


# Subset Sum Made Simple

**Konstantinos Koiliaris**

Department of Computer Science, University of Illinois Urbana-Champaign  
201 North Goodwin Avenue, Urbana, IL 61801, USA  
koiliar2@illinois.edu

 <https://orcid.org/0000-0002-1842-1829>


**Chao Xu**

*Current affiliation:* Yahoo! Research

770 Broadway, 6th Floor, New York, NY 10003, USA

*Previous affiliation:* Department of Computer Science, University of Illinois Urbana-Champaign  
201 North Goodwin Avenue, Urbana, IL 61801, USA

chao.xu@oath.com

 <https://orcid.org/0000-0003-4417-3299>

---

## Abstract

SUBSETSUM is a classical optimization problem taught to undergraduates as an example of an NP-hard problem, which is amenable to dynamic programming, yielding polynomial running time if the input numbers are relatively small. Formally, given a set  $S$  of  $n$  positive integers and a target integer  $t$ , the SUBSETSUM problem is to decide if there is a subset of  $S$  that sums up to  $t$ . Dynamic programming yields an algorithm with running time  $O(nt)$ . Recently, the authors [17] improved the running time to  $\tilde{O}(\sqrt{nt})$ , and it was further improved to  $\tilde{O}(n+t)$  by a somewhat involved randomized algorithm by Bringmann [5], where  $\tilde{O}$  hides polylogarithmic factors.

Here, we present a new and significantly simpler algorithm with running time  $\tilde{O}(\sqrt{nt})$ . While not the fastest, we believe the new algorithm and analysis are simple enough to be presented in an algorithms class, as a striking example of a divide-and-conquer algorithm that uses FFT to a problem that seems (at first) unrelated. In particular, the algorithm and its analysis can be described in full detail in two pages (see pages 3–5).

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

**Keywords and phrases** subset sum, pseudopolynomial time, divide-and-conquer, FFT

**Acknowledgements** We would like to thank Sarel Har-Peled for his invaluable help in the editing of this paper.

## 1 Introduction

Given a (multi) set  $S$  of  $n$  positive integers and an integer target value  $t$ , the SUBSETSUM problem is to decide if there is a (multi) subset of  $S$  that sums up to  $t$ . The SUBSETSUM is a classical problem with relatively long history. It is one of Karp’s original NP-complete problems [14], closely related to other fundamental NP-complete problems such as KNAPSACK [7], CONSTRAINED SHORTEST PATH [2], and various other graph problems with cardinality constraints [9, 12, 16]. Furthermore, it is one of the initial *weakly* NP-complete problems; problems that admit *pseudopolynomial* time algorithms – a classification identified by Garey and Johnson in [11]. The first such algorithm was given in 1957\* by Bellman, who showed how to solve the problem in  $O(nt)$  time using dynamic programming [3].

---

\* Note that Bellman wrote this paper before the definition of pseudopolynomial time algorithms was provided by Garey and Johnson in 1977.

The importance of the SUBSETSUM problem in computer science is further highlighted by its role in teaching. Both the problem and its algorithm have been included in undergraduate algorithms courses' curriculums and textbooks for several decades ([6, Chapter 34.5.5], used as archetypal examples for introducing the notions of weak NP-completeness and pseudopolynomial time algorithms to college students [15, Chapter 8.8]. In addition, the conceptually simple problem statement makes this problem a great candidate in the study of NP-completeness [8, Chapter 8.1]), and, finally, Bellman's algorithm is also often introduced in the context of teaching dynamic programming [10, Chapter 5.6].

Extensive work has been done on finding better and faster pseudopolynomial time algorithms for the SUBSETSUM (for a collection of previous results see [17, Table 1.1]). The first improvement on the running time was a  $O(nt/\log t)$  time algorithm by [18], almost two decades go. Recently, the *state-of-the-art* was improved significantly to  $\tilde{O}(\sqrt{nt})$  time by the authors [17]. Shortly after, in a follow up work, the running time was further improved to  $\tilde{O}(n+t)$  time by Bringmann [5] – the algorithm is randomized and somewhat involved. Abboud et al. [1] showed that it is unlikely that any SUBSETSUM algorithm runs in time  $O(t^{1-\varepsilon} 2^{o(n)})$ , for any constant  $\varepsilon > 0$  and target number  $t$ , as such an algorithm would imply that the Strong Exponential Time Hypothesis (SETH) of Impagliazzo and Paturi [13] is false.

In this paper, we present a new *simple* algorithm for the SUBSETSUM problem. The algorithm follows the divide-and-conquer paradigm and uses the Fast Fourier Transform (FFT), matching the best deterministic running time  $\tilde{O}(\sqrt{nt})$  of [17] with a cleaner and more straightforward analysis. The algorithm partitions the input by congruence into classes, computes the subset sums of each class recursively, and combines the results. We believe this new simple algorithm, although not improving upon the state-of-the-art, reduces the conceptual complexity of the problem and improves our understanding of it. We believe the new algorithm can be used in teaching as an example of a pseudopolynomial time algorithm for the SUBSETSUM problem, as well as a striking example of applying FFT to a seemingly unrelated problem.

### Comparison to previous work

Our previous algorithm [17] used a more complicated divide-and-conquer strategy that resulted in forming sets of two different types, that had to be handled separately. Bringmann's algorithm [5] uses randomization and a two-stage color-coding process. Both algorithms are significantly more complicated than the one presented here.

## 2 Preliminaries

Let  $[u] = \{0, 1, \dots, \lceil u \rceil\}$  denote the set of integers in the interval  $[0, \lceil u \rceil]$ . Given a set  $X \subset \mathbb{N}$ , let  $\Sigma X = \sum_{x \in X} x$  and denote the *set of all subset sums of  $X$  up to  $u$*  by

$$\mathcal{S}_u(X) = \{ \Sigma Y \mid Y \subseteq X \} \cap [u],$$

and the *set of all subset sums of  $X$  up to  $u$  with cardinality information* by

$$\mathcal{S}_u^\#(X) = \{ (\Sigma Y, |Y|) \mid Y \subseteq X \} \cap ([u] \times \mathbb{N}).$$

Let  $X, Y$  be two sets, the *set of pairwise sums of  $X$  and  $Y$  up to  $u$*  is denoted by

$$X \oplus_u Y = \{ x + y \mid x \in X, y \in Y \} \cap [u].$$

If  $X, Y \subseteq \mathbb{N} \times \mathbb{N}$  are sets of points in the plane, then

$$X \oplus_u Y = \{(x_1 + y_1, x_2 + y_2) \mid (x_1, x_2) \in X, (y_1, y_2) \in Y\} \cap ([u] \times \mathbb{N}).$$

Observe, that if  $X$  and  $Y$  are two disjoint sets, then  $\mathcal{S}_u(X \cup Y) = \mathcal{S}_u(X) \oplus_u \mathcal{S}_u(Y)$ .

Next, we define two generalizations of the SUBSETSUM problem. Both can be solved by the new algorithm.

<b>ALLSUBSETSUMS</b> <b>INPUT:</b> Given a set $S$ of $n$ positive integers and an upper bound integer $u$ . <b>OUTPUT:</b> The set of <i>all realizable subset sums</i> of $S$ up to $u$ .	<b>ALLSUBSETSUMS<sup>#</sup></b> <b>INPUT:</b> Given a set $S$ of $n$ positive integers and an upper bound integer $u$ . <b>OUTPUT:</b> The set of all realizable subset sums along with <i>the size of the subset that realizes each sum</i> of $S$ up to $u$ .
---	--

■ **Figure 1** Two generalizations of the SUBSETSUM problem.

Note that the case where the input is a multiset can be reduced to the case of a set with little loss in generality and running time (see [17, Section 2.2]), hence for simplicity of exposition we assume the input is a *set* throughout the paper.

### 3 The algorithm

Here, we show how to solve ALLSUBSETSUMS in  $\tilde{O}(\sqrt{nt})$  time. Clearly, computing all subset sums up to  $u$  also decides SUBSETSUM with target value  $t \leq u$ .

#### 3.1 Building blocks

The following well-known lemma describes how to compute pairwise sums between sets in almost linear time, in the size of their ranges, using FFT.

► **Lemma 1** (Computing pairwise sums  $\oplus_u$ ). *The following are true:*

- (A) *Given two sets  $S, T \subseteq [u]$ , one can compute  $S \oplus_u T$  in  $O(u \log u)$  time.*
- (B) *Given  $k$  sets  $S_1, \dots, S_k \subseteq [u]$ , one can compute  $S_1 \oplus_u \dots \oplus_u S_k$  in  $O(k u \log u)$  time.*
- (C) *Given two sets of points  $S, T \subseteq [u] \times [v]$ , one can compute  $S \oplus_u T$  in  $O(u v \log(u v))$  time.*

**Proof.** (A) Let  $f_S = f_S(x) = \sum_{i \in S} x^i$  be the characteristic polynomial of  $S$ . Construct, in a similar fashion, the polynomial  $f_T$  (for the set  $T$ ) and let  $g = f_S * f_T$ . Observe that for  $i \leq u$ , the coefficient of  $x^i$  in  $g$  is nonzero if and only if  $i \in S \oplus_u T$ . Using FFT, one can compute the polynomial  $g$  in  $O(u \log u)$  time, and extract  $S \oplus_u T$  from it.

(B) Let  $Z_1 = S_1$ , and let  $Z_i = Z_{i-1} \oplus_u S_i$ , for  $i \in [2, k]$ . Compute each  $Z_i$ , from  $Z_{i-1}$  and  $S_i$ , in  $O(u \log u)$  time using part (A). The total running time is  $O(k u \log u)$ .

(C) As in (A), let  $f_S = f_S(x, y) = \sum_{(i,j) \in S} x^i y^j$  and  $f_T$  be the characteristic polynomials of  $S$  and  $T$ , respectively, and let  $g = f_S * f_T$ . For  $i \leq u$  the coefficient of  $x^i y^j$  is nonzero if and only if  $(i, j) \in S \oplus_u T$ . One can compute the polynomial  $g$  by a straightforward reduction to regular FFT (see multidimensional FFT [4, Chapter 12.8]), in  $O(u v \log(u v))$  time, and extract  $S \oplus T$  from it. ◀

The next lemma shows how to answer ALLSUBSETSUMS<sup>#</sup> quickly, originally shown by the authors in [17], the proof is included for completeness.

► **Lemma 2** (ALLSUBSETSUMS<sup>#</sup> [17]). *Let  $S \subseteq [u]$  be a given set of  $n$  elements. One can compute, in  $O(un \log n \log u)$  time, the set  $\mathcal{S}_u^\#(S)$ , which includes all subset sums of  $S$  up to  $u$  with cardinality information.*

**Proof.** Partition  $S$  into two sets  $S_1$  and  $S_2$  of roughly the same size. Compute  $\mathcal{S}_u^\#(S_1)$  and  $\mathcal{S}_u^\#(S_2)$  recursively, and observe that  $\mathcal{S}_u^\#(S_1), \mathcal{S}_u^\#(S_2) \subseteq ([u] \times [\frac{n}{2}])$ . Finally, note that  $\mathcal{S}_u^\#(S_1) \oplus_u \mathcal{S}_u^\#(S_2) = \mathcal{S}_u^\#(S)$ . Applying Lemma 1.C yields  $\mathcal{S}_u^\#(S)$ .

The running time follows the recursive formula  $T(n) = 2 \cdot T(n/2) + O(un \log u)$ , which is  $O(un \log u \log n)$ , proving the claim. ◀

Next, we show how to compute the subset sums of elements in a congruence class quickly.

► **Lemma 3.** *Let  $\ell, b \in \mathbb{N}$  with  $\ell < b$ . Given a set  $S \subseteq \{x \in \mathbb{N} \mid x \equiv \ell \pmod{b}\}$  of size  $n$ , one can compute  $\mathcal{S}_u(S)$  in  $O((u/b)n \log n \log u)$  time.*

**Proof.** An element  $x \in S$  can be written as  $x = yb + \ell$ . Let  $Q = \{y \mid yb + \ell \in S\}$ . As such, for any subset  $X = \{y_1b + \ell, \dots, y_jb + \ell\} \subseteq S$  of size  $j$ , we have that

$$\sum_{x \in X} x = \sum_{i=1}^j (y_i b + \ell) = \left( \sum_{i=1}^j y_i \right) b + j\ell$$

In particular, a pair  $(z, j) \in \mathcal{S}_{u/b}^\#(Q)$  corresponds to a set  $Y = \{y_1, \dots, y_j\} \subseteq Q$  of size  $j$ , such that  $\sum_i y_i = z$ . The set  $Y$  in turn corresponds to the set  $X = \{y_1b + \ell, \dots, y_jb + \ell\} \subseteq S$ . By the above, the sum of the elements of  $X$  is  $zb + j\ell$ . As such, compute  $\mathcal{S}_{u/b}^\#(Q)$ , using the algorithm of Lemma 2, and return  $\{zb + j\ell \mid (z, j) \in \mathcal{S}_{u/b}^\#(Q)\} = \mathcal{S}_u(S)$  as the desired result. ◀

### 3.2 Algorithm

The new algorithm partitions the input into sets by congruence. Next it computes the ALLSUBSETSUMS<sup>#</sup> for each such set, and combines the results. The algorithm is depicted in Figures 2 and 3.

ALLSUBSETSUMS<sup>#</sup>( $S, u$ ):

INPUT: A set  $S$  of  $n$  positive integers and an upper bound integer  $u$ .

OUTPUT: The set of all subset sums with cardinality information of  $S$  up to  $u$ .

1. if  $S = \{x\}$
2.     **return**  $\{(0, 0), (x, 1)\}$
3.  $T \leftarrow$  an arbitrary subset of  $S$  of size  $\lfloor n/2 \rfloor$
4. **return** ALLSUBSETSUMS<sup>#</sup>( $T, u$ )  $\oplus_u$  ALLSUBSETSUMS<sup>#</sup>( $S \setminus T, u$ )

■ **Figure 2** The algorithm for the ALLSUBSETSUMS<sup>#</sup> problem, used as a subroutine in Figure 3.

### 3.3 Result

► **Theorem 4** (ALLSUBSETSUMS). *Let  $S \subseteq [u]$  be a given set of  $n$  elements. One can compute, in  $O(\sqrt{n \log n} u \log u)$  time, the set  $\mathcal{S}_u(S)$ , which contains all subset sums of  $S$  up to  $u$ .*

ALLSUBSETSUMS( $S, u$ ):

INPUT: A set  $S$  of  $n$  positive integers and an upper bound integer  $u$ .

OUTPUT: The set of all realizable subset sums of  $S$  up to  $u$ .

```

1.  $b \leftarrow \lfloor \sqrt{n \log n} \rfloor$ 
2. for  $\ell \in [b-1]$  do
3.    $S_\ell \leftarrow S \cap \{x \in \mathbb{N} \mid x \equiv \ell \pmod{b}\}$ 
4.    $Q_\ell \leftarrow \{(x - \ell)/b \mid x \in S_\ell\}$ 
5.    $\mathcal{S}_{u/b}^\#(Q_\ell) \leftarrow \text{ALLSUBSETSUMS}^\#(Q_\ell, \lfloor u/b \rfloor)$ 
6.    $R_\ell \leftarrow \{zb + \ell j \mid (z, j) \in \mathcal{S}_{u/b}^\#(Q_\ell)\}$ 
7. return  $R_0 \oplus_u \cdots \oplus_u R_{b-1}$ 

```

■ **Figure 3** The algorithm for ALLSUBSETSUMS.

**Proof.** Partition  $S$  into  $b = \lfloor \sqrt{n \log n} \rfloor$  sets  $S_\ell = S \cap \{x \in \mathbb{N} \mid x \equiv \ell \pmod{b}\}$ ,  $\ell \in [b-1]$ , each of  $n_\ell$  elements. For each  $S_\ell$ , compute the set of all subset sums  $\mathcal{S}_u(S_\ell)$  in  $O((u/b)n_\ell \log n_\ell \log u)$  time by Lemma 3. The time spent to compute all  $\mathcal{S}_u(S_\ell)$  is  $\sum_{\ell \in [b-1]} O((u/b)n_\ell \log n_\ell \log u) = O((u/b)n \log n \log u)$ . Combining  $\mathcal{S}_u(S_0) \oplus_u \cdots \oplus_u \mathcal{S}_u(S_{b-1})$  using Lemma 1.B takes  $O(bu \log u)$  time. Hence, the total running time is  $O((u/\lfloor \sqrt{n \log n} \rfloor)n \log n \log u + \lfloor \sqrt{n \log n} \rfloor u \log u) = O(\sqrt{n \log n} u \log u)$ . ◀

► **Remark.** ALLSUBSETSUMS is a generalization of SUBSETSUM, so the algorithm of Theorem 4 applies to it.

## References

- 1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. *CoRR*, abs/1704.04546, 2017. URL: <http://arxiv.org/abs/1704.04546>, arXiv:1704.04546.
- 2 Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster knapsack and graph algorithms. *CoRR*, abs/1802.06440, 2018. URL: <http://arxiv.org/abs/1802.06440>, arXiv:1802.06440.
- 3 Richard Bellman. Notes on the theory of dynamic programming IV - maximization over discrete sets. *Naval Research Logistics Quarterly*, 3(1-2):67–70, 1956. URL: <http://dx.doi.org/10.1002/nav.3800030107>, doi:10.1002/nav.3800030107.
- 4 Richard E. Blahut. *Fast Algorithms for Digital Signal Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1985.
- 5 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1073–1084. Society for Industrial and Applied Mathematics, 2017.
- 6 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
- 7 George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–277, 1957. URL: <http://www.jstor.org/stable/167356>.
- 8 Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., New York, NY, USA, first edition, 2008.
- 9 David Eppstein. Minimum range balanced cuts via dynamic subset sums. *Journal of Algorithms*, 23(2):375 – 385, 1997. URL: <http://www.sciencedirect.com/science/article/pii/S019667749690841X>, doi:http://dx.doi.org/10.1006/jagm.1996.0841.

- 10 Jeff Erickson. Algorithms, etc., January 2015. Course materials, 1250 pages. URL: <https://jeffe.cs.illinois.edu/teaching/algorithms/>.
- 11 Michael R Garey and David S Johnson. “Strong” NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.
- 12 Venkatesan Guruswami, Yury Makarychev, Prasad Raghavendra, David Steurer, and Yuan Zhou. Finding almost-perfect graph bisections. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 321–337, 2011. URL: <http://conference.itcs.tsinghua.edu.cn/ICS2011/content/papers/11.html>.
- 13 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367 – 375, 2001. URL: <http://www.sciencedirect.com/science/article/pii/S0022000000917276>, doi:<https://doi.org/10.1006/jcss.2000.1727>.
- 14 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972. URL: [http://dx.doi.org/10.1007/978-1-4684-2001-2\\_9](http://dx.doi.org/10.1007/978-1-4684-2001-2_9), doi:10.1007/978-1-4684-2001-2\_9.
- 15 Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.
- 16 Bettina Klinz and Gerhard J. Woeginger. A note on the bottleneck graph partition problem. *Networks*, 33(3):189–191, 1999. URL: [http://dx.doi.org/10.1002/\(SICI\)1097-0037\(199905\)33:3<189::AID-NET5>3.0.CO;2-2](http://dx.doi.org/10.1002/(SICI)1097-0037(199905)33:3<189::AID-NET5>3.0.CO;2-2), doi:10.1002/(SICI)1097-0037(199905)33:3<189::AID-NET5>3.0.CO;2-2.
- 17 Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1062–1072. SIAM, 2017.
- 18 David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33(1):1 – 14, 1999.