1. [10 marks] = 2 + 5 + 2 + 1 each part

   (a) There will be an even number of white socks in the drawer at the end of the day.

   (b) Let w_i be the number of white socks in the drawer at the beginning of the day.
   Let w_f be the number of white socks in the drawer at the end of the day.
   We must prove that $even(w_i) \Rightarrow even(w_f)$.
   Assume $even(w_i)$.
   Maureen will draw two socks from the drawer. There are three possibilities:
   Either the socks are different colours $\vee$ the socks are both black *vee* the socks are both white.
   Case 1: Suppose the socks drawn are different colours.
   Then the white sock is returned to the drawer.
   So $w_f = w_i - 1 + 1$ (since we removed and added one white sock).
   Thus $w_f = w_i$.
   thus $even(w_f)$.
   Case 1: Suppose both socks drawn are black.
   Then one black sock from the pile is added to the drawer.
   So $w_f = w_i$ (because we don't remove or add a white sock).
   Thus $even(w_f)$.
   Case 2: Suppose both socks drawn are white.
   Then one black sock from the pile is added to the drawer.
   Thus $w_f = w_i - 2$ (because we removed two white socks and added none).
   Thus $even(w_f)$.
   Thus $even(w_f)$
   Hence $even(w_i) \Rightarrow even(w_f)$.
   (Formally, we prove $\forall w_i \in \mathbb{N}, \forall w_f \in \mathbb{N}$, if w_i and w_f are related by the above procedure such that w_i is the number of white socks at the start of the day and w_f is the number of socks at the end of the day, then $even(w_i) \Rightarrow even(w_f)$.)

   (c) Black. By part (b), we proved that after each day, the parity of white socks remains the same. Therefore, if the initial number of white socks in the drawer is even, after each day (including the last day) there will still be an even number of white socks. If there is only one sock left, it cannot be white, and must be black.

   (d) White. By a similar argument to (c).

2. [14 marks] = 4 + 10 each part

   (a) We want to construct an array in such a way that the algorithm will run the maximum number of steps possible.

   Assume we ran the algorithm on some worst-case input array A. We will now describe this run of the algorithm.

   Key Point 1: To run the maximum number of steps, we want the condition in line 3 (that is, $A[i] \geq b$) to be true for every iteration of the outer for-loop.

   Key Point 2: When the condition in line 3 is true, b gets reset in line 4 to $b = A[i]$. Since we want line 3's condition to always be true, we know that b will be reset in every iteration i to $b = A[i]$.

We now need to track how b varies in the outer for-loop, to determine how to design A (so that the condition in line 3 is always true).

Before the loop
$b = A[0]$

Considering each iteration of the outer loop

when $i = 0$
$A[i] = A[0] \geq A[0] = b$
So $A[i] \geq b$, and b gets reset at line 4 to $b = A[0]$.

when $1 \leq i + 1 < A.length$:
From *Key Point 2*, we know that at the beginning of iteration $i + 1$, $b = A[i]$ (since it was set to that in the previous iteration).
Since (from *Key Point 1*) we want $A[i + 1] \geq b$ to be true, then we want $A[i + 1] \geq A[i]$ to be true.
So we should make $A[i + 1] \geq A[i]$.

Therefore, a worst case input array A is one where for each index i (with $1 \leq i+1 < A.length$), $A[i + 1] \geq A[i]$. In other words, A should be sorted in non-decreasing order.

So $A = [1, 1, \ldots, 1]$ or $A = [1, 2, 3, \ldots, n]$ are worst-case inputs for this algorithm.

(b) We have two general approaches to try to prove this statement:

(1) figure out (and prove) an expression for $T(n)$, then show this expression is in $\Theta(n^2)$, or
(2) bound $T(n)$ above and below by expressions that grow like n^2.

Typically, the second way is easier (less stuff to prove).
Let's try the first way. First we must derive and prove an expression for $T(n)$:

Lemma 1: $T(n) = n^2 + 4n + 2$. [1]

Proof: To determine the number of steps the algorithm runs in the worst case, we need to determine the number of iterations in both the outer and inner loop.
Number of iterations in the outer for-loop:
The outer for-loop executes n iterations.
Number of iterations in the inner for-loop:
Recall from 1(a) that for a worst case array, the if condition in line 3 should be true for every iteration of the outer for-loop.
Therefore, line 6 is executed in every iteration of the outer for-loop.
So the inner for-loop is reached for all i, $0 \leq i \leq n - 1$.
For each value of i, the inner for-loop runs i iterations (for j set to each of $0, 1, \ldots, i-1$).
Total number of inner for-loop iterations, for all values of i
$= 0 + 1 + 2 + \cdots + (n - 1)$
$= \frac{n(n-1)}{2}$

So total number of lines executed
$= 2 \cdot \frac{n(n-1)}{2}$ (lines 6-7) $+5n$(lines 2-6) $+2$ (lines 1-2)
$= n^2 - n + 5n + 2$
$= n^2 + 4n + 2.$ □

[1] From where do we get this expression? We will derive it as we prove the lemma.

Now we make use of Lemma 1 and prove $T(n) \in \Theta\left(n^2\right)$

Let $c_1 = 1$. Then $c_1 \in \mathbb{R}^+$.
Let $c_2 = 7$. Then $c_2 \in \mathbb{R}^+$.
Let $B = 1$. Then $B \in \mathbb{N}$.
Assume $n \in \mathbb{N}$.
Assume that $n \geq B$.
Then $c_1 n^2 = n^2 \leq n^2 + 4n + 2 = T(n)$ by Lemma 1.
Also, since $n \geq 1$, $T(n) = n^2 + 4n + 2 \leq n^2 + 4n^2 + 2n^2 = 7n^2 = c_2 n^2$.
Thus $c_1 n^2 \leq T(n) \leq c_2 n^2$.
So $n \geq B \Rightarrow c_1 n^2 \leq T(n) \leq c_2 n^2$.
Since n is arbitrary, $\forall n \in \mathbb{N}, n \geq B \Rightarrow c_1 n^2 \leq T(n) \leq c_2 n^2$.
Thus $\exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow c_1 n^2 \leq T(n) \leq c_2 n^2$.
Thus $\exists c_2 \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow c_1 n^2 \leq T(n) \leq c_2 n^2$.
Thus $\exists c_1 \in \mathbb{R}^+, \exists c_2 \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow c_1 n^2 \leq T(n) \leq c_2 n^2$.

Now let's try the second way instead. We want to show $T(n) \in \Theta(n^2)$ without having to know an exact expression for $T(n)$.

Let $c_1 = \frac{1}{4}$. Then $c_1 \in \mathbb{R}^+$.
Let $c_2 = 10$. Then $c_2 \in \mathbb{R}^+$.
Let $B = 2$. Then $B \in \mathbb{N}$.
Assume $n \in \mathbb{N}$.
Assume that $n \geq B$.
To show $c_1 n^2 \leq T(n)$:
On our array A from 1(a), we count the number of times line 7 is executed. The algorithm executes the outer for loop body n times, and the if statement is true each time, so the inner loop is executed every time.
The inner loop body is executed i times for $i = 0, \ldots, n-1$, so line 7 is executed at least $0 + 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2}$ times.
Thus $T(n) \geq \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} \geq \frac{n^2}{2} - \frac{n^2}{4} = \frac{n^2}{4} = c_1 n^2$ since $n \geq 2$.

To show $T(n) \leq c_2 n^2$:
Now, looking at the algorithm, the outer loop executes no more than n times, and the inner loop executes no more than n times per iteration of the outer loop (since i is no more than n).
So the total number of lines executed must be no more than
$2n^2$ (lines 6-7) + $5n$ (lines 2-6) + 2 (lines 1-2).
Thus $T(n) \leq 2n^2 + 5n + 2 \leq 2n^2 + 5n^2 + 2n^2 = 9n^2 \leq c_2 n^2$.

Hence $c_1 n^2 \leq T(n) \leq c_2 n^2$.
So $n \geq B \Rightarrow c_1 n^2 \leq T(n) \leq c_2 n^2$.
Since n is arbitrary, $\forall n \in \mathbb{N}, n \geq B \Rightarrow c_1 n^2 \leq T(n) \leq c_2 n^2$.
Thus $\exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow c_1 n^2 \leq T(n) \leq c_2 n^2$.
Thus $\exists c_2 \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow c_1 n^2 \leq T(n) \leq c_2 n^2$.
Thus $\exists c_1 \in \mathbb{R}^+, \exists c_2 \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow c_1 n^2 \leq T(n) \leq c_2 n^2$.

3. [15 marks] = 4 + 3 + 2 + 2 + 4 marks each part

(a) Total number of real numbers representable =
number of positive real numbers + number of negative real numbers + 1 (for zero)

Finding the number of positive real numbers:
The system is normalized, so for each exponent, the numbers are in the format:
$1.b_1b_2b_3b_4b_5b_6$, where $b_i \in \{0, 1\}$
Therefore, for each exponent, we have 2^6 different real numbers.

Total positive real numbers
= number of exponents × number of positive real numbers for each exponent
$= 16 \text{ x } 2^6 = 2^4 \times 2^6 = 2^{10} = 1024$

Finding the number of negative real numbers:
Same as the number of positive real numbers

So total number of real numbers representable $= 2^{10} + 2^{10} + 1 = 2^{11} + 1 = 2049$

(b) Largest positive real number $= (1.111111 \times 2^8)_2 = 2^8 + 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 = 508$
Smallest positive real number $= (1.00000 \times 2^{-7})_2 = 2^{-7} = 0.0078125$

(c) Overflow will result if a positive real number is greater than the largest positive real number representable. 509 is one such number.

(d) Underflow will result if a positive real number is less than the smallest positive real number representable. 0.001 is one such number.

(e) The neighbouring numbers in the system are:
$l = (1.100100 \times 2^2)_2$ and $u = (1.100110 \times 2^2)_2$
The range of real numbers that are approximated by x' must lie between l and u. The range must also not include any real number that is within $(0.0000001 \times 2^2)_2$ from u and l (otherwise to represent that number, we would round to nearest and get u or l, not x').
So the range lies between
$l + (0.0000001 \times 2^2)_2 = 6.28125$, inclusive
and
$u - (0.0000001 \times 2^2)_2 = 6.34375$, exclusive.
In other words, $[6.28125, 6.34375)$.

4. [9 marks] $= 2 + 2 + 2 + 3$ each part

(a) We note that we can represent x exactly as 2.04×10^0. Now, to compute the expression, we first need to compute the expression $1 - x$, multiply it by itself, then subtract from 1, representing each answer in the floating point system.

$$1 - x = 1.00 \times 10^0 - 2.04 \times 10^0 \approx -1.04 \times 10^0$$
$$(1 - x)^2 = (-1.04 \times 10^0)(-1.04 \times 10^0) = 1.0816 \approx 1.08 \times 10^0$$
$$1 - (1 - x)^2 \approx 1.00 \times 10^0 - 1.08 \times 10^0 = -0.08 = -8.00 \times 10^{-2}$$

(b) To compute $2x - x^2$, we first determine the value of each term, then compute the difference, representing each answer in the floating point system.

$$2x = (2.00)(2.04) = 4.08 = 4.08 \times 10^0$$
$$x^2 = (2.04)(2.04) = 4.1616 \approx 4.16 \times 10^0$$
$$2x - x^2 \approx 4.08 \times 10^0 - 4.16 \times 10^0 = -0.08 = -8.00 \times 10^{-2}$$

(c) To compute $x(2-x)$, we first determine the value of $2-x$, then compute the product, representing each answer in the floating point system.

$$2 - x = 2.00 - 2.04 = -0.04 = -4.00 \times 10^{-2}$$
$$x(2-x) \approx (2.04)(-4.00 \times 10^{-2}) = -0.0816 \approx -8.16 \times 10^{-2}$$

(d) We note that the exact value of the expression is $-0.0816 = -8.16 \times 10^{-2}$.
The relative error for the value computed in parts (a) and (b) is:
$\frac{|-0.0816 - -0.08|}{|-0.0816|} \approx 0.0196 \approx 2\%$
The relative error for the value computed in parts (a) and (b) is:
$\frac{|-0.0816 - -0.0816|}{|-0.0816|} = 0$
The evaluation in part (c) gives a smaller relative error, and so the formula in part (c) is more stable for this value of x.