Thm

For every natural number \( n \), there exists \( p \in \mathbb{N} \), and bits \( b_p, b_{p-1}, \ldots, b_0 \in \{0, 1\} \) such that \( n = \sum_{i=0}^{p} b_i 2^i \).

Related:

For all \( m \in \mathbb{N} \), every natural number \( \leq m \) has a binary representation.

Translate:

\[ \forall m \in \mathbb{N}, \ (\forall n \in \mathbb{N}, \ n \leq m \implies \left( \exists p \in \mathbb{N}, \exists b_p, b_{p-1}, \ldots, b_0 \in \{0, 1\}, \ n = \sum_{i=0}^{p} b_i 2^i \right) ) \]

Call this \( P(m) \), where \( m \in \mathbb{N} \).

Form: \( \forall m \in \mathbb{N}, \ P(m) \).

Proof: by induction

Base case: Let \( m = 0 \). Prove \( P(0) \).

Let \( n \in \mathbb{N} \) and assume \( n \leq m = 0 \).
There is only one number to consider, \( n = 0 \).

\[
0 = 0.2^0
\]

Let \( p = 0 \) and \( b_0 = 0 \).

Then

\[
\sum_{i=0}^{p} b_i 2^i = b_0 2^0 = 0.1 = 0 = n, \text{ as required.}
\]

**Inductive Step:**

Let \( k \in \mathbb{N} \) and assume \( P(k) \).

That is, every natural number less than or equal to \( k \) has a binary representation.

We want to prove \( P(k+1) \).

Let \( n \in \mathbb{N} \) and assume \( n \leq k+1 \).

Need to prove that every number \( n \leq k+1 \) has a binary representation.

**Case:** \( n \leq k \).

The by the inductive hypothesis, \( n \) has a binary representation.
case: \( n = k+1 \).

**case 1:** Assume \( n = k+1 \) is even.

i.e. \( \exists q \in \mathbb{N}, \ n = 2q \).

Let \( r \in \mathbb{N} \) be such that \( n = 2r \).

So \( r = n/2 \), by divisibility property, we know \( r \leq k \).

Therefore by the induction hypothesis, we can write

\[
    r = \sum_{i=0}^{p} b_i 2^i
\]

for some \( p, b_0, \ldots, b_p \in \mathbb{N} \).

\[
    \therefore \ n = 2 \left( \sum_{i=0}^{p} b_i 2^i \right)
\]

\[
    = \sum_{i=0}^{p} b_i 2^{i+1}
\]

\[
    = b_0 2^1 + b_1 2^2 + b_2 2^3 + \ldots + b_p 2^{p+1}
\]

\[
    = c_0 2^0 + c_1 2^1 + c_2 2^2 + \ldots + c_p 2^p
\]

\[
    = \sum_{i=0}^{p} c_i 2^i
\]

where we let \( p' = p+1 \)

and \( c_i = b_i \) for \( i \in \{0, \ldots, p\} \).
So we have a binary representation for $n$, as reg'd.

**Case 2:** Assume $n = k + 1$ and $n$ is odd.

i.e. $\exists q \in \mathbb{N}, \ n = 2q + 1$

Let $s \in \mathbb{N}$ be such that $n = 2s + 1$.

As before, we know $s < n$

$s \leq k$

and we can write

$s = \sum_{i=0}^{p} b_i 2^i$

for some $p \in \mathbb{N}$

and $b_0, b_1, \ldots, b_p \in \{0, 1\}$.

Since $n = 2s + 1$

\[ = 2 \left( \sum_{i=0}^{p} b_i 2^i \right) + 1 \]

\[ = \left( \sum_{i=0}^{p} b_i 2^{i+1} \right) + 1 \cdot 2^0 \]

\[ = 1 \cdot 2^0 + b_0 2^1 + b_1 2^2 + \ldots + b_p 2^{p+1} \]

\[ = d_0 \cdot 2^0 + d_1 \cdot 2^1 + d_2 \cdot 2^2 + \ldots + d_p \cdot 2^p \]
\[ \sum_{i=0}^{n} 2^i \]

where \( p' = p + 1 \), \( d_0 = 1 \) and \( d_i = b_{i-1} \) for \( i \in \{1, 2, 3, \ldots, p'\} \)

\[ \therefore P(k+1) \text{ is True} \]

We can conclude that every natural number has a binary representation.

---

**Ch 5  Analyzing Algorithm Running Time (runtime)**

**Focus:** write a program that computes the right result.
- write programs that follow design and style guidelines
- write programs that finish on time
- write programs that use algorithms with suitable runtimes.

**How to measure runtime?**
Stop watch approach.

Flaws: measurement could be affected by:
- CPU, memory, cache, other computations

Hard to extend to questions about how runtime changes as amount of data changes.

Another issue:

Can be situation dependent.

Insertion sort:
- List almost sorted:
  - Runtime \( \sim n \) (length of list)
- List almost in reverse order:
  - Runtime \( \sim n^2 \)

- Worst case runtime (upper bound)
- Best case runtime (lower bound)

- Average case for "average" input.

Need a formal presentation to confirm results.

---

How to describe the runtime of the following:
```python
def print_items(lst: List[bool]) -> None:
    for item in lst:
        print(item)
```

- try to count the number of "basic operations"
  - list length \( n \)
  - \( n \) print operations

1. describe runtime as \( n \) basic operations

2. try to also account for time it takes for item to get values
   - \( L \) happens \( n \) times

2. describe runtime as \( n + n \)
   - \( = 2n \) basic operations

3. try to also account for relative time for print vs. variable assignment
   - Suppose 10:1

3. describe runtime as \( 10n + n \)
   - \( = 11n \) basic operations

4. try to count the cost of calling and returning from \( f \).
   - constant cost: 1.5 basic operations
4. describe runtime as $1.5 \ln n + 1.5$

Summary:

- $n$: basic operations
- $2n$: linear
- $1.5 \ln n$: logarithmic
- $1.5 \ln n + 1.5$: mixed

Q: Which is right?
A: Depends on unknown/unknown factors.

Can take away: all descriptions grow linearly.

New Q: "How does the runtime change as the size of the list changes?"

Try to describe formally how runtime grows as problem size increases

→ Big-Oh descriptions