# A Simple Near-Linear Pseudopolynomial Time Randomized Algorithm for Subset Sum

**Ce Jin**
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
jinc16@mails.tsinghua.edu.cn

**Hongxun Wu**
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
wuhx18@mails.tsinghua.edu.cn

## Abstract

Given a multiset $S$ of $n$ positive integers and a target integer $t$, the Subset Sum problem asks to determine whether there exists a subset of $S$ that sums up to $t$. The current best deterministic algorithm, by Koiliaris and Xu [SODA'17], runs in $\tilde{O}(\sqrt{n}t)$ time, where $\tilde{O}$ hides poly-logarithm factors. Bringmann [SODA'17] later gave a randomized $\tilde{O}(n+t)$ time algorithm using two-stage color-coding. The $\tilde{O}(n+t)$ running time is believed to be near-optimal.

In this paper, we present a simple and elegant randomized algorithm for Subset Sum in $\tilde{O}(n+t)$ time. Our new algorithm actually solves its counting version modulo prime $p > t$, by manipulating generating functions using FFT.

**2012 ACM Subject Classification** Theory of computation → Algorithm design techniques

**Keywords and phrases** subset sum, formal power series, FFT

**Digital Object Identifier** 10.4230/OASIcs.SOSA.2019.17

**Acknowledgements** The authors would like to thank the anonymous reviewers for their helpful comments.

## 1 Introduction

Given a multiset $S$ of $n$ positive integers and a target integer $t$, the Subset Sum problem asks to determine whether there exists a subset of $S$ that sums up to $t$. It is one of Karp's original NP-complete problems [9], and is widely taught in undergraduate algorithm classes. In 1957, Bellman gave the well-known dynamic programming algorithm [2] in time $O(nt)$. Pisinger [12] first improved it to $O(nt/\log t)$ on word-RAM models. Recently, Koiliaris and Xu gave a deterministic algorithm [10, 11] in time $\tilde{O}(\sqrt{n}t)$, which is the best deterministic algorithm so far. Bringmann [4] later improved the running time to randomized $\tilde{O}(n+t)$ using color-coding and layer splitting techniques. Abboud et al. [1] recently showed that Subset Sum has no $O(t^{1-\epsilon}n^{O(1)})$ algorithm for any $\epsilon > 0$, unless the Strong Exponential Time Hypothesis (SETH) is false, so the $\tilde{O}(n+t)$ time bound is likely to be near-optimal.

In this paper, we present a new randomized algorithm matching the $\tilde{O}(n+t)$ running time by Bringmann [4]. The basic idea of our approach is quite straightforward. For prime $p > t$, we give an $\tilde{O}(n+t)$ algorithm for $\#_p$Subset Sum, the counting version of Subset Sum problem modulo $p$. Then the decision version can be solved with high probability by randomly picking a sufficiently large prime $p$.


2nd Symposium on Simplicity in Algorithms (SOSA 2019).
Editors: Jeremy Fineman and Michael Mitzenmacher; Article No. 17; pp. 17:1–17:6
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A closely related problem is #KNAPSACK, which asks for the number of subsets $S$ such that $\sum_{s \in S} s \le t$. There are extensive studies on approximation algorithms for the #KNAPSACK problem [6, 8, 13, 7]. Our algorithm can solve the modulo $p$ version $\#_p$KNAPSACK in near-linear pseudopolynomial time for prime $p > t$.

Compared to the previous near-linear time algorithm for SUBSET SUM by Bringmann [4], our algorithm is simpler and more practical. The precise running time of our algorithm is $O(n + t \log^2 t)$ with error probability $O((n+t)^{-1})$. If a faster algorithm for manipulating formal power series by Brent [3] is applied, it can be improved to $O(n + t \log t)$ time (see Remark on Lemma 2), which is faster than Bringmann's algorithm by a factor of $\log^4 n$.

## 1.1 Main ideas of our algorithm

The SUBSET SUM instance can be encoded as a generating function $A(x) = \prod_{i=1}^{n}(1 + x^{s_i})$, where $s_1, \ldots, s_n$ are the input integers, and our goal is to compute the $t$-th coefficient of $A(x)$ and see whether it is zero or not.

Instead of directly expanding $A(x)$, we consider its logarithm $B(x) = \ln(A(x))$. Using basic properties of the logarithm function and its power series, it's possible to compute the first $t+1$ coefficients of $B(x)$ in $\tilde{O}(t)$ time. Then we can recover the first $t+1$ coefficients of $A(x) = \exp(B(x))$ in $\tilde{O}(t)$ time using a simple divide and conquer algorithm with FFT (or a slightly faster algorithm by Brent [3]).

The coefficients involved in the algorithm could be exponentially large. To avoid dealing with high-precision numbers, we pick a prime $p$ and perform arithmetic operations efficiently in the finite field $\mathbb{F}_p$, and in the end check whether the result is zero modulo $p$. By picking random $p$ from a large interval, the algorithm succeeds with high probability.

# 2 Preliminaries

## 2.1 Subset sum problem

Given $n$ (not necessarily distinct) positive integers $s_1, s_2, \ldots, s_n$ and a target sum $t$, the SUBSET SUM problem is to decide whether there exists a subset of indices $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} s_i = t$. We also consider the $\#_p$SUBSET SUM problem, which asks for the number of such subsets $I$ modulo $p$. We use the word RAM model with word length $w = \Theta(\log t)$ throughout this paper.

## 2.2 Polynomials and formal power series

### Formal power series

Let $R[x]$ denote the ring of polynomials over a ring $R$, and $R[[x]]$ denote the ring of formal power series over $R$. A formal power series $f(x) = \sum_{i=0}^{\infty} f_i x^i$ is a generalization of a polynomial with possibly an infinite number of terms. Polynomial addition and multiplication naturally generalize to $R[[x]]$. Composition $(f \circ g)(x) = f(g(x)) = \sum_{i=0}^{\infty} f_i \left( \sum_{j=1}^{\infty} g_j x^j \right)^i$ is well-defined for $f(x) = \sum_{i=0}^{\infty} f_i x^i \in R[[x]]$ and $g(x) = \sum_{j=1}^{\infty} g_j x^j \in xR[[x]]$. Here $xR[[x]]$ (or $xR[x]$) denotes the set of series in $R[[x]]$ (or polynomials in $R[x]$) with zero constant term.

### Exponential and logarithm

We are familiar with the following two series in $\mathbb{Q}[[x]]$,

$$\ln(1+x) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1}x^k}{k}, \tag{1}$$

$$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}, \tag{2}$$

satisfying

$$\exp\big(\ln(1+f(x))\big) = 1 + f(x), \tag{3}$$

and

$$\ln\big((1+f(x))(1+g(x))\big) = \ln(1+f(x)) + \ln(1+g(x)) \tag{4}$$

for any $f(x), g(x) \in x\mathbb{Q}[x]$.

### Modulo $x^{t+1}$

Our algorithm only deals with the first $t+1$ terms of any formal power series. For $f(x), g(x) \in R[[x]]$, we write $f(x) \equiv g(x) \pmod{x^{t+1}}$ if $[x^i]f(x) = [x^i]g(x)$ for all $0 \le i \le t$, where $[x^i]f(x)$ denotes the $i$-th coefficient of $f(x)$.

As an example, define

$$\exp_t(x) = \sum_{i=0}^{t} \frac{x^i}{i!} \tag{5}$$

as a $t$-th degree polynomial in $\mathbb{Q}[x]$. Then $\exp(f(x)) \equiv \exp_t(f(x)) \pmod{x^{t+1}}$ clearly holds for any $f(x) \in x\mathbb{Q}[[x]]$.

## 2.3 Modulo prime $p$

To avoid dealing with large fractions or floating-point numbers, we will work in the finite field $\mathbb{F}_p = \{\bar{0}, \bar{1}, \ldots, \overline{p-1}\}$ of prime order $p = 2^{\Theta(\log t)}$. Addition and multiplication in $\mathbb{F}_p$ take $O(1)$ time in the word RAM model. Finding the multiplicative inverse of a nonzero element in $\mathbb{F}_p$ takes $O(\log p)$ time using extended Euclidean algorithm [5, Section 31.2].

Our algorithm will regard polynomial coefficients as elements from $\mathbb{F}_p$. The coefficients can be rational numbers, but their denominators should not have prime factor $p$. Formally, let

$$\mathbb{Z}_{p\mathbb{Z}} = \{r/s \in \mathbb{Q} : r, s \text{ are coprime integers}, p \text{ does not divide } s\} \tag{6}$$

and apply the canonical homomorphism from $\mathbb{Z}_{p\mathbb{Z}}[x]$ to $\mathbb{F}_p[x]$, determined by

$$r/s \mapsto \bar{s}^{-1}\bar{r}, \quad x \mapsto x. \tag{7}$$

We use $\bar{A}$ or $A \bmod p$ to denote $A$'s image in $\mathbb{F}_p[x]$.

From now on we assume $p > t$, so that $\exp_t(x) \in \mathbb{Z}_{p\mathbb{Z}}[x]$ (see equation (5)), and let $\overline{\exp_t}(x)$ denote its image in $\mathbb{F}_p[x]$.

```
procedure COMPUTE(l, r)  ▷ after COMPUTE(l, r) returns, all values g_1, ..., g_r are ready
    if l < r then
        m ← ⌊(l + r)/2⌋
        COMPUTE(l, m)
        for i ← m + 1, m + 2, ..., r do
            g_i ← g_i + i^{-1} ∑_{j=l}^{m} (i − j) f_{i−j} g_j
        end for
        COMPUTE(m + 1, r)
    end if
end procedure

procedure MAIN
    Initialize g_0 ← 1, g_i ← 0 (1 ≤ i ≤ t)
    COMPUTE(0, t)
end procedure
```

**Figure 1** Algorithm for computing $g_1, \dots, g_t$.

## 2.4 Computing exponential using FFT

▶ **Lemma 1** (FFT). *Given two polynomials $f(x), g(x) \in \mathbb{F}_p[x]$ of degree at most $t$, one can compute their product $f(x)g(x)$ in $O(t \log t)$ time.*

**Proof.** The classic FFT algorithm [5, Chapter 30] can multiply $f(x)$ and $g(x)$, regarded as polynomials in $\mathbb{Z}[x]$, in $O(t \log t)$ time. Then take the remainder of each coefficient modulo $p$. ◀

Lemma 2 is a classical result on manipulating formal power series, and is the main building block of our algorithm.

▶ **Lemma 2** (Brent [3]). *Given a polynomial $f(x) \in x\mathbb{F}_p[x]$ of degree at most $t$ $(t < p)$, one can compute a polynomial $g(x) \in \mathbb{F}_p[x]$ in $\tilde{O}(t)$ time such that $g(x) \equiv \overline{\exp_t}(f(x)) \pmod{x^{t+1}}$.*

▶ Remark. Brent's algorithm [3] uses Newton's iterative method and runs in time $O(t \log t)$. Here we describe a simpler $O(t \log^2 t)$ algorithm by standard divide and conquer. We present the algorithm as over $\mathbb{Q}$ for notational simplicity.

**Proof.** Let $f(x) = \sum_{i=1}^{t} f_i x^i$ and $g(x) = \exp(f(x)) = \sum_{i=0}^{\infty} g_i x^i$. Then $g'(x) = g(x)f'(x)$. Comparing the $(i-1)$-th coefficients on both sides gives a recurrence relation

$$g_i = i^{-1} \sum_{j=0}^{i-1} (i - j) f_{i-j} g_j \tag{8}$$

with initial value $g_0 = 1$. The desired coefficients $g_1, \dots, g_t$ can be computed using the algorithm in Figure 1, which simply reorganizes the computation of recurrence formula (8) as a recursion.

To speed up this algorithm, define polynomial $F(x) = \sum_{k=0}^{r-l} k f_k x^k, G(x) = \sum_{j=0}^{m-l} g_{j+l} x^j$ and use FFT to compute $H(x) = F(x)G(x)$ in $O((r-l)\log(r-l))$ time after COMPUTE$(l, m)$ returns. Then $\sum_{j=l}^{m}(i-j)f_{i-j}g_j = [x^{i-l}]H(x)$, and hence the **for** loop runs in $O(r-m)$ time. The total running time is $T(t) = 2T(t/2) + O(t \log t) = O(t \log^2 t)$. ◀

## 3 Main algorithm

Recall that we are given $n$ positive integers $s_1, \ldots, s_n$ and a target sum $t$. Consider the generating function $A(x)$ defined by

$$A(x) = \prod_{i=1}^{n} (1 + x^{s_i}). \tag{9}$$

The number of subsets that sum up to $t$ is $[x^t]A(x)$. The Subset Sum instance has a solution if and only if $[x^t]A(x) \neq 0$.

▶ **Lemma 3.** *Suppose $[x^t]A(x) \neq 0$. Let $p$ be a uniform random prime from $[t+1, (n+t)^3]$. With probability $1 - O((n+t)^{-1})$, $p$ does not divide $[x^t]A(x)$.*

**Proof.** Notice that $[x^t]A(x) \leq 2^n$, so it has at most $n$ prime factors. Since there are $\Omega((n+t)^2)$ primes in the interval, the probability that $p$ divides $[x^t]A(x)$ is $O((n+t)^{-1})$. ◀

▶ **Lemma 4.** *Let $B(x) = \ln(A(x)) \in \mathbb{Q}[[x]]$. For prime $p > t$, in $\tilde{O}(t)$ time one can compute $([x^r]B(x)) \bmod p$ for all $0 \leq r \leq t$.*

**Proof.** By definition of $B(x)$,

$$B(x) = \ln\left(\prod_{i=1}^{n}(1 + x^{s_i})\right) = \sum_{i=1}^{n} \ln(1 + x^{s_i}) = \sum_{i=1}^{n}\sum_{j=1}^{\infty} \frac{(-1)^{j-1}}{j} x^{s_i j}. \tag{10}$$

Let $a_k$ be the size of the set $\{j : s_j = k\}$, and define polynomial

$$B_t(x) = \sum_{i=1}^{n}\sum_{j=1}^{\lfloor t/s_i \rfloor} \frac{(-1)^{j-1}}{j} x^{s_i j} = \sum_{k=1}^{t}\sum_{j=1}^{\lfloor t/k \rfloor} \frac{a_k(-1)^{j-1}}{j} x^{jk}. \tag{11}$$

Then $[x^r]B_t(x) = [x^r]B(x)$ for all $0 \leq r \leq t$.

Note that the denominators $j$ in (11) do not have prime factor $p$. After preparing the multiplicative inverses $\bar{j}^{-1}$ for each $1 \leq j \leq t$, we can compute all $([x^r]B_t(x)) \bmod p$ by simply iterating over $k, j$ in equation (11), which only takes $\sum_{k=1}^{t} \lfloor t/k \rfloor = O(t \log t)$ time. ◀

▶ **Lemma 5.** *For prime $p > t$, one can compute $([x^r]A(x)) \bmod p$ for all $0 \leq r \leq t$ in $\tilde{O}(t)$ time.*

**Proof.** Let $B(x) = \ln(A(x))$. Then $A(x) = \exp(B(x)) \equiv \exp_t(B_t(x)) \pmod{x^{t+1}}$, where $B_t(x) = \sum_{i=0}^{t}([x^i]B(x))x^i$. We use Lemma 4 to compute $B_t(x)$'s image $\overline{B_t}(x) \in \mathbb{F}_p[x]$, and then use Lemma 2 to compute the first $t+1$ terms of $\overline{\exp_t}(\overline{B_t}(x))$, which give the values of $([x^r]A(x)) \bmod p$ for all $0 \leq r \leq t$. ◀

▶ **Theorem 6.** *The Subset Sum problem can be solved in time $\tilde{O}(n+t)$ by a randomized algorithm with one-sided error probability $O((n+t)^{-1})$.*

**Proof.** By sampling and using Miller-Rabin primality test [5, Section 31.8], we can pick a uniform random prime $p$ from interval $[t+1, (n+t)^3]$ in $(\log(n+t))^{O(1)}$ time with $O((n+t)^{-1})$ failure probability. Then the theorem immediately follows from Lemma 3 and Lemma 5. ◀

## References


1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019. To appear. URL: http://arxiv.org/abs/1704.04546.

2 Richard E. Bellman. *Dynamic programming*. Princeton University Press, 1957.

3 Richard P. Brent. Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In *Analytic Computational Complexity*, pages 151–176. Elsevier, 1976. doi:10.1016/B978-0-12-697560-4.50014-9.

4 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1073–1084, 2017. doi:10.1137/1.9781611974782.69.

5 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 3rd edition, 2009.

6 Martin Dyer. Approximate counting by dynamic programming. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 693–699, 2003. doi:10.1145/780542.780643.

7 Paweł Gawrychowski, Liran Markin, and Oren Weimann. A Faster FPTAS for #Knapsack. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 64:1–64:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.64.

8 Parikshit Gopalan, Adam Klivans, Raghu Meka, Daniel Štefankovic, Santosh Vempala, and Eric Vigoda. An FPTAS for #knapsack and related counting problems. In *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 817–826, 2011. doi:10.1109/FOCS.2011.32.

9 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer US, 1972. doi:10.1007/978-1-4684-2001-2_9.

10 Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for subset sum. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1062–1072, 2017. doi:10.1137/1.9781611974782.68.

11 Konstantinos Koiliaris and Chao Xu. Subset Sum Made Simple. *CoRR*, abs/1807.08248, 2018. URL: http://arxiv.org/abs/1807.08248.

12 David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33(1):1–14, 1999. doi:10.1006/jagm.1999.1034.

13 Romeo Rizzi and Alexandru I. Tomescu. Faster FPTASes for counting and random generation of knapsack solutions. In *European Symposium on Algorithms (ESA)*, pages 762–773, 2014. doi:10.1007/978-3-662-44777-2_63.