \log_4 2 = \frac{1}{2}$ and $k = \frac{1}{2}$. So $\alpha = k$, and therefore $T(n) = \Theta(n^\alpha \cdot \log n) = \Theta(\sqrt{n} \cdot \log n)$.

Proposed solution to problem 2

(a) The base case for $k = 0$ is true: $A^0 \cdot x_0 = x_0 = x(0)$. For the inductive case, when $k > 0$, we have by induction hypothesis that $x(k-1) = A^{k-1} \cdot x_0$. So $x(k) = A \cdot x(k-1) = A \cdot (A^{k-1} \cdot x_0) = (A \cdot A^{k-1}) \cdot x_0 = A^k \cdot x_0$.

(b) In the first place A^k is computed using the algorithm of fast exponentiation in time $\Theta(\log k)$ (since n is constant). Then the result of multiplying A^k times *x0* is returned, which can be done in time $\Theta(1)$ (again, as n is constant). The cost in time of the algorithm is then $\Theta(\log k)$.

Proposed solution to problem 3

(a) $x = x_2 \cdot 3^{2n/3} + x_1 \cdot 3^{n/3} + x_0$.

(b) $x \cdot y = x_2 y_2 \cdot 3^{4n/3} + (x_1 y_2 + x_2 y_1) \cdot 3^{3n/3} + (x_0 y_2 + x_1 y_1 + x_2 y_0) \cdot 3^{2n/3} + (x_1 y_0 + x_0 y_1) \cdot 3^{n/3} + x_0 y_0$.

(c) $x \cdot y =$

$$x_2 y_2 \cdot 3^{4n/3}$$
$$+(x_1 y_2 + x_2 y_1) \cdot 3^{3n/3}$$
$$+((x_0 + x_1 + x_2) \cdot (y_0 + y_1 + y_2) - x_2 y_2 - (x_1 y_2 + x_2 y_1) - (x_1 y_0 + x_0 y_1) - x_0 y_0) \cdot 3^{2n/3}$$
$$+(x_1 y_0 + x_0 y_1) \cdot 3^{n/3}$$
$$+x_0 y_0$$

We use 7 products.

(d) To compute the product of x and y of n digits, the algorithm computes recursively $(x_0 + x_1 + x_2) \cdot (y_0 + y_1 + y_2)$, $x_2 y_2$, $x_1 y_2 + x_2 y_1$, $x_1 y_0 + x_0 y_1$ and $x_0 y_0$, which are products of numbers of $n/3$ digits. Then $x \cdot y$ is computed using the equation of the previous section. As numbers are represented in base 3, to multiply by a power of 3 consists in adding zeroes to the right and can be done in time $\Theta(n)$. The involved additions can also be done in time $\Theta(n)$. Therefore the cost $T(n)$ satisfies the recurrence $T(n) = 7T(n/3) + \Theta(n)$, which has solution $\Theta(n^{\log_3 7})$.

Proposed solution to problem 4

(a) We define function $U(m) = T(b^m)$. Then $T(n) = U(\log_b(n))$. Moreover, we have:

$$U(m) = T(b^m) = T(b^m/b) + \Theta(\log^k(b^m)) = T(b^{m-1}) + \Theta(m^k \log^k(b)) =$$
$$= T(b^{m-1}) + \Theta(m^k) = U(m-1) + \Theta(m^k)$$

The Master Theorem of subtractive recurrences claims that if we have a recurrence of the form $U(m) = U(m-c) + \Theta(m^k)$ with $c > 0$ and $k \geq 0$, then $U(m) = \Theta(m^{k+1})$. So the solution to the recurrence of the statement is $T(n) = \Theta((\log_b(n))^{k+1}) = \Theta(\log^{k+1} n)$.

(b) A possible solution:

```
bool search(const vector<int>& a, int x, int l, int r) {
  if (l == r) return x == a[l];
  int m = (l+r)/2;
  auto beg = a.begin();
  if (a[m] < a[m+1])
    return search(a, x, m+1, r) or binary_search(beg + l,   beg + m + 1, x);
  else
    return search(a, x, l,   m) or binary_search(beg + m+1, beg + r + 1, x);
}

bool search(const vector<int>& a, int x) {
  return search(a, x, 0, a.size()-1);
}
```

(c) The worst case takes place for instance when x does not appear in a. In this situation the cost $T(n)$ is described by the recurrence $T(n) = T(n/2) + \Theta(\log n)$, as we make one recursive call over a vector of size $\frac{n}{2}$, and the cost of the non-recursive work is dominated by the binary search, which has cost $\Theta(\log(\frac{n}{2})) = \Theta(\log(n))$. By applying the first section we have that the solution is $T(n) = \Theta(\log^2(n))$.