Q1.

a)

f(n) is O(g(n)) if:

$f(n) \leq c^* g(n)$ for $n \geq n_0$; $c \in \mathbb{R}$, $n_0 \in \mathbb{Z} \mid n_0 \geq 1$

$f(n) = \log_{10}(n^2)$
$f(n) = 2\log_{10}(n)$

$\log_2(n) = \log_{10}(n) / \log_{10}(2) = (1/\log_{10}(2)) * \log_{10}(n)$

and $(1/\log_{10}(2)) > 2$ for $n \geq 1$, so $f(n) \leq c^*\log_2(n)$ for $n \geq n_0$, where c = 1 and $n_0 = 1$. f(n) is $O(\log_2(n))$

b)

f(n) is Ω(g(n)) if:

$f(n) \geq c^* g(n)$ for $n \geq n_0$; $c \in \mathbb{R}$, $n_0 \in \mathbb{Z} \mid n_0 \geq 1$

$f(n) = n(10n^2 - 2sqrt(n)) = 10n^3 - 2sqrt(n)^3 = 10n^3 - 2n^{3/2}$
$g(n) = n^3$

$10n^3 \geq 10 n^3$ for $n \geq 0$
$-2n^{3/2} \geq -2n^3$ for $n \geq 1$

$10n^3 - 2n^{3/2} \geq 10n^3 - 2n^3$ for $n \geq 1$
$f(n) \geq 8n^3$ for $n \geq 1$
$f(n) \geq 8^* g(n)$ for $n \geq 1$
So, c = 8 and $n_0 = 1$. f(n) is $\Omega(n^3)$

c)

f(n) is Θ(g(n)) if:

$c_1^*g(n) \leq f(n) \leq c_2^* g(n)$ for $n \geq n_0$; $c_1,c_2 \in \mathbb{R}$, $n_0 \in \mathbb{Z} \mid n_0 \geq 1$

$f(n) = (n\sin(n))^2 + 100$
$g(n) = n^2$

$f(n) \leq c_2 * g(n)$
$n^2\sin^2(n) \leq n^2$ for $n \geq 0$
$100 \leq 100n^2$ for $n \geq 1$
$(n\sin(n))^2 + 100 \leq 101n^2$ for $n \geq 1$,=, so for this, $c_2 = 101$ and $n_0 = 1$

$c_1 * g(n) \leq f(n)$

Saying $n^2 \le n^2\sin^2(n) + 100$ is false due to oscillatory nature of sin(N). As a result, values of c_1 and n_0 cannot be found such that $c_1 * g(n) \le f(n)$ is satisfied (f(n) will periodically take value of 100, which will be less than value of n^2 for n values past 10)

So, f(n) is not $\Theta(n^2)$

Q2)

a)

Worst case: loop runs until condition $i^2 \le n$ is broken. If such case happens, $i^2 > n$ is true and so i > sqrt(n). So, this would run sqrt(n) - 2 + 1 = sqrt(n) - 1 , which is O(sqrt(n)).

b)

Best case: Immediately exiting loop if (n%i == 0). Since i initially equals 2, input n that is multiple of 2 (n = 2k; $k \in \mathbb{Z}$). This would only be 1 iteration. O(1).

Q3

a)

temp is a variable, in is original stack, and tmp is temporary stack

```
while (¬ in.isEmpty())
    Temp = in.pop()
    while (¬ temp.isEmpty() ∧ temp.peek() > temp)
        in.push(tmp.pop())
    tmp.push(temp)
while (¬ tmp.isEmpty())
    in.push(tmp.pop())
```

b)

$O(n^2)$

Worst case: input is (bottom to top) increasing. Each popped element from input would require n pushes back to input stack; $n*n = n^2$

Q4

a)

```
for (int i = 1; i <= 3; i++)
    Q.enque(D.removeFirst())
for (int i = 1; i <= 3; i++)
    Q.enque(D.removeLast())
Q.enque(D.removeFirst())
Q.enque(D.removeLast())
for (int i = 1; i <= 8; i+=)
    Q.enque(D.removeFirst())
D.addFirst(Q.deque())
for (int i = 1; i <= 4; i++)
    D.addLast(Q.deque())
for (int i = 1; i <= 3; i++)
```

D.addFirst()

return (D)

b)

O(n). no nested loops are present, and largest number of execution by loops is 8, which is equal to n in this case.

c)

O(n). Supposing D is given, we are using initially empty queue, Q, to perform algorithm. Q is only variable used in algorithm, and may at most contain the same number of elements as D.