On algorithms to find p-ordering *

Aditya Gulati, Sayak Chakrabarti, and Rajat Mittal

IIT Kanpur
{aditg, sayak, rmittal}@iitk.ac.in

Abstract. The concept of *p-ordering* for a prime p was introduced by Manjul Bhargava (in his PhD thesis) to develop a generalized factorial function over an arbitrary subset of integers. This notion of p-ordering provides a representation of polynomials modulo prime powers, and has been used to prove properties of roots sets modulo prime powers. We focus on the complexity of finding a p-ordering given a prime p, an exponent k and a subset of integers modulo p^k.
Our first algorithm gives a p-ordering for a set of size n in time $\widetilde{\mathcal{O}}(nk \log p)$, where set is considered modulo p^k. The subsets modulo p^k can be represented concisely using the notion of representative roots (Panayi, PhD Thesis, 1995; Dwivedi et al., ISSAC, 2019); a natural question is, can we find a p-ordering more efficiently given this succinct representation. Our second algorithm achieves precisely that, we give a p-ordering in time $\widetilde{\mathcal{O}}(d^2k \log p + nk \log p + nd)$, where d is the size of the succinct representation and n is the required length of the p-ordering. Another contribution is to compute the structure of roots sets for prime powers p^k, when k is small. The number of root sets have been given before (Dearden and Metzger, Eur. J. Comb., 1997; Maulick, J. Comb. Theory, Ser. A, 2001), we explicitly describe all the root sets for $k \leq 4$.

Keywords: root-sets · p-ordering · polynomials · prime powers

1 Introduction

Polynomials over finite fields have played a crucial role in computer science with impact on diverse areas like error correcting codes [8,15,23,24], cryptography [11,18,21], computational number theory [1,2] and computer algebra [17,25]. Mathematicians have studied almost all aspects of these polynomials; factorization of polynomials, roots of a polynomial and polynomials being irreducible or not are some of the natural questions in this area. There is lot of structure over finite field; we can deterministically count roots and find if a polynomials is irreducible in polynomial time [19]. Not just that, we also have efficient randomized algorithms for the problem of factorizing polynomials over finite fields [3,9].

The situation changes drastically if we look at rings instead of fields. Focusing our attention on the case of numbers modulo a prime power (ring $\mathbb{Z}_{p^k}$, for a prime p and a natural number $k \geq 2$) instead of numbers modulo a prime (field

* This paper is a contribution to the special issue of CALDAM 2021

$\mathbb{F}_p$), we don't even have unique factorization and the fact that the number of roots are bounded by the degree of the polynomial. Still, there has been some interesting work in last few decades. Maulik [20] showed bound on number of roots sets, sets which are roots for a polynomial modulo a prime power. There has been some recent works giving a randomized algorithm for root finding [4] and a deterministic algorithm for root counting [10,14].

The concept of *p-ordering* and *p-sequences* for a prime p, introduced by Bhargava [5], is an important tool in studying the properties of roots sets and polynomials over powers of the prime p [5,7,20]. Bhargava's main motivation to introduce p-ordering was to generalize the concept of factorials ($n!$ for $n \geq 0 \in \mathbb{Z}$) from the set of integers to any subset of integers. He was able to show that many number-theoretic properties of this factorial function (like the product of any k consecutive non-negative integers is divisible by $k!$) remain preserved even with the generalized definition for a subset of integers [6].

For polynomials, Bhargava generalized Polya's theorem, showing that the GCD of the outputs of a degree k polynomial on a subset S of integers divides the analogous factorial of k for S. He also gave a characterization of polynomials which are integer valued on a subset of integers. Both results use a convenient basis, using p-ordering, for representing polynomials on a subset of integers.

A similar convenient basis can be obtained for subsets of $\mathbb{Z}_{p^k}$. An interesting problem for polynomials over rings, of the kind $\mathbb{Z}_{p^k}$, is to find the allowed *root sets* (Definition 4). Maulik [20] was able to use this representation of polynomials over $\mathbb{Z}_{p^k}$ (from p-ordering) to give asymptotic estimates on the number of root sets modulo a prime power p^k; he also gave a recursive formula for root counting.

Our contributions While a lot of work has been done on studying the properties of p-orderings [7,16,20], there's effectively no work on finding the complexity of the problem: given a subset of numbers modulo a prime power, find a p-ordering. Our main contribution is to look at the computational complexity of finding p-ordering in different settings. We also classify and count the root-sets for $\mathbb{Z}_{p^k}$, when $k \leq 4$, by looking at their symmetric structure.

- *p-ordering for a general set:* Suppose, we want to find the p-ordering of a set $S \subseteq \mathbb{Z}_{p^k}$ such that $|S| = n$. A naive approach gives a $\widetilde{\mathcal{O}}(n^3 k \log(p))$ time algorithm. We exploit the recursive structure of p-orderings and optimize the resulting algorithm using data structures. These optimizations allow us to give an algorithm that works in $\widetilde{\mathcal{O}}(nk \log(p))$ time. The details of the algorithm, its correctness and time complexity is given in Section 3.
- *p-ordering for a subset in representative root representation:* A polynomial of degree d in $\mathbb{Z}_{p^k}$ can have exponentially many roots, but they can have at most d *representative roots* [4,14,22] giving a succinct representation. In general, any efficient algorithm for root finding or factorization should output the roots in form of such kind of representative roots (the complete set of roots could be exponentially large). The natural question is, can we have an efficient algorithm for finding a p-ordering where the complexity scales according to the number of representative roots and not the size of

the complete set. We answer this in affirmative, and provide an algorithm which works in $\widetilde{\mathcal{O}}(d^2k\log p + nk\log p + nd)$ time, where d is the number of representative roots and n is the length of p-ordering. The details of this algorithm and its analysis are presented in Section 4.

- *Roots sets for small powers:* A polynomial in $\mathbb{Z}_{p^k}$, even with small degree, can have exponentially large number of roots. Still, not all subsets of $\mathbb{Z}_{p^k}$ are a root-set for some polynomial. The number of root-sets for the first few values of k were calculated numerically by Dearden and Metzger [13]. Building on previous work, Maulik [20] produced an upper bound on the number of root-sets for any p and k. He also gave a recursive formula for the exact number of root-sets using the symmetries in their structure. We look at the structure of these root sets and completely classify all possible root-sets for $k \leq 4$ (we describe these in Section 5).

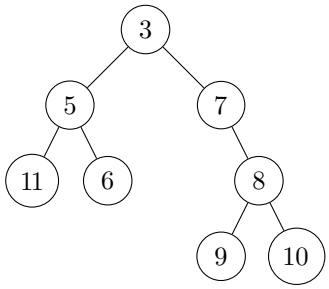
2 Preliminaries

2.1 Notation and time complexity:

Our primary goal is to find a p-ordering of a given set $S \subseteq \mathbb{Z}_{p^k}$, for a given prime p and an integer $k > 0$. Since the input size is polynomial in $|S|, \log p, k$; an efficient algorithm should run in time polynomial in these parameters. For the sake of clarity, $\log k$ factors will be ignored from complexity calculations; this omission will be expressed by using notation $\widetilde{\mathcal{O}}$ instead of $\mathcal{O}$ in time complexity. We also use $[n]$ for the set $\{0, 1, \ldots, n-1\}$.

2.2 Min-heap:

We will also use min-heap data structure. It is a tree in which each node has at most two children and exactly one parent node (except root, no parents). The defining property is that the key value of any node is equal or lesser than the key value of its children.



We will use three standard functions on a min-heap with n nodes [12].

1. CREATE_MIN_HEAP(S): Takes a set S as input and returns a min-heap with elements of S as the nodes in $\widetilde{\mathcal{O}}(n)$.
2. EXTRACT_MIN(H): Removes the element with the minimum key from the heap and rebalances the heap structure in $\widetilde{\mathcal{O}}(\log(n))$.
3. INSERT(H, a): Inserts the element a into the heap H in $\widetilde{\mathcal{O}}(\log(n))$.

2.3 *p*-ordering

Definition 1. *For any ring S with the usual operations $+$ and $*$, we define*

$$S + a := \{x + a \mid x \in S\} \quad \textit{and} \quad a * S := \{a * x \mid x \in S\}$$

To define p-ordering, we begin by defining the valuation of an integer modulo a prime p.

Definition 2. *Let p be a prime and $a \neq 0$ be an integer. The* valuation *of the integer a modulo p, denoted $v_p(a)$, is the integer v such that $p^v \mid a$ but $p^{v+1} \nmid a$. We also define $w_p(a) := p^{v_p(a)}$.*

If $a = 0$ then both, $v_p(a)$ and $w_p(a)$, are defined to be ∞.

Bhargava [5] introduced the concept of p-ordering for any subset of a Dedekind domain. We restrict to the rings of the form $\mathbb{Z}_{p^k}$ (similar to $\mathbb{Z}$) which has been explained in [6].

Definition 3 ([5]). *p-ordering on a subset S of $\mathbb{Z}_{p^k}$ is defined inductively.*

1. *Choose any element $a_0 \in S$ as the first element of the sequence.*
2. *Given an ordering $a_0, a_1, \ldots a_{i-1}$ up to $i-1$, choose $a_i \in S \backslash \{a_0, a_1 \ldots a_{i-1}\}$ which minimizes $v_p((a_i - a_0)(a_i - a_1)\ldots(a_i - a_{i-1}))$.*

The i-th element of the associated *p-sequence* for a p-ordering $a_0, a_1, \ldots a_n$ is defined by

$$v_p(S, i) = \begin{cases} 0 & i = 0, \\ v_p((a_i - a_0)(a_i - a_1)\ldots(a_i - a_{i-1})) & i > 0. \end{cases}$$

In the $(i+1)$-th step, let $x \in S \setminus \{a_0, a_1, ..., a_{i-1}\}$ then the value $v_p((x-a_0)(x-a_1)\ldots(x-a_{i-1}))$ is denoted as the *p-value* of x at that step. If the step is clear from context, we call the p-value of that element at that step as its p-value.

To take an example, $S = \{1, 3, 4, 6, 9, 10\} \in \mathbb{Z}_{3^3}$ has $(4, 6, 1, 9, 3, 10)$ and $(3, 10, 6, 4, 9, 1)$ as two valid 3-orderings. The 3-sequence associated with both these 3-orderings is $(1, 1, 3, 3, 9, 27)$. At the first glance it is not clear why associated p-sequences are same. In fact, Bhargava proved the following theorem.

Theorem 1 ([5]). *For any two p-orderings of a subset $S \subseteq \mathbb{Z}$ and a prime p, the associated p-sequences are same.*

We notice few more facts about p-ordering.

Observation 2. *Let S be a subset of integers, let $(a_0, a_1, a_2, ...)$ be a p-ordering on S, then*

1. *For any $x \in \mathbb{Z}$, $(a_0 + x, a_1 + x, a_2 + x, ...)$ is a p-ordering on $S + x$.*
2. *For any $x \in \mathbb{Z}$, $(x * a_0, x * a_1, x * a_2, ...)$ is a p-ordering on $x * S$.*

Observation 3. *Let S be a subset of integers, let $(a_0, a_1, a_2, ...)$ be a p-ordering on S. Then, for any $x \in \mathbb{Z}$*

1. $v_p(x * S, k) = v_p(S, k) + k \cdot v_p(x)$.
2. $v_p(S + x, k) = v_p(S, k)$.

Theorem 4 ([20]). *Let S be a subset of integers, let $S_j = \{s \in S \mid s \equiv j \pmod p\}$ for $j = 0, 1, ..., p-1$, then for any $x \in \mathbb{Z}$, s.t. $x \equiv j \pmod p$,*

$$w_p\left(\prod_{a_i \in S}(x - a_i)\right) = w_p\left(\prod_{a_i \in S_j}(x - a_i)\right). \tag{1}$$

2.4 Root sets and representative roots:

Definition 4. *A given set S is called a* root set *in a ring R if there is a polynomial $f(x) \in R[x]$, whose roots in R are exactly the elements of S.*

It is non-trivial to check if a subset is a root set. For example, $\{0, 3\}$ is not a root set in $\mathbb{Z}_{3^2}$, whereas a large set $\{0, 3, 6, 9, 12, 15, 18, 21, 24\} \in \mathbb{Z}_{3^3}$ is a root set for a small degree polynomial x^3.

The notion of representative roots in the ring $\mathbb{Z}_{p^k}$ has been used to concisely represent roots of a polynomial [4,14,22].

Definition 5. *The representative root $(a + p^i*)$ is a subset of $\mathbb{Z}_{p^k}$,*

$$a + p^i* := \{a + p^i y \mid y \in \mathbb{Z}_{p^{k-i}}\}$$

For example, the set $\{1, 26, 51, 76, 101\} \in \mathbb{Z}_{5^3}$ can be represented as $1 + 25*$. It gives a powerful way to represent big subsets concisely; a polynomial of degree d in $\mathbb{Z}_{p^k}$ can have at most d *representative roots* [4,14,22] (but exponentially many roots). *Extending* a set $S = \{r_1, \cdots, r_l\}$ of representative roots corresponds to the subset $\bigcup_{i=1}^{l} r_i \subseteq \mathbb{Z}_{p^k}$. Conversely, we show that an $S \subseteq \mathbb{Z}_{p^k}$ can be uniquely represented by representative roots.

Definition 6. *Let $S \subseteq \mathbb{Z}_{p^k}$, then the set of representative roots $S^{rep} = \{r_1, ..., r_l\}$ (for $r_i = \beta_i + p^{k_i}*$, for some $\beta_i \in \mathbb{Z}_{p^k}$ and $k_i \in [k]$) is said to be a minimal root set representation of S if*

1. $S = \bigcup_{i=1}^{l} r_i$,
2. $\nexists\ r_i, r_j \in S^{rep} : r_i \subseteq r_j$,
3. $\forall i : \bigcup_{b \in [p]} \left(r_i + p^{k_i - 1} \cdot b\right) \nsubseteq S$

Theorem 5. *Given any set $S \subseteq \mathbb{Z}_{p^k}$, the minimal root set representation of S is unique.*

Proof. For the sake of contradiction, let S^{rep} and $\widehat{S^{rep}}$ be two different minimal representations of a set S. There exists an $a \in S$ such that it belongs to both representations, $r \in S^{rep}$ and $\widehat{r} \in \widehat{S^{rep}}$ and $r \neq \widehat{r}$. Then r can be written as $a + p^{k_1}*$ and $\widehat{r}$ can be written as $a + p^{k_2}*$.

By Observation 6, $r \cap \widehat{r} \neq \emptyset$ implies $r \subset \widehat{r}$ or $\widehat{r} \subset r$. Without loss of generality, let $\widehat{r} \subset r$ (equivalently $k_1 < k_2$).

From $\widehat{r} \subset r$ and $k_1 < k_2$, $(\widehat{r} + b \cdot p^{k_2-1}) \subseteq r$ for all $b \in [p]$. Using $r \subseteq S$, we get

$$\bigcup_{b \in [p]} \left(\widehat{r} + b \cdot p^{k_2-1}\right) \subseteq S,$$

contradicting minimality of $\widehat{S^{rep}}$. □

We note a few observations about representative roots.

Observation 6. *Given any two representative roots $A_1 = \beta_1 + p^{k_1}*$ and $A_2 = \beta_2 + p^{k_2}*$, then either $A_1 \subseteq A_2$ or $A_2 \subseteq A_1$ or $A_1 \cap A_2 = \emptyset$.*

Proof. Let $A_1 = \beta_1 + p^{k_1}*$ and $A_2 = \beta_2 + p^{k_2}*$ be two root sets such that $A_1 \cap A_2 \neq \emptyset$, then we show that $A_1 \subseteq A_2$ or $A_2 \subseteq A_1$.

Without loss of generality, let's assume that $k_1 \leq k_2$. Let there be some element $a \in A_1 \cap A_2$. Then, A_1 can be defined as $A_1 = a + p^{k_1}*$ and similarly $A_2 = a + p^{k_2}*$.

Let $b \in A_2$, then $b = a + p^{k_2}y$ for some $y \in \mathbb{Z}_{p^{k-k_2-1}}$. Now, we know that $\forall z \in \mathbb{Z}_{p^{k-k_1-1}}$, $a + p^{k_1}z \in A_1$. Hence, for $z = p^{k_2-k_1}y$, we get $a + p^{k_1} \cdot (p^{k_2-k_1}y) = b$, hence $b \in A_1$. Hence, $A_2 \subseteq A_1$ and $\tilde{A} = A_2$. □

Observation 7. *Let $A_1 = \beta_1 + p^{k_1}*$ and $A_2 = \beta_2 + p^{k_2}*$ with $A_1 \cap A_2 = \emptyset$. Let $a_1 \in A_1$ and $a_2 \in A_2$, then,*

$$v_p(a_1 - a_2) = v_p(\beta_1 - \beta_2).$$

Proof. WLOG assume that $k_1 \leq k_2$ and $\beta_1 \in \mathbb{Z}_{p^{k_1}}, \beta_2 \in \mathbb{Z}_{p^{k_2}}$. Let $a_1 = \beta_1 + p^{k_1}a_1'$ and $a_2 = \beta_2 + p^{k_2}a_2'$.

Now, we have

$$a_1 - a_2 = (\beta_1 - \beta_2) + p^{k_1}(a_1' - p^{k_2-k_1}a_2'). \tag{2}$$

If $v_p(a_1 - a_2) \geq k_1$, then this would imply $v_p(\beta_1 - \beta_2) \geq k_1$, which means $\beta_2 + p^{k_2}a_2' \in A_1$. Observation 6 implies that $A_2 \subseteq A_1$, which is not true.

This shows that $v_p(a_1 - a_2) < k_1$, and since the second term in equation 2 is $p^{k_1}(a_1' - p^{k_2-k_1}a_2')$, it does not contribute to the valuation. Hence $v_p(a_1 - a_2) = v_p(\beta_1 - \beta_2)$. □

Observation 8. *Let $(a_0, a_1, ...)$ be a p-ordering on $\mathbb{Z}_{p^k}$, then $(\beta + a_0 * p^j, \beta + a_1 * p^j, \beta + a_2 * p^j, ...)$ is a p-ordering on $\beta + p^j*$.*

Proof. A simple proof of this theorem follows from Observation 2 and the fact that $1, 2, 3, \ldots$ form an obvious p-ordering in $\mathbb{Z}_{p^k}$. □

Lemma 1. *Let $S_j := \{s \in \mathbb{Z}_{p^k} \mid s \equiv j \bmod p\}$ and $f(x)$ be a polynomial in $\mathbb{Z}_{p^k}[x]$ with root-set A. If there exist k elements (say $\alpha_1, ..., \alpha_k$) in $A \cap S_j$ such that $\alpha_l \not\equiv \alpha_m \bmod p^2$ for all (l, m) pairs, then $S_j \subseteq A$. (Notice that this implies $k < p$.)*

Proof. Let $f(x) = \sum_{i=0}^{n} b_i x^i$. Let, for all $i \in [k]$, $\alpha_l = j + p \cdot \beta_l$, then since α_l is in the root set of $f(\cdot)$, therefore,

$$f(j + p * \beta_l) = \sum_{i=0}^{n} b_i \cdot (j + p * \beta_l)^i \equiv 0 \bmod p^k.$$

Hence, for all $l \in [k]$

$$\sum_{i=0}^{k-1} p^i \cdot \beta_l^i \cdot g_i(j) \equiv 0 \bmod p^k,$$

where, $g_i(x) = \sum_{m=0}^{n-i} \binom{m+i}{m} \cdot b_{m+i} \cdot x^m$. Writing this system of equations in the form of matrices $B \cdot X = 0 \bmod p^k$, we get,

$$\begin{bmatrix} 1 & \beta_0 & \cdots & \beta_0^{k-1} \\ 1 & \beta_1 & \cdots & \beta_1^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \beta_{k-1} & \cdots & \beta_{k-1}^{k-1} \end{bmatrix} \begin{bmatrix} g_0(j) \\ p \cdot g_1(j) \\ \vdots \\ p^{k-1} \cdot g_{k-1}(j) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \bmod p^k.$$

Here, since, B is a Vandermonde matrix, $|det(B)| = \left| \prod_{i \neq j \in [k]} (\beta_i - \beta_j) \right|$. Since $\beta_i - \beta_j \not\equiv 0 \bmod p$, therefore, $det(B) \not\equiv 0 \bmod p$ ($det(B)$ isn't a zero divisor). Hence, B has an inverse. Multiplying by the inverse on both sides, we get,

$$\begin{bmatrix} g_0(j) \\ p \cdot g_1(j) \\ \vdots \\ p^{k-1} \cdot g_{k-1}(j) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \bmod p^k, or,$$

for $i \in [k]$, $g_i(j) \equiv 0 \bmod p^{k-i}$. Hence, for any element $j + p \cdot \beta \in S_j$, $f(j + p * \beta) = \sum_{i=0}^{k-1} p^i \cdot \beta_l^i \cdot g_i(j) \equiv 0 \bmod p^k$ (since $p^i \cdot g_i(j) \equiv 0 \bmod p^k$). Therefore, all elements of S_j are a root of $f(\cdot)$, or $S_j \subseteq A$. □

3 Algorithm to Find p-ordering on a Given Set

A simple algorithm to compute p-ordering was given by [5]. We restrict ourselves to subsets of $\mathbb{Z}_{p^k}$ and describe that algorithm, which can be directly derived from the definition of p-ordering.

In order to compute a p-ordering, suppose we have already chosen

$\{a_0, a_1, \ldots a_{t-1}\}$ forming a p-ordering of length t. We choose the next element from $S \backslash \{a_0, a_1, \ldots a_{t-1}\}$ such that $v_p((x-a_0)(x-a_1)\ldots(x-a_{t-1}))$ is minimum, which gives a p-ordering of length $t+1$. Given a set of integers $S \subseteq \mathbb{Z}_{p^k}$ we can find a p-ordering by naively checking the element which will give us the minimum valuation with respect to p for the given expression as in Definition 3. The naive approach given in [5] iterates over all x in $S \backslash \{a_0, a_1, \ldots a_{t-1}\}$ and adds the element to the p-ordering which gives the minimum valuation.
Suppose, we add an element to the already existing p-ordering of length t. For any given value of $x \in S \backslash \{a_0, a_1, \ldots a_{t-1}\}$, calculating $x - a_i$ and multiplying for every $0 \geq i < t$ takes $\mathcal{O}((n-t)t)$ operations in $\mathbb{Z}$ and since each of them are less than p^k, this takes $\mathcal{O}((n-t)tk \log p) \leq \mathbb{O}(n^2 k \log p)$. So repeating this n times gives us the time complexity $\mathcal{O}(n^3 k \log p)$.

In this section, we give an algorithm to output the p-ordering more efficiently. The main result of this section is the following theorem.

Theorem 9. *Given a set $S \subseteq \mathbb{Z}$, a prime p and an integer k such that each element of S is less than p^k, we can find a p-ordering on this set in $\widetilde{\mathcal{O}}(nk \log p)$ time.*

The proof of Theorem 9 follows from the construction of Algorithm 1.

Outline of Algorithm 1 We use the recursive structure of p-ordering given by Maulik [20]. Crucially, to find the p-value of an element a at each step, we only need to look at elements congruent to $a \mod p$ (Theorem 4).

Suppose S_j is the set of elements of S congruent to $j \bmod p$. By the observation above, our algorithm constructs the p-ordering of set S by merging the p-ordering of S_j's. Given a p-ordering up to some step, the next element for the p-ordering of S is computed by just comparing the first elements in p-ordering of S_j's (not present in the already computed p-ordering). The p-ordering of the *translated* S_j's is computed recursively (Observation 2).

While merging the p-orderings on each of the S_i's, at each step we need to extract and remove the element with the minimum p-value over all S_j's and replace it with the next element from the p-ordering on the same set S_j. Naively, it would need to find the minimum over all possible j's taking $\widetilde{\mathcal{O}}(p)$ time. Instead, we use min heap data structure, using only $\widetilde{\mathcal{O}}(\log p)$ time for extraction and insertion of elements.

Each node of the min-heap(H) contains the value of the element (given as input) as *value* and a key, p-value, that contains the p-value of the element present in the heap. For every element there is another key *set*, which stores the index of the subsets $S_0, S_1, \ldots S_{p-1}$ to which it belongs. The heap is sorted in terms of the p-value and whenever we do any operation like extracting the minimum, or adding to the heap, we consider ordering given by increasing value of the p-value attribute of the nodes.

Algorithm 1 Find p-ordering

1: **procedure** MERGE($S_0, S_1, ..., S_{p-1}$)
2: $S \leftarrow [\,]$
3: **for** $i \in [0, 1, ..., p-1]$ **do**
4: **for** $j \in [0, ..., len(S_i) - 1]$ **do**
5: $S_i[j].set \leftarrow i$
6: $i_0, i_1, i_2, ...i_{p-1} \leftarrow (0, 0, ..., 0)$
7: $H \leftarrow$ CREATE_MIN_HEAP($node = \{S_0[i_0], ..., S_{p-1}[i_{p-1}]\}, key = p\_value$)
8: **while** H.IsEmpty()!=true **do**
9: $a \leftarrow$ EXTRACT_MIN(H)
10: $j \leftarrow a.set$
11: **if** $i_j < len(S_j)$ **then**
12: $i_j \leftarrow i_j + 1$
13: INSERT($H, S_j[i_j]$)
14: $S \leftarrow a$
15: **return** S
16: **procedure** FIND_p-ORDERING(S)
17: **if** length(S)==1 **then**
18: $S[0].p\_value \leftarrow 0$
19: **Return** S
20: $S_0, S_1, ..., S_{p-1} \leftarrow ([\,], [\,], ..., [\,])$
21: **for** $m \in S$ **do**
22: $t \leftarrow m.value \mod p$
23: Append m to S_t
24: **for** $i \in [0, 1, ..., p-1]$ **do**
25: $S_i \leftarrow$ FIND_p-ORDERING($(S_i - i)/p$)
26: **for** $j \in [0, ..., len(S_i) - 1]$ **do**
27: $S_i[j].value \leftarrow p * S_i[j].value + i$
28: $S_i[j].p\_value \leftarrow S_i[j].p\_value + j$
29: $S \leftarrow$ MERGE($S_0, S_1, ..., S_{p-1}$)
30: **return** S

In Algorithm 1, we use a sorted list $\mathcal{I}$ of non-empty S_i's, and only iterate over $\mathcal{I}$ in steps 3-5, 23-28. Hence, decreasing the time complexities of these loops. We can create/update the list $\mathcal{I}$ in the loop at steps 21-23.

3.1 Proof of Correctness

To prove the correctness of Algorithm 1, we need two results: the valuation is maintained correctly in the algorithm and MERGE() procedure works correctly.

Theorem 10 (Correctness of valuations during translation). *In Algorithm 1, let S be a subset of integers, then FIND_p-ORDERING(S) gives a valid p-values for all elements of S, i.e. for the i^{th} element in S,*

$$S[i].p\_value = v_p\left(\prod_{j=0}^{i-1}(S[i].value - S[j].value)\right).$$

Proof. We prove this by induction on the size of S.

Let $|S| = 1$, then by the definition of $v_p(\cdot)$, we need to show $S[0].p\_value = 0$. Hence, it is clear (by step 18) that the base case is correct.

Let our assumption is true for $|S| < k$ and let $|S| = k$.

We divide the proof into 2 cases depending on whether S breaks into multiple S_i's or a single S_i in steps 21-23.

Case 1: Let S breaks into multiple S_i's.

It is easy to see that $|S_i| < k$ for all $i \in [p]$. Hence, the sizes of $(S_i - i)/p$ is also less than k for all $i \in [p]$. Hence, after FIND_p-ORDERING$((S_i - i)/p)$ in step 25, for the l^{th} element in S_i, we get

$$S_i[l].p\_value = v_p\left(\prod_{j\in[l]}\left(\frac{S_i[l].value - i}{p} - \frac{S_i[j].value - i}{p}\right)\right),$$

or

$$S_i[l].p\_value = v_p\left(p^{-l}\prod_{j\in[l]}(S_i[l].value - S_i[j].value)\right),$$

or

$$S_i[l].p\_value = v_p\left(\prod_{j\in[l]}(S_i[l].value - S_i[j].value)\right) - l.$$

Hence, at the end of step 27, for the l^{th} element in S_i, we get

$$S_i[l].p\_value = v_p\left(\prod_{j\in[l]}(S_i[l].value - S_i[j].value)\right).$$

It is easy to see that in MERGE(), each S_i $(i \in [p])$ forms a sub-sequence of MERGE$(S_0, S_1, ..., S_{p-1})$. Let the m^{th} element in S_i occurs at the l^{th} location in the output of MERGE$(S_0, S_1, ..., S_{p-1})$. By Theorem 4, it is easy to see that

$$v_p\left(\prod_{j\in[l]}(S[l].value - S[j].value)\right) = v_p\left(\prod_{j\in[m]}(S_i[m].value - S_i[j].value)\right).$$

Since, MERGE() does not change the p_value associated with any number, hence, $S[l].p\_value = S_i[m].p\_value$, or

$$S[l].p\_value = v_p\left(\prod_{j\in[l]}(S[l].value - S[j].value)\right).$$

Hence, the p-values at the end of MERGE() are correct (step 29). Hence, FIND_p-ORDERING(S) gives the correct p-values.

Case 2: Let all elements of S go into a single S_i.

Since $k > 1$, it is easy to see that at some point S breaks into multiple S_i's on which FIND_p-ORDERING(S) gives the correct p-values. Hence, given the correct p-values on FIND_p-ORDERING($(S_i - i)/p$) in step 25 ($|S_i| = k$), we need to show that we get the correct p-values on S at the end of step 29. Since FIND_p-ORDERING($(S_i - i)/p$) gives the correct p-values, for the l^{th} element in S_i, we get

$$S_i[l].p\_value = v_p\left(\prod_{j\in[l]}\left(\frac{S_i[l].value - i}{p} - \frac{S_i[j].value - i}{p}\right)\right),$$

or

$$S_i[l].p\_value = v_p\left(\prod_{j\in[l]}\left(S_i[l].value - S_i[j].value\right)\right) - l.$$

Now, at the end of step 28, for the l^{th} element in S_i, we get

$$S_i[l].p\_value = v_p\left(\prod_{j\in[l]}\left(S_i[l].value - S_i[j].value\right)\right).$$

It's easy to see that if we have only one of the S_i's is non-empty, the MERGE($S_0, S_1, ..., S_{p-1}$) just outputs that S_i as S (acts as identity). Therefore,

$$S[l].p\_value = v_p\left(\prod_{j\in[l]}\left(S[l].value - S[j].value\right)\right).$$

Hence, the output at the end of Step 29 has the correct p-values. Hence, FIND_p-ORDERING(S) gives the correct p-values for sets of size k.

Hence, by induction, FIND_p-ORDERING($\cdot$) outputs the correct p-values. □

Theorem 11 (Correctness of Merge()). *In Algorithm 1, given a p-ordering on each of the S_k's (for $k \in [p]$ and $a \in S_k$ only if $a \equiv k \pmod p$), MERGE($S_0, S_1, ..., S_{p-1}$) gives a valid p-ordering on $\bigcup_{k\in[p]} S_k$.*

Proof. We need to prove that, at the end of MERGE(), the list S is a valid p-ordering on $\bigcup_{k\in[p]} S_k$. Let $T := \left(\bigcup_{k\in[p]} S_k\right)$ and for each S_k ($k \in [p]$), $\left(a_0^k, a_1^k, ..., a_{|S_k|-1}^k\right)$ be the p-ordering given as input to MERGE(). From the description of MERGE(), it's easy to see that:

1. The element a_l^k (for some $l < |S_k|$) is added to the heap H only after the elements a_i^k (for all $i < l$) have been added to the list S.
2. At any point during MERGE(), at most one element from any S_k can belong to the heap.

We prove by induction that the elements in the list S form a correct p-ordering. We know that any element can be chosen as the first element by the definition p-ordering (Definition 3), hence, the base case is correct.

Let at the i^{th} step, $(a_0, a_1, ...a_{i-1})$ be a p-ordering on T, then the $(i+1)^{th}$ element $a_i \in T \setminus \{a_0, a_1, ...a_{i-1}\}$ is a valid element if

$$v_p\left(\prod_{j\in[i]}(a_i - a_j)\right) = \min_{x\in T\setminus\{a_0,a_1,...a_{i-1}\}}\left(v_p\left(\prod_{j\in[i]}(x - a_j)\right)\right).$$

By the 2^{nd} point above, we know that at most one element from any S_k is in H (for $k \in [p]$). Let the $(i_k+1)^{th}$ element of S_k be in H. Hence, the elements $(a^0_{i_0}, a^1_{i_1}, ..., a^{p-1}_{i_{p-1}})$ are in H. Then by the 1^{st} point above, the elements $(a^k_0, a^k_1, ..., a^k_{i_k-1}) \in S$ (for all $k \in [p]$).

Let the next element to be added to S (given by EXTRACT_MIN(H)) be $a^l_{i_l}$ (for some $l \in [p]$). Hence, we have to show that

$$v_p\left(\prod_{j\in[i]}(a^l_{i_l} - a_j)\right) = \min_{x\in T\setminus\{a_0,a_1,...a_{i-1}\}}\left(v_p\left(\prod_{j\in[i]}(x - a_j)\right)\right).$$

Let $x \in T \setminus \{a_0, a_1, ...a_{i-1}\}$ be any element. Since, $T = \left(\bigcup_{k\in[p]} S_k\right)$, $x \in S_k$ (for some $k \in [p]$). Now, since $\left(a^k_0, a^k_1, ..., a^k_{|S_k|-1}\right)$ is a p-ordering on S_k (we know the correct valuations are stored by Theorem 10), hence

$$v_p\left(\prod_{j\in[i_k]}(a^k_{i_k} - a^k_j)\right) = \min_{x\in S_k\setminus\{a^k_0,a^k_1,...a^k_{i_k-1}\}}\left(v_p\left(\prod_{j\in[i_k]}(x - a^k_j)\right)\right),$$

or

$$v_p\left(\prod_{j\in[i_k]}(a^k_{i_k} - a^k_j)\right) \le v_p\left(\prod_{j\in[i_k]}(x - a^k_j)\right).$$

By Theorem 4,

$$v_p\left(\prod_{j\in[i]}(a^k_{i_k} - a_j)\right) \le v_p\left(\prod_{j\in[i]}(x - a_j)\right).$$

Now, by the correctness of EXTRACT_MIN(),

$$v_p\left(\prod_{j\in[i_l]}(a^l_{i_l} - a^l_j)\right) \le v_p\left(\prod_{j\in[i_k]}(a^k_{i_k} - a^k_j)\right).$$

Again, by Theorem 4,

$$v_p\left(\prod_{j\in[i]}(a^l_{i_l} - a_j)\right) \le v_p\left(\prod_{j\in[i]}(a^k_{i_k} - a_j)\right).$$

Finally, combining the above 2 inequalities, we get

$$v_p\left(\prod_{j\in[i]}(a_{i_l}^l - a_j)\right) \leq v_p\left(\prod_{j\in[i]}(x - a_j)\right).$$

Since, x was any element in $T \setminus \{a_0, a_1, ...a_{i-1}\}$, we have

$$v_p\left(\prod_{j\in[i]}(a_{i_l}^l - a_j)\right) = \min_{x\in T\setminus\{a_0,a_1,...a_{i-1}\}}\left(v_p\left(\prod_{j\in[i]}(x - a_j)\right)\right).$$

Hence, by induction, at the end of MERGE(), the list S is a valid p-ordering on $\bigcup_{k\in[p]} S_k$. □

Using the above two theorems, we prove the correctness of Algorithm 1.

Proof of correctness of Algorithm 1. For the base case, if $|S| = 1$, then it's easy to see that FIND_p-ORDERING(S) gives the correct output (just the single element).

Let FIND_p-ORDERING(S) gives the correct p-ordering on S for $|S| < k$.

We assume that S breaks into multiple S_i's in steps 21-23. The proof for the case in which all elements of S got into a single S_i follows similarly.

It's easy to see that in this case, $|S_i| < k$ (for all $i \in [p]$). Hence, by the induction hypothesis, FIND_p-ORDERING($(S_i - i)/p$) outputs the correct p-ordering on each $(S_i - i)/p$ (step 25). From Observation 2, the p-ordering on each S_i is correct (step 27). Finally by Theorem 11, we get the correct p-ordering on S. By induction, our algorithm returns a valid p-ordering on any subset of integers. □

3.2 Time Complexity

Theorem 12. *Given a set $S \subset \mathbb{Z}$ of size n and a prime p, such that for all elements $a \in S$, $a < p^k$ for some k, Algorithm 1 returns a p-ordering on S in $\widetilde{\mathcal{O}}(nk \log p)$ time.*

Proof. We break the complexity analysis into 2 parts, the time complexity for merging the subsets S_i's and the time complexity due the to recursive step.

Time complexity of MERGE($S_0, S_1, ..., S_{p-1}$) in Algorithm 1 Let $|S_0| + |S_1| + ... + |S_{p-1}| = m$. Then, the time complexity of making the heap (Step 7) is $\widetilde{\mathcal{O}}(\min(m, p))$ (the size of the heap). Next, the construction of common p-ordering(Steps 8-14) takes $\widetilde{\mathcal{O}}(m \log p)$ time, this is because extraction of an element and addition of an element are both bound by $\widetilde{\mathcal{O}}(\log p)$ and the runs a total of m times. Hence, the total time complexity of MERGE($S_0, S_1, ..., S_{p-1}$) is $\widetilde{\mathcal{O}}(\min(m, p) + m \log p) = \widetilde{\mathcal{O}}(m \log p)$ time.

Time complexity of Algorithm 1 Let $|S| = n$ and $S \subset \mathbb{Z}_{p^k}$. Then the recursion depth of FIND_p-ORDERING(S) is bound by k. Now at each depth, all the elements are distributed into multiple heaps (of sizes $m_1, m_2, ..., m_q$). The sum of sizes of all smaller sets at a given depth is $\sum_{i=1}^{q} m_i = n$ and the time to run any depth is $\sum_{i=1}^{q} \widetilde{\mathcal{O}}(m_i \log p) = \widetilde{\mathcal{O}}(n \log p)$.

Hence, total time complexity for k depth is

$$\widetilde{\mathcal{O}}(nk \log p).$$

□

The proof of Theorem 9 follows from the proof of correctness of Algorithm 1 and time complexity obtained from Theorem 12.

4 Algorithm to Find p-ordering on a Set of Representative Roots

The notion of representative roots (Definition 5) allows us to represent an exponentially large subset of $\mathbb{Z}_{p^k}$ succinctly. Further imposing a few simple conditions on this representation, namely the minimal representation (Definition 6), our subset is represented in a unique way (Theorem 5). A natural question arises, can we efficiently find a p-ordering given a set in terms of minimal root set representation? For example, given a set $\{1, 2, 4, 7, 10, 11, 13, 16, 19, 20, 22, 25\}$ and prime $p = 3$, we can write this set as an union of root sets modulo 3^3 as $\{1 + 3*, \mathbf{2 + 3^2*}\}$. A 3-ordering on this set is $\{1, \mathbf{2}, 4, 7, \mathbf{11}, 10, \mathbf{20}, 13, 16, 19, 22, 25\}$. In this section we give an efficient algorithm to find a p-ordering of a given length n on a set expressed in minimal representation.

Theorem 13. *Given a set $S \subset \mathbb{Z}_{p^k}$, for a prime p and an integer k, that can be represented in terms of d representative roots, we can efficiently find a p-ordering of length n for S in $\widetilde{\mathcal{O}}(d^2 k \log p + nk \log p + np)$ time.*

Outline of the algorithm Similar to the previous section, we find p-ordering on different representative roots instead of congruence classes, and merge them together.

The important observation is, we already have a natural p-ordering defined on a representative root (Observation 8). Since a p-ordering on each representative root is already known, we just need to find a way to merge them. Merging was easy in Algorithm 1 because progress in any one of the p-ordering of an S_j did not affect the p-value of an element outside S_j. On the other hand, for this case the exact increase in the p-value is given by Observation 7, and is precisely equal to $v_p(\beta_i - \beta_j)$.

We are given with a set S containing d representative roots. Say, a root $S[i]$ is of the form $\{\beta_i + p^{\ell_i} * | i = 1, 2 \dots d\}$. This represents a set of values which can be specified only using β_i and ℓ_i. So, in each of $S[i]$'s, we store β_i as $S[i].value$ and ℓ_i as $S[i].exponent$. Further, we can assume that the representative roots

are disjoint. If they are not, one representative root will be contained in another (Observation 6), all such occurrences can be deleted in $\widetilde{\mathcal{O}}(d^2k\log p)$ time.

In the algorithm, we first run the CORRELATE() to store the values of $v_p(\beta_i - \beta_j)$ in the matrix $Corr$. Then the procedure p-EXPONENT_INCREASE() returns an array $p\_exponent$ that stores the increase in power of p with factorial, which is basically $v_p((j+1)! - j!)$.

In the main part of the algorithm, we maintain an array of size d to store the valuations that we would get whenever we add the next element from a representative root. To update the i^{th} value of this array when an element from the j^{th} representative root is added, we simply add the value $v_p(\beta_i - \beta_j)$ $(i \neq j)$. Hence, at each step we find the minimum value in this array (in $\widetilde{\mathcal{O}}$(d)) and add it to the combined p-ordering (in $\widetilde{\mathcal{O}}(1)$), then we update all the d values in this array (in $\widetilde{\mathcal{O}}$(d)). In case there are more than one indices having the same minimum value, we select the lowest index corresponding to the value. We repeat this process till we get the p-ordering of the desired length.

With the above intuition in mind, we develop Algorithm 2 to find the p-ordering of length n on a subset S of $\mathbb{Z}_{p^k}$ given in representative root representation.

4.1 Proof of Correctness

To prove the correctness of this algorithm, we first prove that valuations are correctly maintained.

Theorem 14. *In Algorithm 2, FIND_p-ORDERING(S, n) maintains the correct valuations on the set S of representative roots in valuations at every iteration of the loop.*

Proof. The array *valuations* is initialized to 0 in Step 17, which is equivalent to the power of the prime in a p-ordering on the null set. At this stage we can start with any element according to Definition 3.

Now, let us assume that we have correctly generated a p-ordering of length $\tilde{n}$, with $i_1, i_2, \ldots, i_{|S|}$-many elements from each of the representative roots in S.

Suppose we have generated a p-ordering upto length $\tilde{n}$ with $i_1, i_2 \ldots i_{|S|}$ being the number of elements from each representative root in S. We obviously have

$$i_1 + i_2 + \cdots + i_{|S|} = \tilde{n}.$$

In the next step (Step 23) of the loop, we will add another element to this p-ordering. Let this element be an element of the e^{th} representative root. According to Observation 7, this element will contribute a p-value of

$$v_p(\beta_t - \beta_e)$$

to the p-sequence from elements which correspond to the t^{th} representative root, for $t \neq e$. There are i_t-many elements already present in the p-ordering and hence

Algorithm 2 Find p-ordering from minimal notation

procedure CORRELATE(S)
  $Corr \leftarrow [0]_{len(S)\times len(S)}$
  $Corr \leftarrow [0]_{len(S)\times len(S)}$
  for $j \in [1, ..., len(S)]$ **do**
    for $k \in [1, ..., len(S)]$ **do**
      $Corr[j][k] \leftarrow v_p(S[j].value - S[k].value)$
  return $Corr$
procedure p-EXPONENT_INCREASE(n)
  $v_p(1) \leftarrow 1$
  for $j \in [1, ..., n]$ **do**
    $v_p((j+1)!) \leftarrow v_p(j+1) * v_p(j!)$
    $p\_exponent[j] \leftarrow v_p((j+1)!) - v_p(j!)$
  return $p\_exponent$
procedure FIND_p-ORDERING(S, n)
  $corr \leftarrow$ CORRELATE(S)
  $increase \leftarrow p$-EXPONENT_INCREASE(n)
  $valuations \leftarrow [0]_{|S|}$
  $p\_ordering \leftarrow \{\}$
  $i_1, i_2 \dots i_{|S|} \leftarrow 0$
  for $i \in \{1, 2, \dots n\}$ **do**
    $min \leftarrow \min\{valuations\}$
    $index \leftarrow \text{argmin}\{valuations\}$
    $p\_ordering.append(S[index].value + p^{S[index].exponent} * i_{index})$
    for $j \in [1, ..., len(S)]$ **do**
      if $j = index$ **then**
        $valuations[j] \leftarrow valuations[j] + S[index].exponent + increase[i_j]$
      else
        $valuations[j] \leftarrow valuations[j] + corr(index, j)$
    $i_{index} \leftarrow i_{index} + 1$
  return $p\_ordering$

the total power of p due to this new element from each of the representative roots will be $i_t v_p(\beta_t - \beta_e)$, for $t \neq e$.

Next, we find the p-value contributed due to the same representative root. Notice that, from Observation 8 we will have the elements of the e^{th} representative root as a p-ordering as well on $\beta_e + p^{k_e}*$, of length i_e. Now by Theorem 2, we will have this p-ordering on $\beta_j + p^{k_e}*$ as

$$\{\beta_e, \beta_e + p^{k_e}, \beta_e + p^{k_e}2, \dots \beta_e + p^{k_e}(i_e - 1)\}.$$

When we add another element to this the p-value contributed due to e^{th} representative root will be $k_e v_p(i_e!)$.

Summing them the total p-value at each step, considering the next element to be added being from e^{th} representative root is

$$\sum_{t\in[|S|];t\neq e} i_t v_p(\beta_t - \beta_e) + k_e v_p(i_e!). \tag{3}$$

In Steps 21-22, we choose e such that this expression is minimum in our algorithm.

Now, we want to show that

$$valuations[j] = \sum_{t\in[|S|];t\neq j} i_t v_p(\beta_t - \beta_j) + k_j v_p(i_j!)$$

holds true. We do this inductively. First we already have 0 stored in each entry of *valuations*. Let us assume that we have obtained a p-ordering upto length $\tilde{n}$ with the respective indices as $i_1, i_2 \dots i_{|S|}$ with the p-value corresponding to addition of next element from j^{th} representative root correctly stored in $valuations[j]$. Let the last element added to this p-ordering (in Step 23) correspond to the t^{th} representative root ($t = min\_index$) and then we change the *valuations* accordingly in Steps 24-28.

When we add this element, we increase i_t by one ($i'_t = i_t + 1$). Now when we add another element, say m, (after the last element from the t^{th} representative root), if $m \neq t$ then the new p-value will be

$$\sum_{l\in[|S|];l\notin\{t,m\}} i_l v_p(\beta_l - \beta_m) + (i_t + 1)v_p(\beta_t - \beta_m) + k_m v_p(i_m!),$$

which is $v_p(\beta_t - \beta_m)$ more than the previous $valuations[m]$. So accordingly we add this value in the previous step (when we find t as the min_index and then update in Steps 27-29).

However, if this m (the next min_index after adding an element from t^{th} representative root) is same as t, then the p-value will be

$$\sum_{l\in[|S|];l\neq t} i_l v_p(\beta_l - \beta_t) + k_t v_p((i_t + 1)!),$$

while the previous value of $valuations[t]$ was $\sum_{l\in[|S|];l\neq t} i_l v_p(\beta_l - \beta_t) + k_t v_p(i_t!)$ and this difference $v_p((i_t+1)!) - v_p(i_t!)$ is stored in p-Exponent_Increase(i_j). We thereby update Steps 25-26 of Algorithm 2 to incorporate this change. This shows that *valuations* correctly stores the p-value, as desired.

□

Proof of correctness of Algorithm 2. By the definition of p-ordering we know that at each iteration if we choose the element with the least valuation then we get a valid p-ordering. By Theorem 14, we know that *valuations* array has the correct next valuations. Hence, to find the representative root with the least valuation, we find the index of the minimum element in *valuations*.

To add the next value, by Observation 8, we find the next element from the p-ordering on the representative root. Hence, the element added has the least valuation. Hence, Find_p-Ordering(S, n) returns the correct p-ordering. □

4.2 Time Complexity

Theorem 15. *Given a set $S \subset \mathbb{Z}_{p^k}$, for a prime p and an integer k, that can be represented in terms of d representative roots, Algorithm 2 finds a p-ordering of length n for S in $\widetilde{\mathcal{O}}(d^2k\log p + nk\log p + np)$ time.*

Proof. Let S contains d representative roots of $\mathbb{Z}_{p^k}$. We want to find a p-ordering up to length n.

The function CORRELATE(S) runs a double loop, each of size d, and each iteration takes $\widetilde{\mathcal{O}}(k\log p)$, hence, CORRELATE($S$) takes $\widetilde{\mathcal{O}}(d^2k\log p)$.

p-EXPONENT_INCREASE(n) runs a single loop of size n where each iteration takes $\widetilde{\mathcal{O}}(k\log p)$ time, hence, it takes $\widetilde{\mathcal{O}}(nk\log p)$.

Then main loop (Step 20-29) runs a loop of size n, inside this loop we do $\mathcal{O}(d)$ operations on elements of size $\log k$, hence, it takes $\widetilde{\mathcal{O}}(nd)$ time.

Hence, in total, our algorithm takes

$$\widetilde{\mathcal{O}}(d^2k\log p + nk\log p + nd)$$

time. □

Now, the proof of Theorem 13 follows directly from the proof of correctness of Algorithm 2 and the time complexity analysis shown in Theorem 15.

5 Structure of Root Sets for a given k

We know that $\mathbb{Z}_{p^k}$ is not a unique factorization domain. In fact, even small degree polynomials can have exponentially large number of roots as seen in Section 2. Interestingly, not all subsets of $\mathbb{Z}_{p^k}$ can be a root set (Definition 4). If we know the general form of root sets, it can help us decide if a given set is a root set. In this section, we discuss and distinctly describe all the root sets in Z_{p^2}, Z_{p^3} and Z_{p^4}.

Dearden and Metzger [13] showed that R is a root-set iff $R_j = \{r \in R \mid r \equiv j \pmod p\}$ is also a root-set for all $j \in [p]$. For example, we know that $R = \{1, 4, 5, 7, 9, 10, 13, 14, 16, 19, 22, 23, 25\}$ is the root set in $\mathbb{Z}_{3^3}$ for the polynomial $f(x) = (x-1)^3(x-5)^2(x-9)$, then $R_0 = \{9\}$, $R_1 = \{1, 4, 7, 10, 13, 16, 19, 22, 25\}$, and $R_2 = \{5, 14, 23\}$ are also root sets.

The number and structure of R_j is symmetric for all j. Let N_{p^k} be the number of possible R_j's, then total number of possible root-sets become $(N_{p^k})^p$ [13].

Let $S_j = \{s \in \mathbb{Z}_{p^k} \mid s \equiv j \pmod p\}$, we take the following approach to find all possible root-sets R_j's. Given an R_j, define $R = \{(r-j)/p : r \in R_j\}$ to be the translated copy. We show that if R contains at least k many distinct residue classes mod$p$, then $R_j = S_j$ (Lemma 1). We exhaustively cover all the other cases, when R contains less than k residue classes (possible because k is

small). For example, in $\mathbb{Z}_{p^3}$, we find that

$$R_j = \begin{cases} j + p \cdot *, \\ (j + p \cdot \alpha_1 + p^2 *) \cup (j + p \cdot \alpha_2 + p^2 *), \text{ for } \alpha_1 \neq \alpha_2 \in [p], \\ j + p \cdot \alpha + p^2 *, \text{ for } \alpha \in [p], \\ j + p \cdot \alpha_1 + p^2 \cdot \alpha_2, \text{ for } \alpha_1, \alpha_2 \in [p], \\ \emptyset. \end{cases}$$

Using the following approach, we analyze the root sets modulo prime powers by assuming a general polynomial of arbitrarily large degree. We then take a small set $A \subset S_j$ (recall that $S_j = \{s \in \mathbb{Z}_{p^k} \mid s \equiv j \pmod{p}\}$), and find the smallest root-set R_j for which $A \subset R_j$. It is easy to see that we have a set A of size k for which the smallest root-set R_j containing A is the complete S_j (Lemma 1). Next, we iterate over smaller sets A to find smaller root-sets R_j. To iterate over multiple A's at a time, we generalise by looking at structures of A. We know that the number of possible structures of A is small, since, the size of A is less that k, which is small. Hence, we iterate over all possible structures of A to exhaust all possible structures of R_j. This gives the size of R_j, denoted by N_{p^k}. The total number of possible root-sets are $(N_{p^k})^p$ ([13]).

5.1 Root sets modulo p^2

A number $a \in \mathbb{Z}_{p^2}$ can be uniquely represented as $a = a_0 + a_1 \cdot p$ for some $a_0, a_1 \in [p]$. Let $f(x) = \sum_{i=0}^{\infty} b_i \cdot x^i$ be a polynomial, and $\alpha = \alpha_0 + \alpha_1 \cdot p \in \mathbb{Z}_{p^2}$ be a root of $f(x)$ in $\mathbb{Z}_{p^2}$, then by Taylor expansion of f

$$f(\alpha_0) + p \cdot \alpha_1 \cdot f'(\alpha_0) = 0 \bmod p^2.$$

Since all element of R_j are congruent to j mod p, fix α_0 to some j. We find that the root set R_j can only take the following structure.

1. **1** root-set is the complete sub-tree under j (more than 1 residue class), equivalently
$$R_j = j + p \cdot *.$$
2. $\boldsymbol{p}$ root-sets are a single element congruent to j mod p (1 residue class), equivalently
$$R_j = j + p \cdot \alpha, \text{ for } \alpha \in [p].$$
3. **1** root-set is empty (no residue classes), equivalently
$$R_j = \emptyset.$$

Hence, we have, $N_{p^2} = p + 2$.

5.2 Root sets modulo p^3

Similarly in $\mathbb{Z}_{p^3}$, we know that any number a can be uniquely represented as $a = a_0 + a_1 \cdot p + a_2 \cdot p^2$ for some $a_0, a_1, a_2 \in [p]$. Let $f(x) = \sum_{i=0}^{\infty} b_i \cdot x^i$ be a polynomial. Let some $\alpha = \alpha_0 + \alpha_1 \cdot p + \alpha_2 \cdot p^2 \in \mathbb{Z}_{p^3}$ be a root of $f(x)$ in $\mathbb{Z}_{p^3}$. Then we have,

$$f(\alpha_0) + p \cdot \alpha_1 \cdot f'(\alpha_0) + \left((\alpha_1)^2 \cdot \frac{f''(\alpha_0)}{2} + \alpha_2 \cdot f'(\alpha_0) \right) \cdot p^2 = 0 \bmod p^3.$$

Fixing α_0 to some j, we find that the root set can only take the following structure.

1. **1** root-set is the complete sub-tree, equivalently
$$R_j = j + p \cdot *.$$
2. $\boldsymbol{\frac{p(p-1)}{2}}$ root-sets are the union of 2 sub-trees different at the level p^1, equivalently
$$R_j = (j + p \cdot \alpha_1 + p^2 *) \cup (j + p \cdot \alpha_2 + p^2 *), \text{ for } \alpha_1 \neq \alpha_2 \in [p].$$
3. $\boldsymbol{p}$ root-sets are a sub-tree at the level p^1, equivalently
$$R_j = j + p \cdot \alpha + p^2 *, \text{ for } \alpha \in [p].$$
4. $\boldsymbol{p^2}$ root-sets are a single element congruent to j mod p, equivalently
$$R_j = j + p \cdot \alpha_1 + p^2 \cdot \alpha_2, \text{ for } \alpha_1, \alpha_2 \in [p].$$
5. **1** root-set is empty, equivalently
$$R_j = \emptyset.$$

Hence, we have, $N_{p^3} = \frac{3p^2+p+4}{2}$.

5.3 Root sets modulo p^4

Similar to $k = 2, 3$, the root sets R_j can only take the following structures.

1. **1** root-set is the complete sub-tree under j, equivalently
$$R_j = j + p \cdot *.$$
2. $\boldsymbol{\frac{p(p-1)(p-2)}{6}}$ root-sets under j are the union of 3 sub-trees different at the level p^1, equivalently
$R_j = (j+p\cdot\alpha_1+p^2*) \cup (j+p\cdot\alpha_2+p^2*) \cup (j+p\cdot\alpha_3+p^2*)$, for $\alpha_1 \neq \alpha_2 \neq \alpha_3 \in [p]$.
3. $\boldsymbol{\frac{p(p-1)}{2}}$ root-sets are the union of 2 sub-trees different at the level p^1, equivalently
$$R_j = (j + p \cdot \alpha_1 + p^2 *) \cup (j + p \cdot \alpha_2 + p^2 *), \text{ for } \alpha_1 \neq \alpha_2 \in [p].$$
4. $\boldsymbol{p}$ root-sets are a sub-tree at the level p^1, equivalently
$$R_j = j + p \cdot \alpha + p^2 *, \text{ for } \alpha \in [p].$$

5. $\frac{\boldsymbol{p^3(p-1)}}{\boldsymbol{2}}$ root-sets are a union of 2 sub-trees at the level p^2 that are different at the level p^1, equivalently
$R_j = (j+p\cdot\alpha_1+p^2\cdot\beta_1+p^3*)\cup(j+p\cdot\alpha_2+p^2\cdot\beta_2+p^3*)$, for $\alpha_1 \neq \alpha_2\, \beta_1, \beta_2 \in [p]$.
6. $\boldsymbol{p^2}$ root-sets are a sub-tree at the level p^2, equivalently
$$R_j = j + p \cdot \alpha_1 + p^2 \cdot \alpha_2 + p^3 \cdot *, \text{ for } \alpha_1, \alpha_2 \in [p].$$
7. $\boldsymbol{p^3}$ root-sets are a single element congruent to $j \bmod p$, equivalently
$$R_j = j + p \cdot \alpha_1 + p^2 \cdot \alpha_2 + p^3 \cdot \alpha_3, \text{ for } \alpha_1, \alpha_2, \alpha_3 \in [p].$$
8. $\boldsymbol{1}$ root-set is empty, equivalently
$$R_j = \emptyset.$$

Hence, we have, $N_{p^4} = \frac{3p^4+4p^3+6p^2+5p+12}{6}$.

Acknowledgements: We would like to thank Naman Jain for helpful discussions. R.M. would like to thank Department of Science and Technology, India for support through grant DST/INSPIRE/04/2014/001799.

References

1. Adleman, L., Lenstra, H.: Finding irreducible polynomials over finite fields. In: Proc. 18th Annual ACM Symp. on Theory of Computing (STOC), 350 - 355 (1986). pp. 350–355 (11 1986). https://doi.org/10.1145/12130.12166
2. Agrawal, M., Kayal, N., Saxena, N.: Primes is in p. Annals of mathematics pp. 781–793 (2004)
3. Berlekamp, E.: Factoring polynomials over large finite fields. Mathematics of Computation **24**, 713–735 (07 1970). https://doi.org/10.1090/S0025-5718-1970-0276200-X
4. Berthomieu, J., Lecerf, G., Quintin, G.: Polynomial root finding over local rings and application to error correcting codes. Applicable Algebra in Engineering, Communication and Computing **24**(6), 413–443 (2013)
5. Bhargava, M.: P-orderings and polynomial functions on arbitrary subsets of dedekind rings. Journal Fur Die Reine Und Angewandte Mathematik - J REINE ANGEW MATH **1997**, 101–128 (01 1997). https://doi.org/10.1515/crll.1997.490.101
6. Bhargava, M.: The factorial function and generalizations. American Mathematical Monthly **107** (11 2000). https://doi.org/10.2307/2695734
7. Bhargava, M.: On p-orderings, rings of integer values functions, and ultrametric analysis. Journal of the American Mathematical Society **22**(4), 963–993 (2009)
8. Bose, R., Ray-Chaudhuri, D.: On a class of error correcting binary group codes *. Information and Control **3**, 68–79 (03 1960). https://doi.org/10.1016/S0019-9958(60)90287-4
9. Cantor, D., Zassenhaus, H.: A new algorithm for factoring polynomials over finite fields. Mathematics of Computation **36** (04 1981). https://doi.org/10.2307/2007663
10. Cheng, Q., Gao, S., Rojas, J.M., Wan, D.: Counting roots for polynomials modulo prime powers. The Open Book Series **2**(1), 191–205 (2019)

11. Chor, B., Rivest, R.: A knapsack type public key cryptosystem based on arithmetic in finite fields. IEEE Transactions on Information Theory **34** (09 2001). https://doi.org/10.1109/18.21214
12. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge, MA (2001)
13. Dearden, B., Metzger, J.: Roots of polynomials modulo prime powers. Eur. J. Comb. **18**, 601–606 (08 1997). https://doi.org/10.1006/eujc.1996.0124
14. Dwivedi, A., Mittal, R., Saxena, N.: Efficiently factoring polynomials modulo p^4. International Symposium on Symbolic and Algebraic Computation (ISSAC) pp. 139–146 (07 2019). https://doi.org/10.1145/3326229.3326233
15. Hocquenghem, A.: Codes correcteurs d'erreurs. Chiffres, Revue de l'Association Française de Calcul **2** (01 1959)
16. Johnson, K.: P-orderings of finite subsets of dedekind domains. Journal of Algebraic Combinatorics **30**, 233–253 (2009)
17. Lenstra, A., Lenstra, H., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Annalen **261** (12 1982). https://doi.org/10.1007/BF01457454
18. Lenstra, H.: On the chor—rivest knapsack cryptosystem. Journal of Cryptology **3**, 149–155 (01 1991). https://doi.org/10.1007/BF00196908
19. Lidl, R., Niederreiter, H.: Finite fields, vol. 20. Cambridge university press (1997)
20. Maulik, D.: Root sets of polynomials modulo prime powers. J. Comb. Theory, Ser. A **93**, 125–140 (01 2001). https://doi.org/10.1006/jcta.2000.3069
21. Odlyzko, A.: Discrete logarithms and their cryptographic significance. Advances in Cryptography, EUROCRYPT '84, Proceedings, Lecture Notes in Computer Science **209**, 224–314 (1985)
22. Panayi, P.N.: Computation of Leopoldt's P-adic regulator. Ph.D. thesis, University of East Anglia (1995)
23. Reed, I., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics **8**, 300–304 (06 1960). https://doi.org/10.2307/2098968
24. Sudan, M.: Decoding reed solomon codes beyond the error-correction bound. Journal of Complexity **13**, 180–193 (03 1997). https://doi.org/10.1006/jcom.1997.0439
25. Zassenhaus, H.: On hensel factorization ii. Journal of Number Theory **1**, 291–311 (07 1969). https://doi.org/10.1016/0022-314X(69)90047-X