Problem 1. (10 points) Give a tight bound (using the big-theta notation) on the number of Hi’s produced by the following method as a function of \( n \). For simplicity, you can assume \( n \) to be a power of two.

**Algorithm Loopy** \((n : \text{integer})\)

\[
i \leftarrow 1 \\
\text{while } i \leq n \text{ do} \\
\quad \text{for } j \leftarrow i \text{ to } 2i - 1 \text{ do} \\
\quad \quad \text{print “Hi”} \\
\quad i \leftarrow 2i
\]

**Answer:** When \( i = 1 \) the print statement is executed once because \( j \) ranges from 1 to 1. When \( i = 2 \) the print statement is executed twice because \( j \) ranges from 2 to 3. \ldots When \( i = n/2 \) the print statement is executed \( n/2 \) times because \( j \) ranges from \( n/2 \) to \( n - 1 \). When \( i = n \), the print statement is executed \( n \) times because \( j \) ranges from \( n \) to \( 2n - 1 \). The total is \( n + n/2 + \ldots + 2 + 1 \in \Theta(n) \).

Problem 2. (10 points)

Imagine that you are facing a high wall that stretches infinitely in both directions. There is a door in the wall, but you don’t know how far away is the door or in which direction. It is pitch dark, but you have a very dim lighted candle that will enable you to see the door when you are right next to it.

1. Show an algorithm that enables you to find the door by walking at most \( O(n) \) steps in the worst case, where \( n \) is the number of steps that you would have taken if you knew where the door is and walked directly to it (note that your algorithm does not know the value of \( n \) in advance)

2. What is the constant multiple in the worst-case analysis for your algorithm?

**Answer:** First, note that even if we knew the true distance \( n \), we would need \( 3n \) steps in the worst case. Can linear-time be achieved and how bad is the constant when we do not know \( n \)? Consider the following algorithm.
1. $k \rightarrow 1$
2. take $k$ steps on left
3. if door found then STOP
4. take $k$ steps on the right (back to the origin)
5. $k \rightarrow 2 \cdot k$
6. take $k$ steps on the right
7. if door found then STOP
8. take $k$ steps on the left (back to the origin)
9. $k \rightarrow 2 \cdot k$
10. goto 2.

**Analysis:** First, assume that $n = 2^q$. Then, the number of steps would be

$$1 + 1 + 2 + 2 + \ldots + 2^q + 2^q = 2 (2^{q+1} - 1) = 4 \cdot 2^q - 2 = 4n - 2$$

In general, however, $n$ is not a power of two. Let us set $q = \lfloor \log_2 n \rfloor$. Note that $2^q < n < 2^{q+1}$. Because we did not find the door at $2^q$, the algorithm will go back to the origin, go in the other direction $2^{q+1}$ steps, come back, and finally travel $n$ positions to the door. Therefore, we have

$$1 + 1 + 2 + 2 + \ldots + 2^q + 2^q + 2^{q+1} + 2^{q+1} + n = 2 (2^{q+2} - 1) + n = 8 \cdot 2^q - 2 + n = 9n - 2$$

So the constant is 4 when $n$ is a power of two, 9 otherwise (worst-case).

The complexity of the algorithm is $O(n)$, although different algorithms may result in different constants. In fact, what do you think would be the best exponential step in order to minimize the constant?

**Problem 3.** (10 points)

Given the following recurrence relation

$$T(n) = \begin{cases} 1 & n = 1 \\ 4T \left( \frac{n}{2} \right) + 3 & n > 1 \end{cases}$$

1. Solve it exactly (i.e., without using any asymptotic notation) by iterative substitutions
2. Prove by induction that your solution is correct

**Answer:** We have

$$T(n) = 4^i T(n/2^i) + \left(4^{i-1} + 4^{i-2} + \ldots + 1\right) \cdot 3$$

$$= 4^i T(n/2^i) + 4^i - 1$$
now we set $n/2^i = 1$ which is $i = \log_2 n$ and we get

$$
T(n) = 4^{\log_2 n}T(1) + 4^{\log_2 n} - 1 \\
= 2n^2 - 1
$$

We now prove by induction that $T(n) = 2n^2 - 1$ is the correct solution of the recurrence relation.

Base case ($n = 1$). $T(1) = 2 \cdot 1^2 - 1 = 1$.

Induction step. Assume the statement true for $n/2$, that is

$$
T(n/2) = 2(n/2)^2 - 1.
$$

We have

$$
T(n) = 4T(n/2) + 3 \\
= 4 \left(2(n/2)^2 - 1\right) + 3 \\
= 8 \frac{n^2}{4} - 1 \\
= 2n^2 - 1
$$