On algorithms to find p-ordering

Aditya Gulati, Sayak Chakrabarti, and Rajat Mittal

IIT Kanpur

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 set of size n in time $\widetilde{O}(nk \log p)$, where set is considered modulo p^k. The subsets modulo p^k can be represented succinctly using the notion of representative roots (Panayi, PhD Thesis, 1995; Dwivedi et.al, ISSAC, 2019); a natural question would be, 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 $\widetilde{O}(d^2k \log p + nk \log p + nd)$, where d is the size of the succinct representation and n is the required length of the p-ordering. Another contribution that we make 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 in the previous work (Dearden and Metzger, Eur. J. Comb., 1997; Maulick, J. Comb. Theory, Ser. A, 2001), we explicitly describe all the root sets for p^2, p^3 and p^4.

Keywords: root-sets · computational complexity · 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 [BRC60,Hoc59,RS60,Sud97], cryptography [CR01,Odl85,Len91], computational number theory [AL86,AKS04] and computer algebra [Zas69,LLL82]. 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 [LN97]. Not just that, we also have efficient randomized algorithms for the problem of factorizing polynomials over finite fields [CZ81,Ber70].

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 $\mathbb{F}_p$), 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 [Mau01] 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 [BLQ13] and a deterministic algorithm for root counting [DMS19,CGRW19].

The concept of *p-ordering* and *p-sequences* for a prime p, introduced by Bhargava [Bha97], is an important tool in studying the properties of roots sets and polynomials over powers of prime p [Mau01,Bha97,Bha09]. 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 [Bha00].

For polynomials, p-ordering allows us to give a convenient basis for representing polynomials on a subset of ring $\mathbb{Z}_{p^k}$. One of the interesting problem for polynomials over rings, of the kind $\mathbb{Z}_{p^k}$, is to find the allowed *root sets* (Definition 3). Maulik [Mau01] 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 [Mau01,Joh09,Bha09], 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, given in Bhargava [Bha97], gives a $\widetilde{O}(n^3 k \log(p))$ 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 $\widetilde{O}(nk \log(p))$ 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* [Pan95,DMS19,BLQ13] giving a succinct representation. 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 $\widetilde{O}(d^2 k \log p + nk \log p + nd)$ 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. But 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 [DM97]. Building on previous work, Maulik [Mau01] 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 $k \leq 4$. In Section 5, we discuss and distinctly describe all the root sets in $\mathbb{Z}_{p^2}$, $\mathbb{Z}_{p^3}$ and $\mathbb{Z}_{p^4}$.

2 Preliminaries

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 $\widetilde{O}$ instead of O in time complexity. We also use $[n]$ for the set $\{0, 1, \ldots, n-1\}$.

We begin by defining the valuation of an integer modulo a prime p.

Definition 1. *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 ∞.

Definition 2. *For any ring S with the usual operations $+$ and $*$, we have*

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

Definition 3. *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.*

Representative Roots: The notion of representative roots in the ring $\mathbb{Z}_{p^k}$ has been used to concisely represent roots of a polynomial [Pan95,DMS19,BLQ13].

Definition 4. *The representative root $(a + p^i *)$ is a subset of $\mathbb{Z}_{p^k}$,*

$$a + p^i * := \{a + p^i y \mid y \in \mathbb{Z}_{p^{k-i-1}}\}$$

Extending, a set $S = \{r_1, \cdots, r_l\}$ of representative roots correspond to $\bigcup_{i=1}^{l} r_i \subseteq \mathbb{Z}_{p^k}$. Conversely, we show that an $S \subseteq \mathbb{Z}_{p^k}$ can be uniquely represented by representative roots.

Definition 5. *Let $S \subseteq \mathbb{Z}_{p^k}$, then the set of representative roots $S^{rep} = \{r_1 = \beta_1 + p^{k_1} *, r_2 = \beta_2 + p^{k_2} *, \ldots, r_l = \beta_l + p^{k_l} *\}$ 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. $\forall i : \bigcup_{b \in [p]} \left(r_i + p^{k_i - 1} \cdot b\right) \not\subseteq S$

Theorem 1. *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 8, $r \cap \widehat{r} \neq \emptyset$ 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 $\widehat{S^{rep}}$.

■

p-ordering and p-sequence Bhargava [Bha97] introduced the concept of p-ordering for any subset of a Dedekind domain, we restrict to the rings of the form $\mathbb{Z}_{p^k}$ [Bha00].

Definition 6 ([Bha97]). *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

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

. 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.

The p-ordering on a subset of $\mathbb{Z}$ can be defined similarly. Bhargava surprisingly proved the following theorem.

Theorem 2 ([Bha97]). *For any two p-orderings of a subset $S \subseteq \mathbb{Z}$ and a prime p, the associated p-sequences are same.*

Few observations about p-orderings/p-sequences/ representative roots are listed in Appendix B. We also use a min-heap data structure to optimize our algorithm, details of min-heap are given in Appendix C.

3 Algorithm to find p-ordering on a given set

The naive algorithm for finding the p-ordering, from its definition, has time complexity $\widetilde{O}(n^3 k \log(p))$ (Appendix A). The main result of this section is the following theorem.

Theorem 3. *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 $\widetilde{O}(nk \log p)$ time.*

The proof of the theorem follows by constructing an algorithm to find the p-ordering.

Outline of the algorithm We use the recursive structure of p-ordering given by Maulik [Mau01]. Crucially, to find the p-value of an element a at each step, we only need to look at elements congruent mod$p$ to a (Observation 10).

Suppose S_j is the set of elements of S congruent to j mod 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 translated S_j's is computed recursively (Observation 11).

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 p number of elements taking $\widetilde{O}(p)$ time. Instead, we use min heap data structure, using only $\widetilde{O}(\log p)$ time for extraction and insertion of elements.

Each node of the min-heap(H) consists of the following values,

1. p_value: contains p-value of the element when added to p-ordering,
2. set: contains the index of the set $S_0, S_1, ..., S_{p-1}$ element belongs to, and
3. $value$: contains the value of the element.

These values are used to preserve the properties of the data structures used. With above intuition in mind, we develop Algorithm 1 to find the p-ordering on a subset of $\mathbb{Z}$.

3.1 Proof of Theorem 3

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

Theorem 4 (Correctness of MERGE()). *In Algorithm 1, given S be a subset of integers, let for* $k \in \{0, 1, ..., p-1\}$, $S_k = \{s \in S \mid s \equiv k \pmod{p}\}$, *then given a p-ordering on each of the* S_k*'s,* MERGE$(S_0, S_1, ..., S_{p-1})$ *gives a valid p-ordering on S.*

Proof outline. We start with p-orderings on each of the non-empty sets $(S_0, S_1, ..., S_{p-1})$, and create a heap taking the first element of each of these p-ordering. At each successive step, we pick the element in the heap with minimum p-value to add to the p-ordering, and insert the next element from the corresponding S_j to the heap.

We know that the valuation of any element in the combined p-ordering is going to be equal to their valuation in the p-ordering over the set S_j containing them (by Observation 10). If we show that at each step the element chosen has the least valuation out of all the elements left MERGE() works correctly. We prove this by getting a contradiction if any element other than the ones obtained from the min heap is selected by showing the p_value will be greater than what we get from MERGE(). ■

The details of the proof can be found in Appendix D.1.

Theorem 5 (Correctness of valuations). *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.*

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($node = \{S_0[i_0], S_1[i_1], ..., S_{p-1}[i_{p-1}]\}, key = p\_value$)
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: $S[0].p\_value \leftarrow 1$
19: **Return** S
20: $S_0, S_1, ..., S_{p-1} \leftarrow ([\,], [\,], ..., [\,])$
21: **for** $i \in S$ **do**
22: $S_{i.value \bmod p}.append(i)$
23: **for** $i \in [0, 1, ..., p-1]$ **do**
24: $S_i \leftarrow$ FIND_p-ORDERING($(S_i - i)/p$)
25: **for** $j \in [0, ..., len(S_i) - 1]$ **do**
26: $S_i[j].value \leftarrow p * S_i[j].value + i$
27: $S_i[j].p\_value \leftarrow S_i[j].p\_value + j$
28: $S \leftarrow$ MERGE($S_0, S_1, ..., S_{p-1}$)
29: **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 21-22.

Proof outline. The proof requires two parts: MERGE() preserves valuation and changes in the valuation due to *translation* does not induce errors.

- To prove that MERGE() preserves valuation, we make use of the fact that the combined p-ordering after merge has the individual p-orderings as a sub-sequence. Hence, the valuation of each element in the combined p-ordering is going to be equal to the valuation in the individual p-ordering (by Observation 10). Hence, MERGE() preserves valuations.
- We show that the change in valuations due to translation (Step 24) are corrected (Step 27). This is easy to show by just updating the valuation according to Observation 12.

Hence, valuations are correct maintained throughout the algorithm. ■

The details of the proof can be found in Appendix D.2.

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

Proof of Theorem 3. For the base case, if S is a singleton, then the p-ordering over it is just a single element which is also what FIND_p-ORDERING(S) gives. Let FIND_p-ORDERING() works for $|S| < k$, if we show it works for $|S| = k$, then by induction, FIND_p-ORDERING() works for sets of arbitrary sizes.

Let $|S| = k$, then when we break the set into $S_0, S_1, ..., S_{p-1}$ (Steps 20-22), either all element belong in a single S_i or get distributed into multiple sets. We can argue that if all the elements fall into the same group, then when we keep calling recursion (Step 24), after some point set breaks into multiple S_i's. Since, by Observation 11, we know that the p-ordering on reduced elements is preserved, we'll get the correct p-ordering on the original set. Hence, we only need to prove this for the later case.

Since all the element of the set S_i follow $\forall y \in S_i$, $y \equiv i \bmod p$, hence $\forall y \in S_i$, $p \mid (y - i)$, this implies $(S_i - i)/p \subset \mathbb{Z}$. Hence, FIND_$p$-ORDERING($(S_i - i)/p$) gives a p-ordering on $(S_i - i)/p$ with the correct valuations associated with each element (Theorem 5).

From Observation 11, we know that if $(a_0, a_1, ...)$ be a p-ordering on some set A, then $(p*a_0+i, p*a_1+i, ...)$ be a p-ordering on $p*A+i$. Since, FIND_p-ORDERING($(S_i-i)/p$) is a p-ordering on $(S_i-i)/p$, then $p*$FIND_p-ORDERING($(S_i - i)/p$)$+x$ is a p-ordering on S_i (Step 26).

Next, since we have valid p-orderings on $S_0, S_1, ..., S_{p-1}$, MERGE($S_0, S_1, ..., S_{p-1}$) returns a valid p-ordering on S (Theorem 4).

By induction, our algorithm returns a valid p-ordering on any subset of integers.

If each element of S is less than p^k, then p-ordering on set S requires $\widetilde{O}(nk \log p)$ time (Theorem 14 of Appendix D.3). ■

4 Algorithm to find p-ordering on a set of representative roots

The notion of representative roots (Definition 2) 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 5), our subset is represented in a unique way (Theorem 1). A natural question arises, can we efficiently find a p-ordering given a set in terms of representative roots?

In this section we show that the answer is affirmative by constructing an efficient algorithm in terms of the size of the succinct representation.

Theorem 6. *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 $\widetilde{O}(d^2k \log p + nk \log p + np)$ time.*

The proof of the theorem follows by constructing an algorithm to find the p-ordering given a set in representative root notation. We can assume that the representative roots are disjoint. If they are not, one representative root will be contained in another (Observation 8), all such occurrences can be deleted in $\widetilde{O}(d^2k \log p)$ time.

Outline of the algorithm The important observation is, we already have a natural p-ordering defined on a representative root (Observation 13). 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 effect the p-value of an element outside S_j. However, in this case the exact increase in the p-value is known by Observation 9.

Let d be the number of representative roots, we maintain an array of size d to keep 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 $\widetilde{O}(\mathrm{d})$) and add it to the combined p-ordering (in $\widetilde{O}(1)$) and we update all the d values in this array (in $\widetilde{O}(\mathrm{d})$). 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 our algorithm, we first prove that valuations are correctly maintained.

Theorem 7. *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 outline. All elements have 0 valuation at the beginning (Step 17). Also, adding an element from the i-th representative root increases the valuation of the j-th representative root by $corr(i, j)$ (Step 33) for $i \neq j$ (Observation 9). The increase for the next element of i is exponent times the increase in p-sequence of $\mathbb{Z}_{p^k}$ (Step 30) (Observation 13). So, we correctly update the valuations array in each iteration. ■

A detailed proof can be found in Appendix E.1.

Proof of Theorem 6. 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 7, we know that $valuations$ array has the correct next valuations. Hence, to find the representative root with gives the least valuation, we find the index of the minimum element in $valuations$.

To add the next value to the p-ordering, we use Observation 13 to find the next element in 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.

If S contains d representative roots of $\mathbb{Z}_{p^k}$, then Algorithm 2 finds p-ordering on S up to length n in $\widetilde{O}(d^2k \log p + nk \log p + nd)$ time (Theorem 15 of Appendix E.2). ■

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. Interestingly, not all subsets

Algorithm 2 Find p-ordering from minimal notation

```
1:  procedure CORRELATE(S)
2:      Corr ← [0]_{len(S)×len(S)}
3:      Corr ← [0]_{len(S)×len(S)}
4:      for j ∈ [1, ..., len(S)] do
5:          for k ∈ [1, ..., len(S)] do
6:              Corr[j][k] ← v_p(S[j].value − S[k].value)
7:      return Corr
8:  procedure p-EXPONENT_INCREASE(n)
9:      v_p(1) ← 1
10:     for j ∈ [1, ..., n] do
11:         v_p((j + 1)!) ← v_p(j + 1) ∗ v_p(j!)
12:         p_exponent[j] ← v_p((j + 1)!) − v_p(j!)
13:     return p_exponent
14: procedure FIND_p-ORDERING(S, n)
15:     corr ← CORRELATE(S)
16:     increase ← p-EXPONENT_INCREASE(n)
17:     valuations ← [0]_{|S|}
18:     p_ordering ← {}
19:     i_1, i_2 ... i_{|S|} ← 0
20:     for i ∈ {1, 2, ... n} do
21:         min ← ∞
22:         min_index ← ∞
23:         for j ∈ [1, ..., len(S)] do
24:             if valuations[j] < min then
25:                 min ← valuations[j]
26:                 min_index ← j
27:         p_ordering.append(S[min_index].value + p^{S[min_index].exponent} ∗ i_{min_index})
28:         for j ∈ [1, ..., len(S)] do
29:             if j = min_index then
30:                 valuations[j] ← valuations[j] + S[min_index].exponent ∗ increase[i_j]
31:             else
32:                 valuations[j] ← valuations[j] + corr(min_index, j)
33:         i_{min_index} ← i_{min_index} + 1
34:     return p_ordering
```

of $\mathbb{Z}_{p^k}$ can be a root set (Definition 3). Dearden and Metzger [DM97] 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 $j \in [p]$. The size 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 $(N_{p^k})^p$ [DM97]. In this section, we discuss and describe all possible R_j's in Z_{p^2}, Z_{p^3} and Z_{p^4}.

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 $R_j = S_j$ (Observation 16). We exhaustively cover all the other cases, when R contains less than k residue classes (possible because k is small).

5.1 $k = 2$

We find that the root set R_j can only take the following structures (details in Appendix F.1).

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 $j \bmod p$ (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, total root-sets, $N_{p^2} = p + 2$.

5.2 $k = 3$

Similar to $k = 2$, the root sets R_j can only take the following structure (details in Appendix F.2).

1. **1** root-set is the complete sub-tree, equivalently
$$R_j = j + p \cdot *.$$
2. $\frac{\boldsymbol{p(p-1)}}{\boldsymbol{2}}$ 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 $j \bmod p$, 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, total root-sets, $N_{p^3} = \frac{3p^2+p+4}{2}$.

5.3 $k = 4$

Similar to $k = 2, 3$, the root sets R_j can only take the following structure (details in Appendix F.3).

1. **1** root-set is the complete sub-tree under j, equivalently
$$R_j = j + p \cdot *.$$
2. $\frac{\boldsymbol{p(p-1)(p-2)}}{\boldsymbol{6}}$ 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. $\frac{\boldsymbol{p(p-1)}}{\boldsymbol{2}}$ 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 $j \mod p$, 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. $\mathbf{1}$ root-set is empty, equivalently
$$R_j = \emptyset.$$

Hence, total root-sets, $N_{p^4} = \frac{3p^4+4p^3+6p^2+5p+12}{6}$.

References

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004.

[AL86] Leonard Adleman and Hendrik Lenstra. Finding irreducible polynomials over finite fields. In *Proc. 18th Annual ACM Symp. on Theory of Computing (STOC), 350 - 355 (1986)*, pages 350–355, 11 1986.

[Ber70] E.R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24:713–735, 07 1970.

[Bha97] Manjul Bhargava. 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.

[Bha00] Manjul Bhargava. The factorial function and generalizations. *American Mathematical Monthly*, 107, 11 2000.

[Bha09] Manjul Bhargava. On p-orderings, rings of integer values functions, and ultrametric analysis. *Journal of the American Mathematical Society*, 22(4):963–993, 2009.

[BLQ13] Jérémy Berthomieu, Grégoire Lecerf, and Guillaume Quintin. Polynomial root finding over local rings and application to error correcting codes. *Applicable Algebra in Engineering, Communication and Computing*, 24(6):413–443, 2013.

[BRC60] R.C. Bose and D.K. Ray-Chaudhuri. On a class of error correcting binary group codes *. *Information and Control*, 3:68–79, 03 1960.

[CGRW19] Qi Cheng, Shuhong Gao, J Maurice Rojas, and Daqing Wan. Counting roots for polynomials modulo prime powers. *The Open Book Series*, 2(1):191–205, 2019.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001.

[CR01] Benny Chor and Ronald Rivest. A knapsack type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34, 09 2001.

[CZ81] David Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36, 04 1981.

[DM97] Bruce Dearden and Jerry Metzger. Roots of polynomials modulo prime powers. *Eur. J. Comb.*, 18:601–606, 08 1997.

[DMS19] Ashish Dwivedi, Rajat Mittal, and Nitin Saxena. Efficiently factoring polynomials modulo p^4. *International Symposium on Symbolic and Algebraic Computation*, pages 139–146, 07 2019.

[Hoc59] A. Hocquenghem. Codes correcteurs d'erreurs. *Chiffres, Revue de l'Association Française de Calcul*, 2, 01 1959.

[Joh09] Keith Johnson. P-orderings of finite subsets of dedekind domains. *Journal of Algebraic Combinatorics*, 30:233–253, 2009.

[Len91] H. Lenstra. On the chor—rivest knapsack cryptosystem. *Journal of Cryptology*, 3:149–155, 01 1991.

[LLL82] Arjen Lenstra, H. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261, 12 1982.

[LN97] Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20. Cambridge university press, 1997.

[Mau01] Davesh Maulik. Root sets of polynomials modulo prime powers. *J. Comb. Theory, Ser. A*, 93:125–140, 01 2001.

[Odl85] A. Odlyzko. Discrete logarithms and their cryptographic significance. *Advances in Cryptography, EUROCRYPT '84, Proceedings, Lecture Notes in Computer Science*, 209:224–314, 1985.

[Pan95] Peter N Panayi. *Computation of Leopoldt's P-adic regulator.* PhD thesis, University of East Anglia, 1995.

[RS60] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8:300–304, 06 1960.

[Sud97] Madhu Sudan. Decoding reed solomon codes beyond the error-correction bound. *Journal of Complexity*, 13:180–193, 03 1997.

[Zas69] Hans Zassenhaus. On hensel factorization ii. *Journal of Number Theory*, 1:291–311, 07 1969.

Appendix A Naive Algorithm to find p-ordering

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 6. After we have already chosen $\{a_0, a_1, \dots a_{t-1}\}$ we choose the next element from $S\backslash\{a_0, a_1, \dots a_{t-1}\}$ such that $v_p((x-a_0)(x-a_1)\dots(x-a_{t-1}))$ is minimum. The naive approach given in [Bha97] iterates over all x in $S\backslash\{a_0, a_1, \dots a_{t-1}\}$ and adds the element to the p-ordering which gives the minimum valuation.

Time Complexity: Every time we keep on adding another element to the already existing p-ordering, say of length t. For any given value of $x \in S\backslash\{a_0, a_1, \dots a_{t-1}\}$, calculating $x - a_i$ and multiplying for every $0 \geq i < t$ takes $O((n-t)t)$ operations in $\mathbb{Z}$ and since each of them are less than p^k this takes $O((n-t)tk \log p) \leq \mathbb{O}(n^2 k \log p)$. So repeating this n times gives us the time complexity $O(n^3 k \log p)$.

Appendix B Observations about representative roots and p-sequences/p-orderings

B.1 Representative roots

Observation 8. *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 $A_1 \cap A_2 = \tilde{A} \neq \emptyset$, then we show that $\tilde{A} = A_1$ or $\tilde{A} = A_2$.

Case 1: Let $k_1 = k_2 = x$. Let there is some element $a \in \tilde{A}$. Then $a \in A_1$ and $a \in A_2$, hence, A_1 can be defined as $A_1 = a + p^x*$ and similarly $A_2 = a + p^x*$. Hence, $\tilde{A} = A_1 = A_2$.

Case 2: Let $k_1 \neq k_2$, then without loss of generality, let's assume that $k_1 < k_2$. Let there is some element $a \in \tilde{A}$. 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. Now, we know that the elements of A_1 are of the form $a + p^{k_1}*$. Hence, putting $* = p^{k_2-k_1} y$, we get $a + p^{k_1} \cdot (p^{k_2-k_1} y) = b$, hence $b \in A_1$. Hence, $A_2 \subset A_1$. Hence, $\tilde{A} = A_2$. ■

Observation 9. *Let $a_1 \in \beta_1 + p^{k_1}*$ and $a_2 \in \beta_2 + p^{k_2}*$ be any 2 elements of the representative roots $\beta_1 + p^{k_1}*$ and $\beta_2 + p^{k_2}*$ respectively, for $\beta \neq \alpha_2$, then,*

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

Proof. We have 2 representative roots of the form $\beta_1 + p^{k_1}*$ and $\beta_2 + p^{k_2}*$. WLOG let us assume that $k_1 \leq k_2$ and $\beta_1 \in \mathbb{Z}_{p^{k_1}}, \beta_2 \in \mathbb{Z}_{p^{k_2}}$.
We definitely have that these two are different representative roots. So if the first k_1 elements of the p-adic expansion of β_2 are equal then the second representative root will

be contained in the first, as the $*$ portion of the first contains all the values of the second representative root as well as its subset. So for them to be different representative roots, $p^{k_1} \nmid \beta_2 - \beta_1$. Let $v_p(\beta_1 - \beta_2) = t$, $t < k_1$, then for any value y_1, y_2 in the respective $*$ sets, we will have $p^t | (\beta_1 + p^{k_1} y_1) - (\beta_2 + p^{k_2} y_2)$. Note that since $k_1 > v_p(\beta_1 - \beta_2)$ we have $v_p((\beta_1 + p^{k_1} y_1) - (\beta_2 + p^{k_2} y_2)) \geq v_p(\beta_1 - \beta_2)$.
Now, since $p^t | \beta_1 - \beta_2$ and $p^{t+1} \nmid \beta_1 - \beta_2$ we can write $\beta_1 - \beta_2 = p^t(a + pb)$ for $a, b \in \mathbb{Z}_{p^k}$ where $a \in \{1, 2, \ldots p-1\}$. This implies that if $p^{t+1} | (\beta_1 + p^{k_1} y_1) - (\beta_2 + p^{k_2} y_2)$, it means $p^{t+1} | p^t(a + pb) + p^{k_1}(y_1 - p^{k_2 - k_2} y_2) \implies p | a + p(\cdots)$ which can not be true as $p \nmid a$. So for any value of y_1, y_2 in the respective $*$ sets of their corresponding representative roots, we will have $v_p((\beta_1 + p^{k_1} y_1) - (\beta_2 + p^{k_2} y_2)) = v_p(\beta_1 - \beta_2)$.
Conversely if $\exists y_1, y_2$ such that $v_p((\beta_1 + p^{k_1} y_1) - (\beta_2 + p^{k_2} y_2)) > v_p(\beta_1 - \beta_2)$, let $l = v_p((\beta_1 + p^{k_1} y_1) - (\beta_2 + p^{k_2} y_2)) \geq k_1$. Then we have $v_p(\beta_1 - \beta_2) \leq l - 1$. So if $p^l | (\beta_1 + p^{k_1} y_1) - (\beta_2 + p^{k_2} y_2) \implies p^l | \beta_1 - \beta_2$ (as $l \leq k_1$). This is a contradiction as $l > v_p(\beta_1 - \beta_2)$.
This completes the proof of Observation 9. ■

B.2 *p*-ordering and *p*-sequence

Observation 10 ([Mau01])**.** *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. $x \equiv j \pmod p$,*

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

Observation 11. *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 12. *Let S be a subset of integers, let $(a_0, a_1, a_2, ...)$ be a p-ordering on S. Then, for any $x \in \mathbb{Z}$*

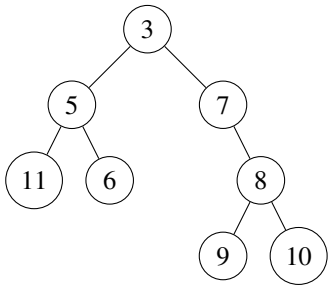
1. $v_p(x * S, k) = v_p(S, k) + k \cdot w_p(x)$.
2. $v_p(S + x, k) = v_p(S, k)$.

Observation 13. *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 11 and the fact that $1, 2, 3, \ldots$ form an obvious p-ordering in $\mathbb{Z}_{p^k}$. ■

Appendix C Min-heap data structure

A min-heap is a data structure 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 [CLRS01].

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

Appendix D Correctness and Complexity of Algorithm 1

D.1 Proof of Theorem 4

Let $(a_0^k, a_1^k, ...)$ be a p-ordering on each of the S_k's. We know that MERGE() on $(S_0, S_1, ..., S_{p-1})$ proceeds in such a way that elements are added to the heap in such a way that the element a_l^k is added to the heap only after $\forall i < l, a_i^k$ have already been added to the p-ordering. Also, we know at any point, only one element from any S_k can belong to the heap.

We know that at any point, let $(a_0, a_1, ...a_{k-1})$ be a p-ordering on S, then the next element a_k in the p-ordering on S if and only if

$$w_p\left(\prod_{i\in\{0,1,...,k-1\}}(a_k - a_i)\right) = \min_{x\in S\setminus\{a_0,a_1,...a_{k-1}\}}\left(w_p\left(\prod_{i\in\{0,1,...,k-1\}}(x - a_i)\right)\right).$$

We know that if at each point, our pick for the next element in the p-ordering satisfies this condition, the p-ordering we get is valid.

Lets say $(a_0, a_1, ..., a_i)$ be the p-ordering we have till now. Let elements currently the elements $(a_0^k, a_1^k, ..., a_{i_k-1}^k)$ of the given p-ordering on set S_k are currently a part of the p-ordering. Let the elements $(a_{i_0}^0, a_{i_1}^1, ..., a_{i_{p-1}}^{p-1})$ are a part of the min-heap.

Let when we extract the min from the min-heap, we get some value $a_{i_k}^k$. If we show that

$$w_p\left(\prod_{j\in\{0,1,...,i\}}(a_{i_k}^k - a_j)\right) = \min_{x\in S\setminus\{a_0,a_1,...a_i\}}\left(w_p\left(\prod_{j\in\{0,1,...,i\}}(x - a_j)\right)\right),$$

then we know that $a_{i_k}^k$ is a valid next element in the p-ordering and hence MERGE() gives a correct p-ordering on S.

We prove this by contradiction. Let there is an element $x \in S \setminus \{a_0, a_1, .., a_i, a_{i_k}^k\}$ such that

$$w_p\left(\prod_{j\in\{0,1,...,i\}}(x-a_j)\right) < w_p\left(\prod_{j\in\{0,1,...,i\}}(a_{i_k}^k - a_j)\right).$$

Then, we have 2 cases, either $x \in S_k$, i.e. $x \equiv k \bmod p \equiv a_{i_k}^k \bmod p$ or $x \in S_l$ for some $l \neq k$, i.e. $x \equiv l \bmod p \not\equiv a_{i_k}^k \bmod p$.

Case 1: $x \in S_k$.

Our assumption is that

$$w_p\left(\prod_{j\in\{0,1,...,i\}}(x-a_j)\right) < w_p\left(\prod_{j\in\{0,1,...,i\}}(a_{i_k}^k - a_j)\right).$$

We know that from our assumption that $S_k \cap (a_0, a_1, ..., a_i) = (a_0^k, a_1^k, ..., a_{i_k-1}^k)$. From Observation 10,

$$w_p\left(\prod_{j\in\{0,1,...,i\}}(a_{i_k}^k - a_j)\right) = w_p\left(\prod_{j\in\{0,1,...,i_k-1\}}(a_{i_k}^k - a_j^k)\right),$$

and

$$w_p\left(\prod_{j\in\{0,1,...,i\}}(x-a_j)\right) = w_p\left(\prod_{j\in\{0,1,...,i_k-1\}}(x - a_j^k)\right).$$

Since, $(a_0^k, a_1^k, ..., a_{i_k-1}^k, a_{i_k}^k, ...)$ is a valid p-ordering on S_k,

$$w_p\left(\prod_{j\in\{0,1,...,i\}}(a_{i_k}^k - a_j)\right) \leq w_p\left(\prod_{j\in\{0,1,...,i\}}(x-a_j)\right).$$

But this is a contradiction.

Case 2: $x \in S_l$ **for some** $l \neq k$.

Our assumption is that

$$w_p\left(\prod_{j\in\{0,1,...,i\}}(x-a_j)\right) < w_p\left(\prod_{j\in\{0,1,...,i\}}(a_{i_k}^k - a_j)\right).$$

Let the element belonging to the set S_l in the heap is $a_{i_l}^l$. We know that from our assumption that $S_l \cap (a_0, a_1, ..., a_i) = (a_0^l, a_1^l, ..., a_{i_l-1}^l)$. From Observation 10,

$$w_p\left(\prod_{j\in\{0,1,...,i\}}(a_{i_l}^l - a_j)\right) = w_p\left(\prod_{j\in\{0,1,...,i_l-1\}}(a_{i_l}^l - a_j^l)\right),$$

and

$$w_p\left(\prod_{j\in\{0,1,\ldots,i\}}(x-a_j)\right)=w_p\left(\prod_{j\in\{0,1,\ldots,i_l-1\}}(x-a_j^l)\right).$$

Since, $(a_0^l, a_1^l, ..., a_{i_l-1}^l, a_{i_l}^l, ...)$ is a valid p-ordering on S_l,

$$w_p\left(\prod_{j\in\{0,1,\ldots,i\}}(a_{i_l}^l-a_j)\right)\leq w_p\left(\prod_{j\in\{0,1,\ldots,i\}}(x-a_j)\right).$$

Also, since both $a_{i_l}^l$ and $a_{i_k}^k$ were part of the heap but $ExtractMin()$ procedure returned $a_{i_k}^k$,

$$w_p\left(\prod_{j\in\{0,1,\ldots,i\}}(a_{i_k}^k-a_j)\right)\leq w_p\left(\prod_{j\in\{0,1,\ldots,i\}}(a_{i_l}^l-a_j)\right).$$

Using the above two inequalities,

$$w_p\left(\prod_{j\in\{0,1,\ldots,i\}}(a_{i_k}^k-a_j)\right)\leq w_p\left(\prod_{j\in\{0,1,\ldots,i\}}(x-a_j)\right).$$

But this is a contradiction.

Since, we arrive at a contradiction in both the cases, hence, our assumption must be wrong. Hence,

$$w_p\left(\prod_{i\in\{0,1,\ldots,k-1\}}(a_k-a_i)\right)=\min_{x\in S\setminus\{a_0,a_1,\ldots a_{k-1}\}}\left(w_p\left(\prod_{i\in\{0,1,\ldots,k-1\}}(x-a_i)\right)\right).$$

Hence, our procedure MERGE() gives a valid p-ordering. ■

D.2 Proof of Theorem 5

We prove this by induction on the size of S.

If S is a singleton, then the p-ordering on S is just that element. And hence the corresponding p-value is just $p^0 = 1$. FIND_p-ORDERING(S) sets this value to 1 in step 18. Hence, our assumption is true for $|S| = 1$.

Let our assumption is true for $|S| < k$, if we can show it for $|S| = k$, then by induction, we know our assumption is true for sets of all sizes.

Let $|S| = k$, then when we break this set into smaller $S_0, S_1, ..., S_{p-1}$ (Steps 21-22), either all element belong in a single S_i or get distributed into multiple sets. We handle the two case separately.

Case 1: Let S breaks into smaller $S_0, S_1, ..., S_{p-1}$.

In this case, we know all the $S_0, S_1, ..., S_{p-1}$ have size less that k. Hence, the sizes of $(S_x - x)/p$ is also less than k for all $x \in \{0, 1, ..., p-1\}$. Hence, we get the correct p-values for all elements when we call $Find\_P\_Ordering((S_x - x)/p)$ in step 24.

From Theorem 12, we know that $v_p(S_i - i, k) = v_p((S_i - i)/p, k) + k$, hence we add k to the p-values of all elements of the output(step 27). We know that $v_p(S_i, k) = v_p(S_i - i, k)$, hence, the p-values of each element are correct at the end of step 27.

Next, we show that MERGE() preserves the p-values, we're done, since we know that MERGE() doesn't update the p-values of any of the elements. Let an element q is added at the j^{th} position in the p-ordering output by MERGE(). Let all the elements before this element are in the set X. Then, we know that the p-value of this element is $w_p\left(\prod_{a_i \in X}(q - a_i)\right)$. By Observation 10, we know that this is equal to $w_p\left(\prod_{a_i \in X_{q \pmod p}}(q - a_i)\right)$. Since, merge doesn't re-order the p-orderings on any input S_x while merging, we know that this is exactly the p-value of q from before. Hence, MERGE() preserves the p-values.

Hence, the p-values at the end of MERGE() are correct (step 28). Hence, FIND_p-ORDERING(S) gives the correct p-values.

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

Since, we recursively keep calling FIND_p-ORDERING($\cdot$) on the reduced set, we know at some point, we would reach case 1. As proven above, at this point, we would get the correct p-values. Hence, if we can show that given a correct p-values in step 24, FIND_p-ORDERING($\cdot$) outputs the correct p-values, then by a recursive argument, this would output the correct p-values for any set of size k.

Let's say that all the elements of S fall into some set S_x. We assume that FIND_p-ORDERING($(S_i - i)/p$) outputs the correct p-values, then if we can prove that we get the correct p-values from S, then by the above argument, we are done.

From Theorem 12, we know that $v_p(S_i - i, k) = v_p((S_i - i)/p, k) + k$, hence we add k to the p-values of all elements of the output (step 27). We know that $v_p(S_i, k) = v_p(S_i - i, k)$, hence, the p-values of each element are correct at the end of step 27.

Since, all the elements in S are in just one S_i, MERGE() acts as identity. Hence, the output at the end of Step 28 has the correct p-values. Hence, FIND_p-ORDERING($\cdot$) gives the correct p-values for sets of size k.

Hence, by induction, FIND_p-ORDERING($\cdot$) outputs the correct p-values on any subset of integers. ■

D.3 Time complexity of Algorithm 1

Theorem 14. *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 $\widetilde{O}(nk \log p)$ 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 $\widetilde{O}(\min(m, p))$ (the size of the heap). Next, the construction of common p-ordering(Steps 8-14) takes $\widetilde{O}(m \log p)$ time, this is because extraction of an element and addition of an element are both bound by $\widetilde{O}(\log p)$ and the runs a total of m times. Hence, the total time complexity of MERGE$(S_0, S_1, ..., S_{p-1})$ is $\widetilde{O}(\min(m, p) + m \log p) = \widetilde{O}(m \log p)$ 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 $m_1, m_2, ..., m_q$). Hence, the sum of sizes of all smaller sets at a given depth $\sum_{i=1}^{q} m_i < n$. Hence, the time to run any depth is $\sum_{i=1}^{q} \widetilde{O}(m_i \log p) = \widetilde{O}(n \log p)$. Hence, total time complexity for k depth is $\widetilde{O}(nk \log p)$. ■

Appendix E Correctness and Complexity of Algorithm 2

E.1 Proof of Theorem 7

In this appendix we prove that the *valuations* array from Algorithm 2 maintains the correct valuations.

First we initialize the valuations array to zero, which implies that when we have our p-ordering as a null set ϕ and add the first element to it, we can select any number according to definition 6.

Suppose we have generated a p-ordering upto length $\tilde{n}$ with $i_1, i_2 \dots i_{|S|}$ being the number of elements from each representative root in S. Now if we add another element to this p-ordering, from say the j^{th} representative root, the p-value contributed corresponding to each of the representative roots apart from the j^{th} one will be $v_p(\beta_t - \beta_j)$ where $t \neq j$, according to Observation 9. Also since we have i_t many elements from each of t^{th} representative root, the contribution to p-value will be $i_t v_p(\beta_t - \beta_j)$. Next, we find the p-value contributed due to the same representative root.

Notice that, from Observation 13 we will have the elements of the j^{th} representative root as a p-ordering as well on $\beta_j + p^{k_j}*$, of length i_j. Now by Theorem 11, we will have this p-ordering on $\beta_j + p^{k_j}*$ as $\{\beta_j, \beta_j + p^{k_j}, \beta_j + p^{k_j}2, \dots \beta_j + p^{k_j}(i_j - 1)\}$. When we add another element to this the p-value contributed due to j^{th} representative root will be $k_j v_p(i_j!)$.

Summing them the total p-value at each step, considering the next element to be added being from j^{th} representative root is $\sum_{t \in [|S|]; t \neq j} i_t v_p(\beta_t - \beta_j) + k_j v_p(i_j!)$. We choose j such that this expression is minimum in our algorithm.

Now, we want to show that $valuations[j] = \sum_{t \in [|S|]; t \neq j} i_t v_p(\beta_t - \beta_j) + k_j v_p(i_j!)$. We do this inductively. First we already have 0 stored in each entry of *valuations*. Let, 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 $valuations[j]$. Next, when we add an element from say the t^{th} representative root ($t = min\_index$) we need to change the *valuations* accordingly.

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 29-30).

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 31-32 of Algorithm 2 to incorporate this change. Hence $valuations$ correctly stores the p-value as desired. ■

E.2 Time complexity of Algorithm 2

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 $\widetilde{O}(d^2 k \log p + nk \log p + np)$ time.*

Proof. Let S contains d representative roots of $\mathbb{Z}_{p^k}$ and we want to find the p-ordering up to length n, then, CORRELATE(S) runs a double loop, each of size d, and each iteration takes $\widetilde{O}(k \log p)$, hence, CORRELATE(S) takes $\widetilde{O}(d^2 k \log p)$. p-EXPONENT_INCREASE(n) runs a single loop of size n where each iteration takes $\widetilde{O}(k \log p)$ time, hence, it takes $\widetilde{O}(nk \log p)$. Then main loop run a loop of size n, inside this loop we do $O(d)$ operations on elements of size $\log k$, hence, it takes $\widetilde{O}(nd)$ time. Hence, in total, our algorithm takes $\widetilde{O}(d^2 k \log p + nk \log p + nd)$ time. ■

Appendix F Structure of root sets

Observation 16. *Let $f(x) = \sum_{i=0}^{\infty} b_i \cdot x^i \in \mathbb{Z}_{p^k}[x]$, for $k < p$(k is small), be a polynomial with root-set A. Let $\alpha_i \equiv j \bmod p$ for all $i \in [k]$, be k numbers such that for no i, j, $\alpha_i - \alpha_j \not\equiv 0 \bmod p^2$. Let $\alpha_i \in A$, for all $i \in [k]$, then $S_j = \{s \in \mathbb{Z}_{p^k} \mid s \equiv j \bmod p\} \subseteq A$.*

Proof. Let, for all $i \in [k]$, $\alpha_l = j + p * \beta_l$, then since α_l is in the root set of $f(\cdot)$, therefore,

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

Hence,

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

where, $g_i(x) = \sum_{n=0}^{\infty} \binom{n+i}{n} \cdot b_{n+i} \cdot x^n$. 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, $|det(B)| = \left| \prod_{i \neq j \in [k]} (\beta_i - \beta_j) \right|$. Since $\beta_i - \beta_j \not\equiv 0 \bmod p$, therefore, $det(B) \not\equiv 0 \bmod p$. 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 $S_j \subseteq A$. ■

F.1 Structure of root sets in $\mathbb{Z}_{p^2}$

From Section 5.1, we know if $\alpha = \alpha_0 + \alpha_1 \cdot p \in \mathbb{Z}_{p^2}$ be a root of some $f(x)$ in $\mathbb{Z}_{p^2}$. Then

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

Fixing α_0 to some j, we start looking at structures.

Case 1: root set contains atleast two roots Let, our root set R_j contains two distinct roots, say $j + \alpha_1^0 \cdot p$ and $j + \alpha_1^1 \cdot p$. Then,

$$f(j) + p \cdot \alpha_1^0 \cdot f'(j) = 0 \bmod p^2,$$

and

$$f(j) + p \cdot \alpha_1^1 \cdot f'(j) = 0 \bmod p^2.$$

Solving the above 2 equations, we get

$$f(j) = 0 \bmod p^2,$$

and

$$f'(j) = 0 \bmod p.$$

Hence, any $j + \tilde{\alpha}_1 \cdot p$, for $\tilde{\alpha}_1 \in [p]$, is a root of the polynomial, or $R_j = j + p \cdot *$.

Since, there's no free variable in R_j, we just have 1 root-set of this structure.

Case 2: root set contains one root Let, our root set R_j contains just one root, say $j + \alpha_1 \cdot p$. Then,

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

One can easily see that no new roots seep in at this point and a root set of this form is possible[1]. Hence, $R_j = j + p \cdot \alpha_1$, for $\alpha_1 \in [p]$.

Since, $\alpha_1 \in [p]$, we just have p root-sets of this structure.

Case 3: root set is empty Let our root set is empty.[2]. Hence, $R_j = \emptyset$.

Since, there's no free variable in R_j, we just have 1 root-set of this structure.

Therefore, $N_{p_2} = p + 2$. Hence,

$$R_j = \begin{cases} j + p \cdot *, \\ j + p \cdot \alpha, \text{ for } \alpha \in [p], \\ \emptyset. \end{cases}$$

F.2 Structure of root sets in $\mathbb{Z}_{p^3}$

From Section 5.2, we know if $\alpha = \alpha_0 + \alpha_1 \cdot p + \alpha_2 \cdot p^2 \in \mathbb{Z}_{p^3}$ be a root of some $f(x)$ in $\mathbb{Z}_{p^3}$. Then,

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

Fixing α_0 to some j, we start looking at structures.

Case 1: root set contains atleast three roots different at p^1 Let, our root set R_j contains three roots, say $j + \alpha_1^0 \cdot p + \alpha_2^0 \cdot p^2$, $j + \alpha_1^1 \cdot p + \alpha_2^1 \cdot p^2$ and $j + \alpha_1^2 \cdot p + \alpha_2^2 \cdot p^2$ for $\alpha_1^0 \neq \alpha_1^1 \neq \alpha_1^2$. Then, substituting the value and solving the 3 equations, we get

$$f(j) = 0 \bmod p^3,$$

$$f'(j) = 0 \bmod p^2,$$

and

$$f''(j) = 0 \bmod p.$$

Hence, any $j + \tilde{\alpha}_1 \cdot p + \tilde{\alpha}_2 \cdot p^2$, for $\tilde{\alpha}_1, \tilde{\alpha}_2 \in [p]$, is a root of the polynomial, or $R_j = j + p \cdot *$.

Since, there's no free variable in R_j, we just have 1 root-set of this structure.

[1] Namely $f(x) = x - (j + \alpha_1 \cdot p)$.

[2] Namely $f(x) = a$, where $a \neq 0$

Case 2: root set contains two roots different at p^1 Let, our root set R_j contains three roots, say $j + \alpha_1^0 \cdot p + \alpha_2^0 \cdot p^2$ and $j + \alpha_1^1 \cdot p + \alpha_2^1 \cdot p^2$ for $\alpha_1^0 \neq \alpha_1^1$. Then, substituting the value and solving the 2 equations, we get

$$f(j) = 0 \bmod p^2,$$

and

$$f'(j) = 0 \bmod p.$$

Hence, any $j + \alpha_1^0 \cdot p + \tilde{\alpha}_2 \cdot p^2$, for $\tilde{\alpha}_2 \in [p]$, and $j + \alpha_1^1 \cdot p + \tilde{\alpha}_2 \cdot p^2$, for $\tilde{\alpha}_2 \in [p]$, is a root of the polynomial, or $R_j = \left(j + p \cdot \alpha_1^0 + p^2 \cdot *\right) \cup \left(j + p \cdot \alpha_1^1 + p^2 \cdot *\right)$, for $\alpha_1^0 \neq \alpha_1^1$ and $\alpha_1^0, \alpha_1^1 \in \{0, 1, ...p-1\}$.

Since, $\alpha_1^0 \neq \alpha_1^1$ and $\alpha_1^0, \alpha_1^1 \in \{0, 1, ...p-1\}$, we just have $\frac{p \cdot (p-1)}{2}$ root-sets of this structure.

Case 3: root set contains two roots same at p^1 Let, our root set R_j contains three roots, say $j + \alpha_1 \cdot p + \alpha_2^0 \cdot p^2$ and $j + \alpha_1 \cdot p + \alpha_2^1 \cdot p^2$ for $\alpha_2^0 \neq \alpha_2^1$. Then, substituting the value and solving the 2 equations, we get

$$f(j) = 0 \bmod p^2,$$

and

$$f'(j) = 0 \bmod p.$$

Hence, any $j + \alpha_1 \cdot p + \tilde{\alpha}_2 \cdot p^2$, for $\tilde{\alpha}_2 \in [p]$, is a root of the polynomial, or $R_j = j + p \cdot \alpha_1 + p^2 \cdot *$, for $\alpha_1, \alpha_1^1 \in \{0, 1, ...p-1\}$.

Since, $\alpha_1 \in \{0, 1, ...p-1\}$, we just have p root-sets of this structure.

Case 4: root set contains one root Similar to Appendix F.1, we can have just one root $\alpha = j + \alpha_1 \cdot p + \alpha_2 \cdot p^2$ as a root of $f(x)$ and no new roots seep in. Hence, $R_j = j + p \cdot \alpha_1 + p^2 \cdot \alpha_2$, for $\alpha_1, \alpha_2 \in [p]$.

Since, $\alpha_1, \alpha_2 \in [p]$, we just have p^2 root-set of this structure.

Case 5: root set is empty Similar to Appendix F.1, our root set can be empty. Hence, $R_j = \emptyset$.

Since, there's no free variable in R_j, we just have 1 root-set of this structure.

Therefore, $N_{p_3} = \frac{3p^2+p+4}{2}$. Hence,

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

F.3 Structure of root sets in $\mathbb{Z}_{p^4}$

From Section 5.3, we know if $\alpha = \alpha_0 + \alpha_1 \cdot p + \alpha_2 \cdot p^2 + \alpha_3 \cdot p^3 \in \mathbb{Z}_{p^4}$ be a root of some $f(x)$ in $\mathbb{Z}_{p^4}$. Then,

$$f(\alpha_0) + p \cdot \alpha_1 \cdot f'(\alpha_0) + \left((\alpha_1)^2 \cdot \frac{f''(\alpha_0)}{2} + \alpha_2 \cdot f'(\alpha_0)\right) \cdot p^2 + \left((\alpha_1)^3 \cdot \frac{f'''(\alpha_0)}{6} + 2 \cdot \alpha_1 \cdot \alpha_2 \cdot f''(\alpha_0) + \alpha_3 \cdot f'(\alpha_0)\right) \cdot p^3 = 0 \bmod p^4.$$

Fixing α_0 to some j, we start looking at structures.

Case 1: root set contains atleast four roots different at p^1 Let, our root set R_j contains four roots different at p^1. Then, substituting the value and solving the 4 equations, we get

$$f(j) = 0 \bmod p^4,$$

$$f'(j) = 0 \bmod p^3,$$

$$f''(j) = 0 \bmod p^2,$$

and

$$f'''(j) = 0 \bmod p.$$

Hence, $R_j = j + p \cdot *$.

Since, there's no free variable in R_j, we just have 1 root-set of this structure.

Case 2: root set contains three roots different at p^1 Let, our root set R_j contains three roots different at p^1. Then, substituting the value and solving the 3 equations, we get

$$f(j) = 0 \bmod p^3,$$

$$f'(j) = 0 \bmod p^2,$$

and

$$f''(j) = 0 \bmod p.$$

Hence,

$$R_j = (j + p \cdot \alpha_1 + p^2 *) \cup (j + p \cdot \alpha_2 + p^2 *) \cup (j + p \cdot \alpha_3 + p^2 *)$$

Since, $\alpha_1 \neq \alpha_2 \neq \alpha_3 \in [p]$, we have $\frac{p(p-1)(p-2)}{6}$ such root sets.

Case 3: root set contains three roots of which 2 are different at p^1 and 2 are different at p^2 Similar to last case, we get

$$f(j) = 0 \bmod p^3,$$
$$f'(j) = 0 \bmod p^2,$$

and

$$f''(j) = 0 \bmod p.$$

Hence,

$$R_j = (j + p \cdot \alpha_1 + p^2 *) \cup (j + p \cdot \alpha_2 + p^2 *)$$

Since, $\alpha_1 \neq \alpha_2 \in [p]$, we have $\frac{p(p-1)}{2}$ such root sets.

Case 4: root set contains two roots different at p^2 Similar to last case, we get

$$f(j) = 0 \bmod p^3,$$
$$f'(j) = 0 \bmod p^2,$$

and

$$f''(j) = 0 \bmod p.$$

Hence,

$$R_j = (j + p \cdot \alpha_1 + p^2 *)$$

Since, $\alpha_1 \in [p]$, we have p such root sets.

Case 5: root set contains two roots different at p^1 Let, our root set R_j contains two roots different at p^1. Then, substituting the value and solving the 2 equations, we get

$$f(j) = 0 \bmod p^2,$$

and

$$f'(j) = 0 \bmod p.$$

Hence,

$$R_j = (j + p \cdot \alpha_1 + p^2 \cdot \beta_1 + p^3 *) \cup (j + p \cdot \alpha_2 + p^2 \cdot \beta_2 + p^3 *)$$

Since, $\alpha_1, \alpha_2, \beta_1, \beta_2 \in [p]$ and $\alpha_1 \neq \alpha_2$, we have $\frac{p^3(p-1)}{2}$ such root sets.

Case 6: root set contains two roots different at p^3 Let, our root set R_j contains two roots different at p^3. Then, substituting the value and solving the 2 equations, we get

$$f(j) = 0 \bmod p^2,$$

and

$$f'(j) = 0 \bmod p.$$

Hence,

$$R_j = (j + p \cdot \alpha_1 + p^2 \cdot \alpha_2 + p^3 \cdot *)$$

Since, $\alpha_1, \alpha_2 \in [p]$, we have p^2 such root sets.

Case 7: root set is a single element Similar to the Appendix F.2,

$$R_j = (j + p \cdot \alpha_1 + p^2 \cdot \alpha_2 + p^3 \cdot \alpha_3)$$

Since, $\alpha_1, \alpha_2, \alpha_3 \in [p]$, we have p^3 such root sets.

Case 8: root set is a empty

$$R_j = \emptyset$$

We have 1 such root set.

Therefore, $N_{p_4} = \frac{3p^4+4p^3+6p^2+5p+12}{6}$. Hence,

$$R_j = \begin{cases} j + p \cdot *, \\ (j + p \cdot \alpha_1 + p^2 *) \cup (j + p \cdot \alpha_2 + p^2 *) \cup (j + p \cdot \alpha_3 + p^2 *), \text{ for } \alpha_1 \neq \alpha_2 \neq \alpha_3 \in [p], \\ (j + p \cdot \alpha_1 + p^2 *) \cup (j + p \cdot \alpha_2 + p^2 *), \text{ for } \alpha_1 \neq \alpha_2 \in [p], \\ j + p \cdot \alpha + p^2 * , \text{ for } \alpha \in [p], \\ (j + p \cdot \alpha_1 + p^2 \cdot \beta_1 + p^3 *) \cup (j + p \cdot \alpha_2 + p^2 \cdot \beta_2 + p^3 *), \text{ for } \alpha_1 \neq \alpha_2 \beta_1, \beta_2 \in [p], \\ j + p \cdot \alpha_1 + p^2 \cdot \alpha_2 + p^3 \cdot *, \text{ for } \alpha_1, \alpha_2 \in [p], \\ j + p \cdot \alpha_1 + p^2 \cdot \alpha_2 + p^3 \cdot \alpha_3, \text{ for } \alpha_1, \alpha_2, \alpha_3 \in [p], \\ \emptyset. \end{cases}$$