1 Parallel Model of Computation

- Example: Finding the sum of $n$ numbers.
- Under the parallel model of computation, multiple processors can compute at the same time using shared memory.
- Two measures of time complexity in parallel computation:
  - **Number of Parallel Steps**: How long an algorithm would take if we had access to an unlimited number of processors.
  - **Total Amount of Work**: How long an algorithm would take if we had only one processor.
- We can find the sum of $n$ numbers in $O(\log n)$ steps and $O(n)$ work.

2 List Ranking

- We are given a linked list. For each node in the list, we want to find how far the node is from the end of the list.
- Without parallelization, this takes $O(n)$ time.
- For every node $v$, a processor runs the following code $\log n$ times in parallel:

```python
if next[v] != None:
    count[v] = count[v] + count[next[v]]
    next[v] = next[next[v]]
```

- $\text{next}[v]$ initially points to the next node in the list.
- $\text{count}[v]$ keeps track of the number of nodes between $v$ and $\text{next}[v]$ and is initialized to 1.
- Warning: In the second line, every processor must read $\text{count}[\text{next}[v]]$ before any processor updates $\text{count}[v]$. A similar restriction is necessary in the third line.
- This takes $O(\log n)$ steps and $O(n \log n)$ work.
3 Randomized List Ranking

- Another way to compute list ranking is to contract the list by recursively getting rid of every other node until there is only one node left.

- But we cannot pick every other node deterministically in constant time without the list ranking.

- Instead, we repeat the following until only one node is left:
  - At the beginning of every iteration, we flip a coin for every node $v$. Let the result be $\text{coin}[v]$.
  - If $\text{coin}[v]$ is heads and $\text{coin}[\text{next}[v]]$ is tails, we contract the nodes $v$ and $\text{next}[v]$.

- This takes $O(\log n)$ steps and $O(n)$ work on average.