1. Sort the coin values. Suppose the first $i$ coins can make all sums in $[0, s_i]$ where $s_i \triangleq \sum_{j=1}^{i} v_i$. If $v_{i+1} > s_i + 1$, then we cannot make the sum $s_i + 1$ no matter how we use the coins $v_{i+1} \sim v_n$. Otherwise we can use the first $i+1$ coins to make all sums in $[0, s_{i+1}]$ (and it's obvious that we cannot make any sum $> s_{i+1}$ using the first $i+1$ coins). $O(\text{sort}(n))$.
2. Use multiway-partitioning to partition the input numbers into $O(n^{1-\epsilon})$ blocks each with size $O(n^{\epsilon})$ in $O(n)$ time [1], such that the blocks are in sorted order, but the numbers within each block may not be sorted. The algorithm proceeds in rounds, each round we add all coin values no larger than the sum at the beginning of the round. After every two rounds, the sum will at least double. Each round takes $O(n^{\epsilon} \cdot (1 + \#\text{ blocks used}))$ by applying selection on the $\leq 2$ partial blocks within the query range, so the running time is $O(n + n^{\epsilon} \log U)$, which is $O(n)$ for all $U = 2^{O(n^{1-\epsilon})}$.

Remark. many proofs in the discuss zone are flawed.

# References


[1] Yijie Han and Mikkel Thorup. Integer sorting in $O(n\sqrt{\log \log n})$ expected time and linear space. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 135–144. IEEE, 2002.