Primitive Data Structures for the 3SUM and k-SUM Problem

Carolin Zöbelein*

January 9, 2022

Abstract

We discuss the usage of primitive data structures for the 3SUM, respectively kSUM problem for integers and real numbers. We show, that we can map the case of rational numbers on the integer case by introducing a new data structure `fixedint` for an universe $U := 2^{2N-1}$, with integers represented by N bits each. Furthermore, we show that we can map the case of irrational numbers on the integer case, too. For this, we introduce a new data structure `realnumber` and get a belonging universe $U_{qVec} := 2^{(2N-1)(1+|\mathbb{I}_a|)}$ with a maximum possible value $|2^{2N-1}\left(1 + |\mathbb{I}_a| \max\left(\mathbb{I}_a\right)\right)|$.

Keywords: 3SUM Problem, k-SUM Problem, Integers, Rational Numbers, Irrational Numbers, Real Numbers, Data Structures, Data Storage Representation, Complexity

ACM Subject Classes: E1, E2

1 Introduction

We want to consider preprocessing data structures for the `3SUM` problem and its limited to only integers version `Int3SUM`, the generalized problem statement `kSUM` and `IntkSUM`, respectively. In this work, we will only concentrate our attention on data structures, not on problem solving algorithms itself.

The problem statements for the 3SUM case are defined as ...

Definition 1 (`3SUM`)**.** *Given a set $S \subset \mathbb{R}$, $|S| < \infty$, determine if there exist $n_1, n_2, n_3 \in S$ such that $n_1 + n_2 + n_3 = 0$.*

*Email: `contact@carolin-zoebelein.de`, PGP: D4A7 35E8 D47F 801F 2CF6 2BA7 927A FD3C DE47 E13B, Website: `https://research.carolin-zoebelein.de`

Definition 2 (`Int3SUM`)**.** *Given a set $S \subseteq \{-U, \ldots, U\} \subset \mathbb{Z}$, $|S| < \infty$, determine if there exist $z_1, z_2, z_3 \in S$ such that $z_1 + z_2 + z_3 = 0$.*

... and for the k-SUM problem as follows:

Definition 3 (`kSUM`)**.** *Given a set $S \subset \mathbb{R}$, $|S| < \infty$, determine if there exist $n_i \in S$ such that $\sum_{i=0}^{k-1} n_i = 0$.*

Definition 4 (`IntkSUM`)**.** *Given a set $S \subseteq \{-U, \ldots, U\} \subset \mathbb{Z}$, $|S| < \infty$, determine if there exist $z_i \in S$ such that $\sum_{i=0}^{k-1} z_i = 0$.*

Since numerous problems can be reduced from `3SUM` [9] [2], it has raised wide interest. Such problems, which are at least as hard as `3SUM`, are called *3SUM-Hardness*. Some of them are ...

- Given a n-point set in the plane, determine if it contains three collinear points. It was shown by Gajentaan and Overmars [9], that this questions can be solved in $\mathcal{O}\left(n^2\right)$ time.
- Given a set of n triangles in the plane, determine if their union contains a hole or not, and compute the area of their union, respectively. Both can be also solved in $\mathcal{O}\left(n^2\right)$ time [9].
- Given two n-point sets $X, Y \subset \mathbb{R}$, each of size n, determine if all elements in $X + Y = \{x + y \,|\, x \in X, y \in Y\}$ are distinct. It can be solved in $\mathcal{O}\left(n^2 \log\left(n\right)\right)$ time [2]. This problem, including its stronger version with sorting $X + Y$, is used for the conditional lower bounds of problems by Barequet and Har-Peled in [2] and are classified as *Sorting $X + Y$-Hard*. It's an open question if it can be solved in $o\left(n^2 \log\left(n\right)\right)$ [4] [5] [6].

- Given two n-edge convex polygons, determine if one can be placed inside the other by translation and rotation. This can be solved in $\mathcal{O}\left(n^2 \log\left(n\right)\right)$ time [2].

- ...

So, it turns out that examinations of `3SUM` are a starting point for wide studies of problem statements in computational geometry.

We assume, that given be a list L of n numbers of a set S and we will look at primitive data structures for storing each element of our list. For this, we have to differ between several cases. The first one of having integers only and the second one of having real numbers, with a special attention on the characteristics of irrational numbers. During our considerations, we have to keep in mind that $\mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R}$. Additionally, as long as not other mentioned, we will always assume that the number of available bits for representing a rational number q, is sufficient enough to represent q without loss of precision.

1.1 Related Work

Already in the past, some work have been done regarding data structure enhancement for the 3SUM and k-SUM problem. In [10], Gold and Sharir considered *Improved Bounds for 3SUM, k-SUM, and Linear Degeneracy* by randomized 4-linear and $(2k-2)$-linear decision trees with the help of a specialized randomized variant of fractional cascading data structure.

Cabello et al. [3] discussed *Encoding 3SUM* by encoding a real 3SUM instance by rounding its numbers to integers and is mainly inspired by Grønlund and Pettie's non-uniform algorithm for 3SUM [12].

Finally, Golovnev et al. considered data structure upper bounds in *Data Structures Meet Cryptography: 3SUM with Preprocessing* [11], in context of the 3SUM-Indexing problem, a variation of the 3SUM problem which consists of two phases. An offline phase where an algorithm receives $A = \{a_1, \ldots, a_n\}$ real numbers and produces a data structure with m words of w bits each. A second, called online, phase takes the given target b, for which the algorithm needs to find a pair (a_i, a_j) such that $a_i + a_j = b$. Here, the mentioned data structure is used for saving pairs (a_i, a_j) in the first phase, which are then forwarded for comparison with b, in the second phase.

All of this works have in common, that they considering the storing of data in context of algorithms for solving the 3SUM or k-SUM problem. We want to make a step backwards and consider already the way of saving data by primitive data structures before we even think about specific algorithms, at all.

2 Integers

If we look at `Int3SUM` and `IntkSUM`, we need to store integers. We assume, our operations work on a van Neumann architecture, a random-access machine (RAM), with a register size of N (mostly 32 or 64) bits. Typically, integers are saved as N bit strings.

Definition 5 (Positive integers)**.** *Each natural number $z^+ \in \mathbb{Z}_0^+$ can be described by the bit string $B_{z^+} := \left(0, b_{[N-2]}, b_{[N-3]}, \cdots, b_{[1]}, b_{[0]}\right)$ with size $N := \lceil \log_2\left(z^+\right) \rceil + 1$ given by the binary representation of a decimal number $z^+ := \sum_{q=0}^{N-2} b_{[q]} 2^q$ with $b_{[q]} \in \{0, 1\}$.*

Definition 6 (Bit toggle)**.** *The toggle function* $\operatorname{toggle}(B)$ *inverts the bits of the bit string B bitwise by $0 \mapsto 1$ and $1 \mapsto 0$, respectively.*

Definition 7 (Two's complement)**.** *The two's complement B^{-1} of a bit string B with highest bit index $N-1$ is defined as its complement of B so that $B + B^{-1} = 2^N - 1$. The two's complement B^{-1} of B can be determined by $B^{-1} = \operatorname{toggle}(B) + 1$ and the two's complement B of B^{-1} by $B = \operatorname{toggle}\left(B^{-1}\right) + 1$.*

Definition 8 (Negative integers)**.** *Each negative integer $z^- \in \mathbb{Z}^-$ can be described by the bit string $B_{z^-} := \left(1, b_{[N-2]}, b_{[N-3]}, \cdots, b_{[1]} b_{[0]}\right)$ with size $N := \lceil \log_2\left(|z^-|\right) \rceil + 1$ given by the binary representation of the belonging positive decimal number by $z^- := (-1)\, z^+$. Negative integers are represented by the two's complement in binary representation.*

For integers whose negatives are represented in two's complement, we can use the ordinary binary addition.

Definition 9 (Integer binary addition)**.** *The addition of integers z_i in their binary representation is given by a bitwise addition in base two of their bit strings B_{z_i} with carriers $c_{[q]} \in \{0, 1\}$, $q = 0, \ldots N-1$.*

Erickson [8] credited Seidel [7] that `Int3SUM` can be solved with fast Fourier transform in $\mathcal{O}\left(n + U \log\left(U\right)\right)$ time, for a universe of size U (an integer range $[-U, \ldots, U]$) [14], which is given by $U := 2^N$ and subquadratic even for a rather large universe size [12]. Baran, Demaine and Pătraşcu [1] showed that `Int3SUM` can be solved in $\mathcal{O}\left(n^2 / \left(\log\left(n\right) / \log\log\left(n\right)\right)^2\right)$ time with high probability on the word RAM with $U = 2^\omega$ and $\omega > \log\left(n\right)$ machine word size. It uses a mixture of randomized universe reduction by hashing, word packing and table lookups.

3 Real numbers

Consider the problem statements `3SUM` and `kSUM`, we have to store real numbers $n \in \mathbb{R}$, now. We want to differentiate between the case of only having rational numbers $q \in \mathbb{Q}$ and the case of having irrational numbers $i \in \mathbb{R}\backslash\mathbb{Q}$, too.

3.1 Rational numbers

We have different possibilities to handle the storing of rational numbers regarding our problem statements. The first one is something which we call *Integer Reduction*.

Theorem 3.1 (Integer Reduction). *Given are k rational numbers $q_i := \alpha_i/\beta_i$, with $\alpha_i \in \mathbb{Z}$ and $\beta_i \in \mathbb{N}_{\geq 1}$. The k-SUM problem can be reduced to the integer zero sum question $0 \overset{!}{=} \sum_{i=1}^{k} \alpha_i \prod_{j=1,j\neq i}^{k} \beta_j$. The necessary number of bits of each new summand gets kN, if the number of bits of all α_i and β_i bit strings is N each.*

Proof. If we look at the sum $0 \overset{!}{=} \sum_{i=1}^{k} q_i = \sum_{i=1}^{k} \alpha_i/\beta_i = \left(\sum_{i=1}^{k} \alpha_i \prod_{j=1,j\neq i}^{k} \beta_j\right) / \prod_{i=1}^{k} \beta_i$, we see that the zero sum question for rational numbers gets reduced to the integer zero sum question $0 \overset{!}{=} \sum_{i=1}^{k} \alpha_i \prod_{j=1,j\neq i}^{k} \beta_j$, which can be achived by multiplying all numbers by the product of denominators of all q's respectively their lowest common denominator. We can see obviously, that each summand needs kN bits, now. $\square$

At the first glance, we could think that this approach gives us a possibility to reduce the rational case to the integer case of k-SUM, so that we can solve it for example with Seidel's fast Fourier transform. But we can see a problem here. In each summand, we get a mix of nominators and denominators of several numbers, whose product also have to be calculated at first, and makes things complicated.

Next, we want to look at the typical way of saving rational numbers on modern computers, these days. We assume that rational numbers $q \in \mathbb{Q}$ in *Floating-Point Arithmetic* following the *IEEE Standard for Binary Floating-Point Arithmetic* [15] [16] [17] are given, which is used by the most computer systems, today.

Definition 10 (Floating-Point Representation). *The Floating-Point Representation of a real number is defined by*

$$n := s \cdot m \cdot b^e \tag{1}$$

with the sign s, the mantissa m, the base b and an exponent e. The sign $s = (-1)^S$ is given by one bit S ($S = 0$ positive, $S = 1$ negative). The mantissa $1 \leq m < 2$ consist of p mantissa bits of value M such that $m = 1.M = 1 + M/2^p$, because of normalization, so that the leading 1 doesn't have to be saved. Following the IEEE standard for Floating-Point Arithmetic [17], b is always 2. Finally the exponent e is saved as not negative binary number E by adding a fixed bias value B with $E := e + B$. The bias value is determined by $2^{r-1} - 1$ for an exponent e represented by r bits.

The specific representation of floating-point parameters isn't consistent from system to system. The most distributed representation consists of a big endian version with most left bit b_{N-1} being defined as the sign bit, then the reserved exponent bits and finally the reserved mantissa bits as most right bit block.

Lemma 3.1. *Given are n rational numbers $n_i \in \mathbb{Q}$ in floating-point representation according to definition 10. It exists a beijective function $f\left(n_i\right) := n_i 10^{p_{max}}$, $p_{max} \in \mathbb{N}_0$, which maps all n_i on an equivalent set of integers $z_i \in \mathbb{Z}$, by $\mathbb{Q} \to \mathbb{Z}$, $f : n_i \mapsto z_i$, with time complexity $\mathcal{O}\left(n\right)$.*

Proof. For a given set of n rational numbers $n_i \in \mathbb{Q}$ of a list L, we can go through the whole list L to determine the maximum number of decimal places p_{max} within the set of given numbers. Then, we can bring all numbers n_i to an integer representation by moving the radix point by the same number of places p_{max} to the right for all n_i of L by $n_i 10^{p_{max}}$. If we choose for all numbers n_i the same radix point shift, which is furthermore clearly right after the maximum number of decimal places, we get obviously a bijective mapping between $\mathbb{Q}$ and $\mathbb{Z}$. Since, each of the necessary steps for this mapping can be done with constant time, we get a total time complexity of $\mathcal{O}(n)$. □

For analysing the zero sum addition of given floating-point numbers in normalized representation, we have to determine the exponent e of each number and bringing all numbers to the same exponent according to lemma 3.1, so we can do an analog addition like for integers. In *Standard: Floating point to integer mapping* [13] of the Geospatial Intelligence Standards Working Group (GWG), we can find a technical standardization of this mapping.

Compared to our approach by integer reduction, here we can save rational numbers with less necessary memory and without mixing of nominators and denominators. After the mapping according to lemma 3.1, we can take the `Int3SUM` solution of Seidel respectively Baran, Demaine and Pătraşcu.

Now, we want to look at more directly 'integer-like' possibilites of saving real numbers, again. The first way which comes in mind, is *Fixed-Point Arithmetic*. Although, today mostly floating-point numbers are used, because of their large range of values, in some cases, also fixed-point numbers are in usage, because of their higher precision and faster calculation without the leak of necessary exponential shifts for arithmetic calculations like additions.

Definition 11 (Fixed-Point Representation)**.** *A binary fixed point number is defined by its total number of bits N, an optional sign bit s and N^{pre} digits before the radix point and $N^{post} = N - 1 - N^{pre}$ decimal places. The radix point position is fixed. The N^{pre} digits are given as standard integers. The bits of the N^{post} decimal places are given by the fractional parts of the given value to N^{post}.*

For example 0.25_{10} is presented exactly by 0.01_2. Instead 0.7_{10} can only be approximated by $\approx 0.00001011_2$ in Fixed-Point Arithmetic. Basic arithmetic operations like addition or subtraction can be done in the usual way like for integers. Since, the radix point has a fixed position for all numbers, the necessary alignment can be trivially done. The total number of bits of a fixed-point number is $N = 1 + N^{pre} + N^{post}$ to present 2^N different numbers. The maximal possible value depends on the specific choice of N^{pre} and N^{post} and is given by $\left(2^{N^{pre}} - 1\right) . \left(2^{N^{post}} - 1\right)$. Like also already for floating-point numbers, instead of saving the new shifted value, we can also save shifting informations, to save memory space.

The problem in this case is, that for the pre radix point, we have ordinary integers, but for the post radix point not. So, in some cases, we only have approximations for post radix point parts of a rational number and a loss of accuracy of the given decimal.

To solve this mentioned problem, we want to introduce a so-called *Fixed-Point Integer Arithmetic*.

Definition 12 (Fixed-Point Integer Arithmetic)**.** *A rational number $q \in \mathbb{Q}$ in **Fixed-Point Integer Arithmetic** is given by a bit string $B^{N_1+N_2} := (z_1, z_2)$, of length $N_1 + N_2$, consisting of two integers $z_1, z_2 \in \mathbb{Z}$, with length N_1, respectively N_2 bits, and the same sign $s(z_1) = s(z_2)$. In the case that both integers have the same length $N = N_1 = N_2$, our total length of B is noted by $2N$. Since for each integer one bit is reserved as sign bit, the real available total length for our Fixed-Point Integer Arithmetic number is $2N - 1$.*

The radix point position is always fixed. Also like the Fixed Point Arithmetic, the N^{pre} digits are given as standard integer. In constrast to the Fixed-Point Arithmetic the N^{post} decimal places are also given by an integer, interpreting all decimal places as one integer, like if we would simple ignore the radix point and all digits before, and not by a fractional part. For example 2.25_{10} is presented by $10_2.11001_2$. A correct representation of all decimal places within the precision of the number of bits N^{post} is possible, but instead usual standard arithmetic operations like addition or substraction can no longer be done like before.

Example 1. *Consider the addition of the numbers* $0.90_{10} = 00_2.01011010_2$ *and* $0.20_{10} = 00_2.00010100_2$ *from which we subtract* $0.10_{10} = 00_2.00001010_2$ *by* $-0.10_{10} = 00_2.11110110_2$ *in Fixed-Point Integer Arithmetic. For the post radix point addition we get* $110 = 90_{10} + 20_{10} = 01011010_2 + 00010100_2 = 01101110_2$. *It is obviously that substracting* -0.10_{10} *will not give us* $0_{10} = 00000000_2$, *instead we will get* $100_{10} = 01100100_2$. *This means, if we are interested for the zero sum case, we have to look for* 01100100_2*, now.*

```
datastructure fixedint x.y {
  if sign(x) == sign(y):
    (int)        x;
    (int,int)    (getLeadingZeros(y), y);
};
```

Listing 1: Data structure `fixedint`

In listing 1, we give the pseudocode for the implementation of `fixedint` for rational numbers by two integers x and y. If both integers have the same sign, we proceed, else we do nothing. Since, mostly computer systems ignore leading zeros when entering numbers, we save for y the number of its leading zeros by `getLeadingZeros(y)` and the y value itself, without leading zeros.

Lemma 3.2. *Given are* n *rational numbers* $n_i \in \mathbb{Q}$ *in fixed-point integer arithmetic according to definition 12 with* $n_i := x_i.y_i$, $x_i, y_i \in \mathbb{Z}$: $sign(x_i) = sign(y_i)$. *It exists a bijective function* $f(y_i) := y_i 10^{p_{max} - nbOfFigures(y_i)}$, $p_{max} \in \mathbb{N}_0$, *which maps all* y_i *on an equivalent set of integers* $z_i \in \mathbb{Z}$, *by* $\mathbb{Z} \to \mathbb{Z}$, $f : y_i \mapsto z_i$, *with the same number of figures, including leading zeroes, totally, and a time complexity* $\mathcal{O}(n)$. *The necessary memory for saving one number in fixed-point integer arithmetic is* $2N$ *and hence for* n *numbers it is* $2Nn$.

Proof. For a given set of n rational number $n_i \in \mathbb{Q}$ of a list L, we can go through the whole list L to determine the maximum number of figures p_{max}, including leading zeroes, within the set of given numbers y_i of n_i. Then, we can bring all numbers y_i to the same total maximum number of figures, including leading zeroes, with filling the missing digits with zeroes for all y_i of n_i in L by $y_i 10^{p_{max} - nbOfFigures(y_i)}$. If we choose for all numbers y_i the same p_{max}, which is furthermore clearly right after the maximum number of figures, including leading zeroes, we get obviously a bijective mapping between $\mathbb{Z}$ and $\mathbb{Z}$. All necessary steps for this mapping can be done with constant time. Hence, the total time complexity is $\mathcal{O}(n)$. Since, we use for each fixed-point integer arithmetic number two integers, the necessary memory for saving one of them is clearly $2N$ and for a list L of n numbers it is $2Nn$. □

```
int getMaxYPrecision (fixedint list L) {
  int pmax = 0;
  for all l in L:
    int p = l.y[0] + nbOfFigures(l.y[1]);
    if p > pmax:
        pmax = p;
  return pmax;
};

fixedint list preprocessList \
    (fixedint list L) {
  int pmax = getMaxYPrecision(L);
  for all l in L:
    l.y[1] = (l.y[1])*pow(10,pmax - \
      l.y[0] - nbOfFigures(l.y[1]));
  return L;
};
```

Listing 2: Preprocessing of `fixedint` list *L* for the k-SUM problem

In listing 2, we have the pseudocode implementation for preprocessing according to lemma 3.2 of a `fixedint` list L for the k-SUM problem. `getMaxYPrecision (fixedint list L)` determines the maximum number of figures p_{max}, including leading zeroes, of given numbers y_i of n_i in L. Finally, `fixedint list preprocessList (fixedint list L)` does the mapping to new numbers $z_i \in \mathbb{Z}$ with the same number of figures, including leading zeroes, itself.

Finally, we want to take a look at the possibilities to use our fixed-point integer arithmetic for the 3SUM respectively k-SUM problem. In example 1, we already mentioned that we can get an overflow during addition in the y part which has still to be considered in the x part during addition. If we take the k-SUM of rational numbers in fixed-point integer arithmetic with a maximum y

precission of p_{max}, then we can have valid solutions for the zero sum questions of the y parts of k numbers n_i by $\{-(k-1)\,10^{p_{max}}, -(k-2)\,10^{p_{max}}, \dots, -10^{p_{max}}, 0, 10^{p_{max}}, \dots, (k-2)\,10^{p_{max}}, (k-1)\,10^{p_{max}}\}$.

Since, Seidel as well as Baran, Demaine and Pătraşcu explicitly discuss the integer 3SUM problem, we want to do this likewise taking into account their algorithms. For $k = 3$ we get the possible addition overflows of two y parts by $\{-10^{p_{max}}, 0, 10^{p_{max}}\}$, which we have to check against the third x and y part. Now, we have different possible ways of handling this overflow.

The first one takes one of the existing `Int3SUM` algorithms and runs it as usual for our numbers n_i by treating x_i and y_i as one integer of size $2N-1$ and with ignoring the sign bit and the belonging carrier of y_i addition which would else slop over to the x_i addition. If we have the case that $\sum_{i=1}^{3} y_i = 0$, then we get directly the correct solutions. If we have the case that $\sum_{i=1}^{3} y_i = \pm 10^{p_{max}}$, then we get a total sum $\sum_{i=1}^{3} n_i$ which differs from the correct solution by $\mp 10^{p_{max}}$. So, if we receive one of the cases $\pm 10^{p_{max}}$, then we can check if they are valid by doing an y_i part only addition $\sum_{i=1}^{3} y_i$ and comparing this result with the total sums of n_i. If we integrate this check directly into the algorithm after the step of determining a specific solution, we can get a smooth algorithm flow.

The other way, which we can take, turns out if we look closer at the algorithms itself. Both work, rawly spoken, as follows: Given be three lists A, B and C, each with n integers. We calculate all possible pairs $A \times B$ with $a_i \in A$, $b_j \in B$, $\sigma_{i,j} := a_i + b_j$ and compare all the resulting $\sigma_{i,j}$'s with the elements of C. Seidel uses a fast Fourier transformation for this comparison step, while Baran, Demaine and Pătraşcu use a randomized approach with hashing and lookup tables. Now, if we look at our first way and at their algorithms, it comes in mind that we can add the overflow checking step at the place between calculating the pairs $A \times B$ and the comparison with C by searching for all three possible overflow version at first, keeping the real overflow of y_i sum into account and hence, in this way, finally to be able to strike out not valid solutions.

In both cases we can easily keep the overflow into account, by differentiate between the three possible addition overflows of two y parts ahead, before we even run any algorithm at all. For this, we sort all n_i for their sign, and then furthermore we build for all numbers with the same sign the sets $S_1^{\pm} := \{x_i.y_i \,|\, \forall\, x_i.y_i \in [0, \pm x_i.5[\}$ and $S_2^{\pm} := \{x_i.y_i \,|\, \forall\, x_i.y_i \in [\pm x_i.5, \pm x_i.0[\}$ for their y_i parts, respectively. Since, we know that for the addition $n_i + n_j$ of two positive numbers, we have an overflow of 0 for the cases $n_i, n_j \in S_1^{+}$ and $n_i \in S_1^{+} \wedge n_j \in S_2^{+}$, and an overflow of $10^{p_{max}}$ for the case $n_i, n_j \in S_2^{+}$ (analog for two negative numbers), we can directly take the correct overflow into account, if we run the algorithm separetly for each of this cases, so that we get totally six algorithm runs, each for one case.

3.2 Irrational numbers

At first, we need to take a short look at the character of irrational numbers, in generally. Irrational numbers are numbers which can not be presented by a fraction $\frac{z_1}{z_2}$, $z_1, z_2 \in \mathbb{Z}$. The most known irrational numbers are π, e and also $\sqrt{2}$. Although this numbers play an important role, there are still some unsolved questions regarding them.

At first, it is not known, if for all cases of $i_1 i_2$ with $i_1, i_1 \in \mathbb{R} \setminus \mathbb{Q}$, the product is still also an irrational number. Until now, there couldn't be found a counterexample but also no general proof, that all cases are leading to irrational numbers [18]. Secondly, it is also not known, if the linear combination of two irrational numbers by $q_1 i_1 + q_2 i_2$ with $q_1, q_2 \in \mathbb{Q}$ is always an irrational number [18] [19], too. Finally, it exists just an assumption, but not a proof, for the irrational nature of the numbers 2^e, π^e, $\pi^{\sqrt{2}}$, π^π, e^e and γ (the Euler–Mascheroni constant) [18].

Because of this unsolved questions, we want to describe irrational numbers regarding the k-SUM problem in a specific way. For this, we introduce **Atomic Irational Numbers**.

Definition 13 (Atomic Irrational Numbers)**.** *Given a finite set $\mathbb{I}_a$ of for being for sure irrational numbers $i_a \in \mathbb{R} \setminus \mathbb{Q}$. We call this numbers* ***Atomic Irrational Numbers*** *if they satisfy all of the following properties:*

1. *For all $i_{a,1}, i_{a,2} \in \mathbb{I}_a$, we have $i_{a,2} \neq q i_{a,1}$ iff $q \neq 1$, and $q \neq i_{a,1}^{-1} i_{a,2}$, $\forall q \in \mathbb{Q}$.*
2. *For all $i_{a,1}, i_{a,2} \in \mathbb{I}_a$ exists an irrational number $i \in \mathbb{R} \setminus \mathbb{Q} \wedge i \notin \mathbb{I}_a$, with $i := q_1 i_1 + q_2 i_2$, $\forall q_1, q_2 \in \mathbb{Q}$ and $(q_1, q_2) \notin \{(0,0), (0,1), (1,0)\}$.*

3. *If such cases exist, we are in the know of the properties $q_1 i_{a,1} + q_2 i_{a,2} = q_3$, $q_1, q_2, q_3 \in \mathbb{Q}$ for all $i_{a,1}, i_{a,2} \in \mathbb{I}_a$ and know the complete tuples $(q_1, i_{a,1}, q_2, i_{a,2}, q_3)$.*

Now, we want to put attention on the possibility to present a real number n itself by a given combination of several single numbers, too. Following from this, we can write, for example, for a complete description of some possible cases of a real number $n = q_1 q_2 + q_3 i_1 + q_4 i_2$, with $q_j \in \mathbb{Q}$ and $i_j \in \mathbb{R} \setminus \mathbb{Q}$. Let's look at the sum of the three real numbers n_1, n_2 and n_3 with keeping this description in mind. We know that an irrational number i can only disappear as the result of the addition of several numbers n, if this number i respectively its multiples are element of some of this numbers n. An irrational number i can not be eliminated by neither an other irrational number $i' \neq i$ and their multiples nor by any rational number. In addition we know that $q_1 q_2, \in \mathbb{Q}$ and $q_3 i_1 \in \mathbb{R} \setminus \mathbb{Q} \wedge q_3 i_1 \notin \mathbb{I}_a$ iff $q_3 \neq 1$. This leads to the point that for further steps of solving the 3SUM problem, we have to split n into two parts: the one with the rational summands $q_1 q_2$ and the one with the irrational summands $q_3 i_1$ and $q_4 i_2$.

Hence, we can write for a sum of three real numbers n_j also $\sum_{j=1}^{3} n_j = \sum_{j=1}^{3} (q_j + q_{1,j} i_1 + q_{2,j} i_2) = \sum_{j=1}^{3} q_j + i_1 \sum_{j=1}^{3} q_{1,j} + i_2 \sum_{j=1}^{3} q_{2,j}$, and we see, that if we are able to identify and flag an irrational number clearly as irrational, it becomes not necessary anymore to save its specific value and our zero sum problem can be reduced to a rational zero sum problem for this specific summand, too. Now, with this, we define our real numbers.

Theorem 3.2 (Real numbers)**.** *Each real number $n \in S \subset \mathbb{R}$ of a finite subset S, can be written by $n = q_0 + \sum_{\forall i_{a,j} \in \mathbb{I}_a} i_{a,j} q_j$, with $q_j \in \mathbb{Q}$ and $\mathbb{I}_a \subset S$.*

Proof. Be n a rational number $q \in \mathbb{Q}$. According to our properties 1-2 from definition 13, each $n \in S$ is rational with $n = q_0$, if $0 = \sum_{\forall i_{a,j} \in \mathbb{I}_a} i_{a,j} q_j$, which we get if $q_j = 0, \forall i_{a,j} \in \mathbb{I}_a$.

Be n an irrational number $i \in \mathbb{R} \setminus \mathbb{Q}$. We get all irrational numbers $i_{a,k} \in \mathbb{I}_a$, if $q_0 = 0$ and $q_j = 0$, $\forall i_{a,j} \in \mathbb{I}_a \setminus \{i_{a,k}\}$. We get irrational numbers $i \in S \setminus \mathbb{I}_a$, if $q_0 = 0$ and $|\{q_j \neq 0 \,|\, \forall i_{a,j} \in \mathbb{I}_a\}| \geq 2$.

If cases of kind $q' := \sum_{\forall i_{a,j} \in \mathbb{I}'_a \subseteq \mathbb{I}_a} i_{a,j} q_j$, $q' \in \mathbb{Q}$ exist, we are aware of this according to property 3 of definition 13. Regarding the right handling of saving all irrational numbers of L for the k-SUM problem, in particular this case, see the following discussion. □

After the definition of our way of handling real numbers, in particular irrational numbers, we have to think about the right way of recording them.

```
// Define irrational number strings
string vec realnumbers = \
  [empty].append([PI, e, ...]);

datastructure realnumber \
    (fixedint vec qVec) {
  if length(realnumbers) == length(qVec):
    map realnumbers -> qVec;
};
```

Listing 3: Data structure `realnumber`

In listing 3, we can see the pseudocode of our data structure `realnumber`. According to our equation $n = q_0 + \sum_{\forall i_{a,j} \in \mathbb{I}_a} i_{a,j} q_j$ with the set of atomic irrational numbers $\mathbb{I}_a$, at first, we define a string vector `string vec realnumbers`, in which we save the symbolic meaning of each component of n. The first element is empty, since it represents the rational part q_0. The further elements represent one element of $\mathbb{I}_a$, each in the string vector. In our data structure itself, we then do a symbolic bijective mapping of our irrational string vector of numbers $i_{a,j}$, to each belonging rational number q_j, which we save as `fixedint` (if we prefer this, we could also use of course for example floating-point numbers, here) in our vector `vec qVec`.

Now, considering a list L for the k-SUM problem, we save all given numbers n_i in a list L of `datastructure realnumber`, for the same fixed defined `string vec realnumbers` for all n_i. If we look at the possible cases of kind $q' := \sum_{\forall i_{a,j} \in \mathbb{I}'_a \subseteq \mathbb{I}_a} i_{a,j} q_j$, $q' \in \mathbb{Q}$, we replace the q_0 entry by $q_0 + q'$ and set $q_j = 0, \forall i_{a,j} \in \mathbb{I}'_a \subseteq \mathbb{I}_a$, in `qVec`.

The total time complexity of storing one real number $n \in \mathbb{R}$ with `realnumber`, including the necessary preproccesing by handling possible q' cases, is $\mathcal{O}(1)$ and the necessary memory for saving one `realnumber` is $(2N-1)(1+|\mathbb{I}_a|)$.

Similiar like for a single `fixedint` rational number, we can also use Seidel's or Baran, Demaine and Pǎtraşcu's solution for a single `fixedint` by taking possible overflows into account and additionally, also by ignoring the result and binary addition carrier of the most left bit $b_{[N-1]}$, the sign bit, of x of each rational number, during calculations, now. The belonging universe for `qVec` is given by $U_{qVec} := 2^{(2N-1)(1+|\mathbb{I}_a|)}$. The maximum possible value which can be represented is $|2^{2N-1}(1 + |\mathbb{I}_a| \max(\mathbb{I}_a))|$.

4 Conclusion

Our considerations of primitive data structures for the 3SUM and k-SUM problem, shows that we can map the problem statement for real numbers, with few work, on the integer case. We want to put attention on the point, that we always assumed of having a fixed size of N bits, for all integers available. Hence, for our definition of `fixedint`, we received a representation for nearly the doubled amount of numbers, like for integers, and for `realnumber`, we received even the $(1 + |\mathbb{I}_a|)$ amount compared to the `fixedint` case. If we want to keep the number of possible presentable numbers constant over all kinds of number types, we have to adjust the number of bits for integer size, in an appropriate way, for `fixedint` and `realnumber`, too.

References

[1] Ilya Baran, Erik D Demaine, and Mihai Pǎtraşcu, *Subquadratic algorithms for 3SUM*, Workshop on Algorithms and Data Structures, Springer, 2005, pp. 409–421.

[2] Gill Barequet and Sariel Har-Peled, *Polygon containment and tranlational in-Hausdorf-distance between segment sets are 3SUM-hard*, International Journal of Computational Geometry & Applications **11** (2001), no. 04, 465–474.

[3] Sergio Cabello, Jean Cardinal, John Iacono, Stefan Langerman, Pat Morin, and Aurélien Ooms, *Encoding 3SUM*, arXiv preprint arXiv:1903.02645 (2019).

[4] Erik D. Demaine, Joseph S. B. Mitchell, and Joseph O'Rourke, *The open problems project*, `http://cs.smith.edu/~jorourke/TOPP/`, 09 2017, (Accessed on 2020/07/30).

[5] ———, *The open problems project - problem 11: 3SUM hard problems*, `http://cs.smith.edu/~jorourke/TOPP/P11.html#Problem.11`, 09 2017, (Accessed on 2020/07/30).

[6] ———, *The open problems project - problem 41: Sorting $x + y$ (pairwise sums)*, `http://cs.smith.edu/~jorourke/TOPP/P41.html#Problem.41`, 09 2017, (Accessed on 2020/07/30).

[7] Herbert Edelsbrunner, Joseph O'Rourke, and Raimund Seidel, *Constructing arrangements of lines and hyperplanes with applications*, SIAM Journal on Computing **15** (1986), no. 2, 341–363.

[8] Jeff Erickson et al., *Lower bounds for linear satisfiability problems.*, SODA, 1995, pp. 388–395.

[9] Anka Gajentaan and Mark H Overmars, *On a class of $O(n^2)$ problems in computational geometry*, Computational geometry **5** (1995), no. 3, 165–185.

[10] Omer Gold and Micha Sharir, *Improved bounds for 3SUM, k-SUM, and linear degeneracy*, arXiv preprint arXiv:1512.05279 (2015).

[11] Alexander Golovnev, Siyao Guo, Thibaut Horel, Sunoo Park, and Vinod Vaikuntanathan, *Data structures meet cryptography: 3SUM with preprocessing*, Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, 2020, pp. 294–307.

[12] Allan Grønlund and Seth Pettie, *Threesomes, degenerates, and love triangles*, 2014 IEEE 55th Annual Symposium on Foundations of Computer Science, IEEE, 2014, pp. 621–630.

[13] Geospatial Intelligence Standards Working Group (GWG), *Standard: Floating point to integer mapping*, `https://gwg.nga.mil/misb/docs/standards/ST1201.1.pdf`, 02 2014, (Accessed on 2021/02/21).

[14] LH Harper, Thomas H Payne, John E. Savage, and E Straus, *Sorting $x+y$*, Communications of the ACM **18** (1975), no. 6, 347–349.

[15] IEEE, *IEEE Standard for binary floating-point arithmetic*, ANSI/IEEE Std 754-1985 (1985), 1–20.

[16] ______, *IEEE Standard for floating-point arithmetic*, IEEE Std 754-2008 (2008), 1–70.

[17] ______, *IEEE Standard for floating-point arithmetic*, IEEE Std 754-2019 (Revision of IEEE 754-2008) (2019), 1–84.

[18] Wolfram MathWorld, *Irrational number – from wolfram mathworld*, `https://mathworld.wolfram.com/IrrationalNumber.html`, (Accessed on 2020/09/05).

[19] ______, *Pi*, `https://mathworld.wolfram.com/Pi.html`, (Accessed on 2020/09/05).