Finding a Divisible Pair *

Stelian Ciurea † Erik D. Demaine ‡ Corina E. Pătraşcu § Mihai Pătraşcu ‡

Abstract

Our problem is the natural algorithmic version of a classic mathematical result: any $(n+1)$-subset of $\{1,\ldots,2n\}$ contains a pair of divisible numbers. How do we actually find such a pair? If the subset is accessible only through a membership oracle, we show a lower bound of $\frac{4}{3}n - O(1)$ and an almost matching upper bound of $\left(\frac{4}{3}+\frac{1}{24}\right)n + O(1)$ on the number of queries necessary in the worst case.

1 Introduction

A natural formalization of our problem is in an oracle model: the $(n+1)$-subset is not known to the algorithm, and is only accessible through membership queries asked to the oracle. The main results of this paper will be a lower bound of $\frac{4}{3}n - O(1)$ on the number of queries necessary in the worst case, and an almost matching upper bound of $\left(\frac{4}{3}+\frac{1}{24}\right)n + O(1)$. We believe the upper bound can be improved to match the lower bound, but we were unable to do that.

We begin with the folklore proof of the existence of a divisible pair, which immediately implies an algorithm making $O(n)$ queries.

Theorem 1. *For any $S \subset \{1,\ldots,2n\}$, $|S| = n+1$, there exist $a, b \in S$ such that a divides b.*

Proof. For every odd number $q \in \{1,3,5,\ldots,2n-1\}$, let $B_q = \{q \cdot 2^i \mid 0 \le i \le \log_2\lfloor n/q \rfloor\}$ – that is, a bin with all power-of-two multiples of q. Since every number is in the bin generated by its largest odd divisor, we have $B_q \cap B_r = \emptyset$ and $\bigcup B_q = \{1,\ldots,2n\}$. There are exactly n such bins, so at least one bin must contain two elements $a, b \in S$. By construction of the bins, a and b are a divisible pair. □

This proof is quite strong: it not only tells us that a divisible pair exists, but that one exists in which one number is a power-of-two multiple of the other! We note that this immediately gives an algorithm which makes at most $\frac{3}{2}n + O(1)$ queries. This is because odd numbers greater than n need never be considered. Indeed, these numbers have no multiple below $2n$, so they cannot be part of a pair where the two numbers differ by a power of two factor.

2 A Good Lower Bound

In this section, we prove a lower bound of $4n/3 - O(1)$ on the number of queries necessary in the worst case. The strategy of the adversary is simple. The first query to every pair $(x, 2x)$, with $x \in \{2n/3+1,\ldots,n\}$, receives a positive answer. As long as the adversary has a choice, i.e. it has not already declared $n-1$ numbers to be out of the set, the second query made to a pair receives a negative answer. Numbers greater than n which are not part of such a pair are always included in the set, so an algorithm which knows the adversary need not ask about these. Numbers below $2n/3$ are *in principle* not included in the set. However, the very last one probed might be included in the set, if the adversary has already declared $n-1$ values to be out of the set.

The lower bound comes from analyzing the number of queries needed to fix the divisible pair. Once the pair is fixed, an optimal algorithm need not actually query the two numbers in the pair. First note that our adversary never generates a divisible pair unless it has already declared $n-1$ numbers to be out of the set. Indeed, before this inevitable moment, no number smaller

*This material originally appeared as part of the paper "Finding a Divisible Pair and a Good Wooden Fence" by the same authors, published in the Proceedings of the 3rd International Conference on Fun with Algorithms (FUN 2004).

†Universitatea Lucian Blaga, Sibiu, Romania; stelian.ciurea@ulbsibiu.ro
‡MIT, Computer Science and Artificial Intelligence Laboratory; edemaine@mit.edu
§Harvard University, Department of Mathematics; patrascu@fas.harvard.edu

than $2n/3$ is included in the set (so numbers above n which are always included have no divisors in the set), and only one number from each special pair is declared to be in.

Therefore, a divisible pair is not fixed, unless $n-1$ queries have already received a negative answer. However, at most $2n/3$ of these can come from positions $1, \ldots, 2n/3$. Therefore, at least $n/3-1$ must come from one of the special pairs. Now remember that the first query touching a special pair always receives a positive answer. So in addition to the $n-1$ negative answers, the adversary must also have given $n/3-1$ positive answers.

Theorem 2. *Finding a divisible pair requires at least $\frac{4}{3}n - O(1)$ queries in the worst case.*

3 A Good Upper Bound

It can be seen that we cannot hope to beat the $3n/2$ barrier by searching only for power-of-two multiples. However, it might seem that a strategy that probes all multiples of an element can do even worse than $3n/2$, since it foregoes even the basic guarantee of not touching odd numbers greater than n. Surprisingly, a naive algorithm which probes all i's in increasing order, and for every i in the set probes all its multiples, has relatively good performance. We can prove that this algorithm does at most $\left(\frac{4}{3}+\frac{1}{12}\right)n+O(1)$ queries, and we can exhibit a family of inputs on which this bound is tight. All the ideas necessary to prove this will also appear in the proofs of this section.

In this section, we analyze a simple improvement to this strategy. The algorithm considers all numbers i in increasing order. Usually, i is probed, and if it is in the set, its multiples are also probed. However, if $i > 3n/2$ and its single multiple $2i$ has previously been probed and received a negative answer, there is no point in querying i. We will show that the performance of this improved algorithm is $\left(\frac{4}{3}+\frac{1}{24}\right)n+O(1)$, which almost matches our lower bound.

For the purpose of the analysis, we break the execution of this algorithm into two stages. The first stage lasts as long as $i \le 2n/3$. We begin with the case when the algorithm finishes during the first stage, i.e. with $i \le 2n/3$. In this case, we show the algorithm does at most $4n/3+O(1)$ queries. The algorithm can ask at most $n-1$ queries which receive a negative answer, so we must prove that at most $n/3+O(1)$ queries can receive a positive answer. At most one query for a number greater than $2n/3$ can receive a positive answer, because such a number is only probed when one of its divisors is in the set, so we stop after the first positive answer. On the other hand, at most $n/3+1$ queries from the range $\{1, \ldots, 2n/3\}$ can receive a positive answer. Indeed, by fact 1 a $(n/3+1)$-subset of $\{1, \ldots, 2n/3\}$ contains a divisible pair, so if we have received $n/3+1$ positive answers, we have also found a divisible pair.

Now assume that the algorithm finishes only in the second stage. In this case, all numbers between 1 and $2n/3$ are probed. Let T be the set of such numbers which received a positive answer, and let N be the set of probed numbers which received a negative answer during the first stage. Given this, exactly $n-1-|N|$ numbers are outside the set and not yet discovered. For every $i > 2n/3$, we only probe i if $2i$ has not been probed already. Thus, either i and $2i$ are both in the set and the algorithm stops, or we discover one new number that is outside the set (and possibly also one that is in the set). Thus, the number of queries done in the second stage is at most $2+2(n-1-|N|) = 2n-2|N|$. The total number of queries must then be at most $|T|+|N|+2n-2|N| = 2n-(|N|-|T|)$. We will show below that $|N| \ge \left(\frac{2}{3}-\frac{1}{24}\right)n+|T|$, which immediately implies our desired bound.

To prove our bound on $|N|$, first note that N contains exactly $2n/3-|T|$ numbers from $\{1, \ldots, 2n/3\}$. Let $M = N \cap \{2n/3+1, \ldots, 2n\}$. Our bound is equivalent to $|M| \ge 2|T| - \frac{n}{24}$. First note that M contains all multiples greater than $2n/3$ of numbers from T, and in particular all power-of-two multiples. There are 2 power-of-two multiples for every number in $T \cap \{1, \ldots, n/2\}$ and one for the rest of T. However, for every number in T between $n/2+1$ and $2n/3$, M must also contain its triple. Thus, we have identified two multiples belonging to M for every number from T. No two numbers from T can have a power-of-two multiple in common, since they would be divisible, and the algorithm would stop during the first stage. Thus, the only double counting can come from triples, namely when $3x = 2^i y$ with some $y \in T, x \in T \cap \{n/2+1, \ldots, 2n/3\}$. Clearly, $3x \in \{3n/2+3, \ldots, 2n\}$, so $2^{i-1}y \in \{3n/4+1, \ldots, n\}$. In addition, $2^i y$ must be a multiple of three, so $2^{i-1}y$ must also be a multiple of three. Finally, $2^{i-1}y$ must be even, because $y \le 2n/3$, so $i \ge 2$. Thus, $2^{i-1}y$ must be a multiple of 6 in the range $\{3n/4+1, \ldots, n\}$. Since there are only $n/24$ such possibilities, and each one defines at most one double-counted multiple, we obtain $|M| \ge 2|T| - n/24$, which completes our proof.

Theorem 3. *In the worst case, $\left(\frac{4}{3}+\frac{1}{24}\right)n+O(1)$ queries are sufficient to find a divisible pair.*