P ≠ NP using sorting keys, a proof by logical contradiction

Author

Robert DiGregorio
0x51B4908DdCD986A41e2f8522BB5B68E563A358De

Abstract

Using a new tool called a "sorting key" it's possible to imply P ≠ NP using a proof by logical contradiction.

Part 1

- Let PS(x) be the unsorted power list (list of all subsets) of unsorted list of naturals x, with each subset folded over the sum operation, such that, given some natural n, PS(x)[n] is the nth element of PS(x)
    - To clarify what "folded over the sum operation" means, here is the set {1, 2, 3} folded over the sum operation in pseudocode: "{1, 2, 3}.fold(sum) = 1 + 2 + 3 = 6"
    - To clarify, PS(x) is the unsorted list of all subset sums of x
    - To clarify, "sorted" means smaller naturals are always before larger naturals
- Let a "valid sorting key" be a natural such that, for some list x, for all natural n, PS(x)[n ⊕(the valid sorting key of PS(x))] is (sort PS(x))[n]
    - Calculating the valid sorting key that sorts for all elements of PS(x) is identical to sorting PS(x). This is because PS(x)[n] is the nth element of PS(x), unsorted, and PS(x)[n ⊕(the valid sorting key of PS(x))] is the nth element of PS(x), sorted, so having the valid sorting key that sorts for all elements of PS(x) means you have a sorted PS(x)
    - ⊕is the bitwise exclusive or operation. If you apply ⊕against some natural x to every natural from 0 (inclusive) to 2^(n) (exclusive), those naturals are reordered such that every unique x gives a unique order. As such, every power list has at least 1 "sorting key" that sorts it
    - If KEY is the sorting key of some list x, reordering x causes KEY to become "invalid" and no longer sort x
    - If all elements of PS(x) are unique, there is only 1 valid sorting key for PS(x). Again, 1 valid sorting key sorts all elements of PS(x)
- Let A be an unsorted list of naturals, given as input
- Let KEY be a natural, given as input

- Let the decision problem be "Given unsorted list A as input and natural KEY as input, is KEY not the valid sorting key of PS(A)?"
- A deterministic polynomial time verifier can verify a YES solution to the decision problem if list A, natural KEY, natural x, and natural y are given, such that $(x < y) \neq (PS(A)[x \oplus KEY] < PS(A)[y \oplus KEY])$
- If a deterministic polynomial time verifier exists for a YES solution to a decision problem such that all deterministic Turing machines calculate it must run in superpolynomial time, P ≠ NP
    - If the decision problem can't be solved in polynomial time, P ≠ NP
    - If the decision problem can be solved in polynomial time, see part 2

Part 2

- It's implied that ALGORITHM exists such that ALGORITHM can determine if a sorting key is invalid or not in polynomial time
    - If ALGORITHM is polynomial time for a YES solution to a decision problem, ALGORITHM polynomial time for a NO solution to a decision problem, and vice versa
- Let M be some deterministic Turing machine such that M, decides "given unsorted list A as input, does an even sorting key for PS(A) exist?"
    - Any such deterministic Turing machine runs in superpolynomial time. Otherwise, such a Turing machine could sort PS(A), which is identical to calculating the sorting key of A, without calculating every element of PS(A) or reordering A (since reordering A invalidates the sorting key of PS(A)), which is a logical contradiction
- It is implied that a verifier can verify M's superpolynomial decision problem in polynomial time, given A and the sorting key of PS(A), by using ALGORITHM to verify the sorting key, then deciding if the sorting key is even (decide YES) or odd (decide NO)
    - This implies P ≠ NP