A1.

[Part A]:
All (i), (ii), and (iii) should happen together. Here is the updated table per each case:

Expression	Always / Sometimes / Never
(case 0) `P1OUT = P1OUT | BIT0`	**Sometimes**
(case 1) `P1OUT = P1OUT & ~BIT7`	**Never**
(case 2) `P1OUT = (P1OUT | BIT0) & ~BIT7`	**Never**
(case 3) `P1OUT = (P1OUT & ~BIT7) | BIT0`	**Never**
(case 4) `P1OUT = P1OUT ^ (BIT0 | BIT7)`	**Sometimes**

(case 0) `| BIT0` guarantees red ON but does not force `BIT7=1` (green OFF), so it depends on the prior value of `BIT7`.
(case 1) `& ~BIT7` makes green ON and does not force `BIT0=1` (red ON), so the final state never happens.
(case 2) `| BIT0` guarantees red ON, but `& ~BIT7` makes green ON, so the final state never happens.
(case 3) `& ~BIT7` makes green ON, and `| BIT0` guarantees red ON, so the final state never happens.
(case 4) XOR toggles both bits, so outcome depends on prior state.

[Part B]:
It is NOT correct.
It is because `&` binds tighter than `|`, so it parses as `(P1OUT & ~BIT1) | ~BIT3`, and `~BIT3` has many 1s, unintentionally setting unrelated bits. Also, `~(A|B)` $\neq$ `~A | ~B`. So, the correct single expression is:

```
P1OUT = P1OUT & ~(BIT1 | BIT3)
```

A2.

[Part A]:
In Up Mode, the timer counts $0 \rightarrow$ CCR0 (inclusive), so the cycle length in ticks is

$$N = \text{CCR0} + 1 = 49{,}999 + 1 = 50{,}000.$$

Period

$$T = \frac{N}{f} = \frac{50{,}000}{1{,}000{,}000} = 0.05 \text{ s} = \boxed{50 \text{ ms}}.$$

[Part B]:
Interrupts per second

$$\boxed{\frac{1}{T} = \frac{1}{0.05} = 20 \text{ interrupts/s}}.$$

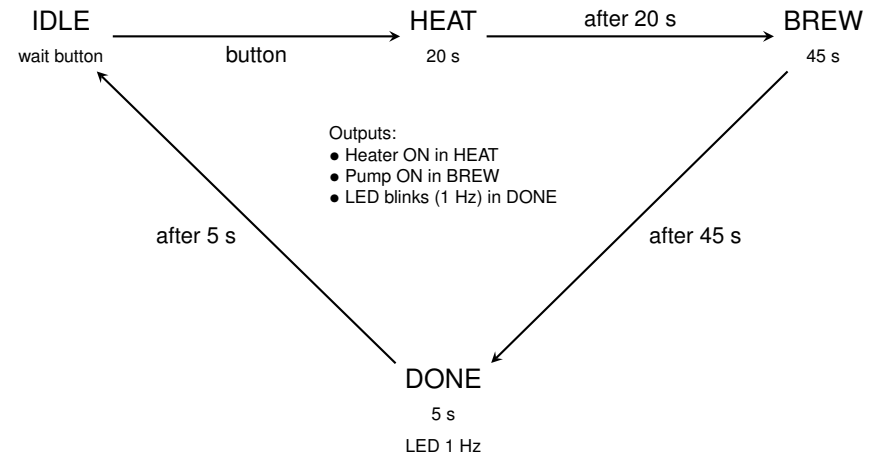
[Part C]:
Target: 10 Hz $\Rightarrow T = 0.1$ s. In Up Mode,

$$\text{CCR0} = fT - 1 = (1{,}000{,}000)(0.1) - 1 = 100{,}000 - 1 = \boxed{99{,}999}.$$

Since CCR0 is 16 bits, $\boxed{99{,}999}$ would be higher than it is maximum possible value (65,535). So, the current configuration of the timer does not work. The timer's configuration needs to be changed (use of divider or up/down mode).

A3.

[Part A]:



[Part B]:
We are using **ACLK** as the clock source, which runs at **32.768 kHz**, in Up mode with maximum divider (64).

$$f_{\text{eff}} = \frac{32768}{64} = 512 \text{ Hz}$$

Counts per state (CCR0 = counts -1):

$$\text{HEAT: } 20 \cdot 512 = \boxed{10{,}240} \quad (\text{CCR0} = 10{,}239)$$

$$\text{BREW: } 45 \cdot 512 = \boxed{23{,}040} \quad (\text{CCR0} = 23{,}039)$$

$$\text{DONE: } 5 \cdot 512 = \boxed{2{,}560} \quad (\text{CCR0} = 2{,}559)$$

$$\text{Ticks} = \text{Time (seconds)} \times \text{ACLK frequency (32,768 Hz)}$$

[Part C]:
At $f_{\text{eff}} = 512$ Hz, 1 s = 512 ticks. So, toggle every 0.5 s = 256 ticks. Configure `TA0CCTL1` in toggle mode, set `CCR1 = CCR1 + 256`. Each 256 ticks $\rightarrow$ toggle $\rightarrow$ 1 Hz square wave.

A4 [30 Points].

[Part A]:
We use

$$f_{\text{LCD}} = \frac{f_{\text{ACLK}}}{2^{\text{LCDPREx}} \cdot (\text{LCDDIVx} + 1)}.$$

Given $f_{\text{ACLK}} = 32768$ Hz and desired $f_{\text{LCD}} = 1$ Hz,

$$2^{\text{LCDPREx}} \cdot (\text{LCDDIVx} + 1) = \frac{32768}{1} = 32768.$$

With 3-bit `LCDPREx` (max $2^7 = 128$) and 5-bit `LCDDIVx` (max 32), the largest possible factor is

$$128 \times 32 = 4096 < 32768.$$

Not possible: the minimum achievable segment frequency is $\dfrac{32768}{4096} = 8$ Hz.

[Part B]:
No unique answer. It is only numbers, so the following is the memory values for the numbers:

```
"0" = 0x28  &   0xFC;
"1" = 0x20  &   0x60;
"2" = 0x00  &   0xDB;
"3" = 0x00  &   0xF1;
"4" = 0x00  &   0x67;
"5" = 0x00  &   0xB7;
"6" = 0x00  &   0xBF;
"7" = 0x00  &   0xE0;
"8" = 0x00  &   0xFF;
"9" = 0x00  &   0xF7;
```

Based on the each student's ID number, it goes to the following registers (here the example is UCF ID = 5412132, so the most right six digits are 412132):

```
LCDM10 = 0x67; // 4 on A1 (part 1)
LCDM11 = 0x00; // 4 on A1 (part 2)

LCDM6 = 0x60;  // 1 on A2 (part 1)
LCDM7 = 0x20;  // 1 on A2 (part 2)

LCDM4 = 0xDB;  // 2 on A3 (part 1)
LCDM5 = 0x00;  // 2 on A3 (part 2)

LCDM19 = 0x60;  // 1 on A4 (part 1)
LCDM20 = 0x20;  // 1 on A4 (part 2)

LCDM15 = 0xF1;  // 3 on A5 (part 1)
LCDM16 = 0x00;  // 3 on A5 (part 2)

LCDM8 = 0xDB;  // 2 on A6 (part 1)
LCDM9 = 0x00;  // 2 on A6 (part 2)
```