Short Answer

8. a)
```python
n = input('enter n: ')  # 1 operation
for i in range(n):  # n iterations
    x = 2 * n  # 2 operations (math *, assignment)
    while x > 1:
        x = x/2  # log_2 x + 1 iterations (because x is reduced at each iteration - it becomes half of the previous x's value - like in binary search)
```

\[
T(n) = 1 + n(2+\log n + 1) = n\log n + 3n + 1
\]

T(n) = \(\Theta(n \log n)\) - asymptotic running time

b) T(n) = \(\Theta(n)\)
```python
n = input('enter n: ')  # 1 operation
total = 0  # 1 operation
for i in range(n):  # n iterations
    for j in range(10000):  # 10 000 iterations
        total += j  # 1 operation
print total  # 1 operation
```

\[
T(n) = 3 + 10000n
\]

Since the asymptotic running time is discussed when \(n \to \infty\), 10000 becomes insignificantly small there, hence T(n) = \(\Theta(n)\)

c) T(n) = \(\Theta(n^2)\)
```python
n = input('enter n: ')  # 1 operation
for i in range(2*n):  # 2n + (2n-1) + (2n-2) + (2n-3) + ... + 2 + 1 iterations
    for j in range(i,n):  # 1 operation
        total += j
for j in range(n):  # n iterations
    total += j  # 1 operation
```

\[
T(n) = 2 + 2n^2 + n + n = 2n^2 + 2n + 1, \text{ hence } T(n) = \Theta(n^2)
\]

\[
1 + 2 + 3 + \ldots + 2n = \frac{2n(1+2n)}{2} = 2n^2 + n
\]