An Elegant Argument that $P \neq NP$

Craig Alan Feinstein

2712 Willow Glen Drive, Baltimore, Maryland 21209. E-mail: cafeinst@msn.com

In this note, we present an elegant argument that $P \neq NP$ by demonstrating that the Meet-in-the-Middle algorithm must have the fastest running-time of all deterministic and exact algorithms which solve the SUBSET-SUM problem on a classical computer.

Consider the following problem: Let $A = \{a_1, \ldots, a_n\}$ be a set of n integers and b be another integer. We want to find a subset of A for which the sum of its elements (we shall call this quantity a *subset-sum*) is equal to b (we shall call this quantity the *target integer*). We shall consider the sum of the elements of the empty set to be zero. This problem is called the *SUBSET-SUM problem* [1,2]. Now consider the following algorithm for solving the SUBSET-SUM problem:

Meet-in-the-Middle Algorithm - First, partition the set A into two subsets, $A^+=\{a_1, \ldots, a_{\lceil \frac{n}{2} \rceil}\}$ and $A^-=\{a_{\lceil \frac{n}{2} \rceil+1}, \ldots, a_n\}$. Let us define S^+ and S^- as the sets of subset-sums of A^+ and A^-, respectively. Sort sets S^+ and $b - S^-$ in ascending order. Compare the first elements in both of the lists. If they match, then output the corresponding solution and stop. If not, then compare the greater element with the next element in the other list. Continue this process until there is a match, in which case there is a solution, or until one of the lists runs out of elements, in which case there is no solution.

This algorithm takes $\Theta(\sqrt{2^n})$ time, since it takes $\Theta(\sqrt{2^n})$ steps to sort sets S^+ and $b - S^-$ and $O(\sqrt{2^n})$ steps to compare elements from the sorted lists S^+ and $b - S^-$. It turns out that no deterministic and exact algorithm with a better worst-case running-time has ever been found since Horowitz and Sahni discovered this algorithm in 1974 [3, 4]. In this paper, we shall prove that it is impossible for such an algorithm to exist.

First of all, we shall assume, without loss of generality, that the code of any algorithm considered in our proof does not contain full or partial solutions to any instances of SUBSET-SUM. This is because only finitely many such solutions could be written in the code, so such a strategy would not be helpful in speeding up the running-time of an algorithm solving SUBSET-SUM when n is large. We now give a definition:

Definition 1: *We define a γ-comparison (a generalized comparison) between two integers, x and y, as any finite procedure that directly or indirectly determines whether or not $x = y$.*

For example, a finite procedure that directly compares $f(x)$ and $f(y)$, where x and y are integers and f is a one-to-one function, γ-compares the two integers x and y, since $x = y$ if and only if $f(x) = f(y)$. Using this expanded definition of compare, it is not difficult to see that the Meet-in-the-Middle algorithm γ-compares subset-sums with the target integer until it finds a subset-sum that matches the target integer. We shall now prove two lemmas:

Lemma 2: *Let x and y be integers. If $x = y$, then the only type of finite procedure that is guaranteed to determine that $x = y$ is a γ-comparison between x and y. And if $x \neq y$, then the only type of finite procedure that is guaranteed to determine that $x \neq y$ is a γ-comparison between x and y.*

Proof: Suppose there is a finite procedure that is guaranteed to determine that $x = y$, if $x = y$. Then if the procedure does not determine that $x = y$, this implies that $x \neq y$. Hence, the procedure always directly or indirectly determines whether or not $x = y$, so the procedure is a γ-comparison between x and y.

And suppose there is a finite procedure that is guaranteed to determine that $x \neq y$, if $x \neq y$. Then if the procedure does not determine that $x \neq y$, this implies that $x = y$. Hence, the procedure always directly or indirectly determines whether or not $x = y$, so the procedure is a γ-comparison between x and y. ■

Lemma 3: *Any deterministic and exact algorithm that finds a solution to SUBSET-SUM whenever a solution exists must (whenever a solution exists) γ-compare the subset-sum of the solution that it outputs with the target integer.*

Proof: Let Q be a deterministic and exact algorithm that finds a solution, $\{a_{k_1}, \ldots, a_{k_m}\}$, to SUBSET-SUM whenever a solution exists. Before Q outputs this solution, it must verify that $a_{k_1} + \ldots + a_{k_m} = b$, since we are assuming that the code of Q does not contain full or partial solutions to any instances of SUBSET-SUM. (See above for an explanation.) In order for Q to verify that $a_{k_1} + \ldots + a_{k_m} = b$, Q must γ-compare the subset-sum, $a_{k_1} + \ldots + a_{k_m}$, with the target integer, b, since a γ-comparison between $a_{k_1} + \ldots + a_{k_m}$ and b is the only type of finite procedure that is guaranteed to determine that $a_{k_1} + \ldots + a_{k_m} = b$, by Lemma 2. ■

As we see above, the Meet-in-the-Middle algorithm makes use of sorted lists of subset-sums in order to obtain a faster worst-case running-time, $\Theta(\sqrt{2^n})$, than that of a naïve brute-force search of the set of all subset-sums, $\Theta(2^n)$. The following lemma shows that sorted lists of subset-sums are necessary to achieve such an improved worst-case running-time.

Lemma 4: *If a deterministic and exact algorithm that finds a solution to SUBSET-SUM whenever a solution exists does not make use of sorted lists of subset-sums, it must run in $\Omega(2^n)$ time in the worst-case scenario.*

Proof: Let Q be a deterministic and exact algorithm that finds a solution to SUBSET-SUM whenever a solution exists without making use of sorted lists of subset-sums. By Lemma 3, Q must (whenever a solution exists) γ-compare the subset-sum of the solution that it outputs with the target integer. In order for Q to avoid wasting time γ-comparing the target integer with subset-sums that do not match the target integer, Q must be able to rule out possible matches between subset-sums and the target integer without γ-comparing these subset-sums with the target integer.

But by Lemma 2, the only type of finite procedure that is guaranteed to rule out a possible match between a subset-sum and the target integer, if they do not match, is a γ-comparison. So in the worst-case scenario, there is no way for Q to avoid wasting time γ-comparing the target integer with subset-sums that do not match the target integer. And since Q does not make use of sorted lists of subset-sums like the Meet-in-the-Middle algorithm does, its γ-comparisons between these subset-sums and the target integer will not rule out possible matches between any other subset-sums and the target integer. Hence, in the worst-case scenario Q must γ-compare each of the 2^n subset-sums with the target integer, which takes $\Omega(2^n)$ time. ■

It is now possible, using Lemma 4, to solve the problem of finding a nontrivial lower-bound for the worst-case running-time of a deterministic and exact algorithm that solves the SUBSET-SUM problem:

Theorem 5: *It is impossible for a deterministic and exact algorithm that solves the SUBSET-SUM problem to have a worst-case running-time of $o(\sqrt{2^n})$.*

Proof: Let T be the worst-case running-time of an algorithm Q that solves SUBSET-SUM, and let M be the size of the largest list of subset-sums that Q sorts. Since it is necessary for Q to make use of sorted lists of subset-sums in order to have a faster worst-case running-time than $\Theta(2^n)$ by Lemma 4 and since it is possible for Q to make use of sorted lists of size M of subset-sums to rule out at most M possible matches of subset-sums with the target integer at a time (just as the Meet-in-the-Middle algorithm does, with $M = \Theta(\sqrt{2^n})$), we have $MT \geq \Theta(2^n)$. And since creating a sorted list of size M must take at least M units of time, we have $T \geq M \geq 1$.

Then in order to find a nontrivial lower-bound for the worst-case running-time of an algorithm solving SUBSET-SUM, let us minimize T subject to $MT \geq \Theta(2^n)$ and $T \geq M \geq 1$. Because the running-time of $T = \Theta(\sqrt{2^n})$ is the solution to this optimization problem and because the Meet-in-the-Middle algorithm achieves this running-time, we can conclude that $\Theta(\sqrt{2^n})$ is a tight lower-bound for the worst-case running-time of any deterministic and exact algorithm which solves SUBSET-SUM. And this conclusion implies that $P \neq NP$ [1, 5]. ■

Acknowledgments

I thank God, my parents, my wife, and my children for their support.

Submitted on January 11, 2011 / Accepted on January 15, 2011

References

1. Cormen T. H., Leiserson C. E., Rivest R. L. Introduction to Algorithms, McGraw-Hill, 1990.
2. Menezes A., van Oorschot P., Vanstone S. Handbook of Applied Cryptography, CRC Press, 1996.
3. Horowitz E., Sahni S. Computing Partitions with Applications to the Knapsack Problem. *Journal of the ACM*, v. 2l, no. 2, April 1974, pp. 277–292.
4. Woeginger G. J. Exact Algorithms for NP-Hard Problems, *Lecture Notes in Computer Science*, 2003, v. 2570, pp. 185–207.
5. Bovet P. B., Crescenzi P. Introduction to the Theory of Complexity, Prentice Hall, 1994.