I have derived and implemented in Python an algorithm that executes in $O(1)$ time.

Through the power of Markov chains, I simultaneously derived closed-form expressions for the n^th even Fibonacci number and the sum of the first n even Fibonacci numbers. The summation equation is evidently useful, but the n^th term equation on its own will not help to solve Problem 2. However, its inverse (which takes in a Fibonacci number and returns n) can be used to determine how many even Fibonacci numbers there are below a certain upper bound (four million, per se). Altogether, these equations supply a fully algebraic solution to Problem 2 with a Pythonic implementation (as mentioned) in $O(1)$ time.

My derivation includes some fairly involved linear algebra, so, because of length considerations, I will only hit the highlights in my post here. A full description of my method can be found on [url=https://github.com/shadypuck/SumEvenFibonacci/blob/master/SumEvenFibonacci.pdf]GitHub[/url].

Proof inside!
[collapse]
[b]Groundwork[/b]
Let's begin. The basis for my method lies in the oft-mentioned fact that every third Fibonacci number is an even Fibonacci number, and the even Fibonacci numbers can be defined recursively by$$E_n=4E_{n-1}+E_{n-1}$$for $n\geq 2$ with initial conditions
$$\begin{align}
E_0 &= 2&
E_1 &= 8
\end{align}$$

[b]Notation[/b]
Before beginning the main derivation, a note on notation: From here on out, e_n refers to the n^th term in the even Fibonacci sequence, where 2 is the 0^th term. Likewise, s_n refers to the sum of all terms through the n^th term in the even Fibonacci sequence. Additionally, E_n refers to a vector in the following form.
$$
E_n =
\begin{bmatrix}
s_n\\
e_n\\
e_{n-1}\\
\end{bmatrix}
$$
Lastly, E refers to the matrix that linearly maps E_{n-1} to E_n.

[b]Formulas for the Even Fibonacci Numbers and Their Sum[/b]
My derivation solves the following Markov chain.
$$
\underbrace{
\begin{bmatrix}
s_n\\
e_n\\
e_{n-1}\\
\end{bmatrix}
}_{E_n}
=
\underbrace{
\begin{bmatrix}
1 & 4 & 1\\
0 & 4 & 1\\
0 & 1 & 0\\
\end{bmatrix}
}_E
\underbrace{
\begin{bmatrix}
s_{n-1}\\
e_{n-1}\\
e_{n-2}\\
\end{bmatrix}
}_{E_{n-1}}
$$
The initial conditions vector for this Markov chain is as follows.
$$
E_1 =
\begin{bmatrix}
10\\
8\\
2\\
\end{bmatrix}
$$
The absolute definition of the Markov chain is $E_n=E^{n-1}E_1$. To facilitate raising E to an arbitrarily high power, diagonalization will be used. E diagonalizes as follows.
$$
E = \frac{1}{32\sqrt{5}}
\begin{bmatrix}
1 & 11+5\sqrt{5} & 11-5\sqrt{5}\\
0 & 8+4\sqrt{5} & 8-4\sqrt{5}\\
0 & 4 & 4\\

\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0\\
0 & 2+\sqrt{5} & 0\\
0 & 0 & 2-\sqrt{5}\\
\end{bmatrix}
\begin{bmatrix}
32\sqrt{5} & -40\sqrt{5} & -8\sqrt{5}\\
0 & 4 & -8+4\sqrt{5}\\
0 & -4 & 8+4\sqrt{5}\\
\end{bmatrix}
$$
Compiling $E_n=S\Lambda^{n-1}S^{-1}E_1$ leads to closed-form expressions for s_n and e_n.
$$
\begin{align}
\begin{bmatrix}
s_n\\
e_n\\
e_{n-1}\\
\end{bmatrix}
&= \frac{1}{32\sqrt{5}}
\begin{bmatrix}
1 & 11+5\sqrt{5} & 11-5\sqrt{5}\\
0 & 8+4\sqrt{5} & 8-4\sqrt{5}\\
0 & 4 & 4\\
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0\\
0 & 2+\sqrt{5} & 0\\
0 & 0 & 2-\sqrt{5}\\
\end{bmatrix}^{n-1}
\begin{bmatrix}
32\sqrt{5} & -40\sqrt{5} & -8\sqrt{5}\\
0 & 4 & -8+4\sqrt{5}\\
0 & -4 & 8+4\sqrt{5}\\
\end{bmatrix}
\begin{bmatrix}
10\\
8\\
2\\
\end{bmatrix}\\
&=

\begin{bmatrix}
\frac{(5-\sqrt{5})(2+\sqrt{5})^{n+2}+(5+\sqrt{5})(2-\sqrt{5})^{n+2}-10}{20}\\
\frac{(2+\sqrt{5})^{n+1}-(2-\sqrt{5})^{n+1}}{\sqrt{5}}\\
\frac{(2+\sqrt{5})^n-(2-\sqrt{5})^n}{\sqrt{5}}\\
\end{bmatrix}
\end{align}
$$

The uppermost value in E_n corresponds to an explicit formula for s_n, and the middle value in E_n corresponds to an explicit formula for e_n, as desired. The results are transcribed below for clarity.
$$
s_n = \frac{1}{20}\left(\left(5-\sqrt{5} \right)\left(2+\sqrt{5} \right)^{n+2}+\left(5+\sqrt{5} \right)\left(2-\sqrt{5} \right)^{n+2}-10 \right)\tag{1}
$$
$$
e_n = \frac{1}{\sqrt{5}}\left(\left(2+\sqrt{5} \right)^{n+1}-\left(2-\sqrt{5} \right)^{n+1} \right)\tag{2}
$$

[b]Inverting e[sub]n[/sub][/b]
It is not algebraically possible to exactly invert Equation 2. However, a sort-of inverse can be found, the upper-bound function on which is shown below.
$$
n = \log_{2+\sqrt{5}}\left(\frac{e_n\sqrt{5}+\sqrt{5e_n^2+4}}{2} \right)-1\tag{3}
$$
[/collapse]
[b]Solving Problem 2[/b]
[i]Mathematically[/i]
Employ Equation 3 to find an upper bound on the number of even Fibonacci numbers beneath four million.
$$
\begin{align}
n &= \log_{2+\sqrt{5}}\left(\frac{4000000\sqrt{5}+\sqrt{5(4000000)^2+4}}{2} \right)-1\\
&\approx 10.0876
\end{align}
$$
Since the upper bound is a decimal between 10 and 11, the 10^th even Fibonacci number is the greatest even Fibonacci number under four million.
Plug $n=10$ into Equation 1 to find the sum of the even Fibonacci numbers beneath four million, solving Problem 2.
$$
\begin{align}

s_{10} &= \frac{1}{20}\left((5-\sqrt{5})(2+\sqrt{5})^{10+2}+(5+\sqrt{5})(2-\sqrt{5})^{10+2}-10 \right)\\
s_{10} &= \boxed{4613732}
\end{align}
$$

[i]Pythonically[/i]
The following is a simple implementation of the above in Python.
[code=Python]
```
import math

def sumEvenFibonaccisBeneath(bound):
    n = math.floor(math.log((bound*5**0.5+(5*bound**2+4)**0.5)/2,2+5**0.5)-1)
    return int(1/20*((5-5**0.5)*(2+5**0.5)**(n+2)+(5+5**0.5)*(2-5**0.5)**(n+2)-10))


print(sumEvenFibonaccisBeneath(4*10**6))
```
[/code]