On the minimum gap between sums of square roots of small integers ☆

Qi Cheng[a], Yu-Hsin Li[a]

[a] *School of Computer Science*
The University of Oklahoma
Norman, OK 73019, USA.
Email: {qcheng,yli}@cs.ou.edu.

Abstract

Let k and n be positive integers, $n > k$. Define $r(n,k)$ to be the minimum positive value of

$$|\sqrt{a_1} + \cdots + \sqrt{a_k} - \sqrt{b_1} - \cdots - \sqrt{b_k}|$$

where $a_1, a_2, \cdots, a_k, b_1, b_2, \cdots, b_k$ are positive integers no larger than n. Define $R(n,k)$ to be $-\log r(n,k)$. It is important to find tight bounds for $r(n,k)$ and $R(n,k)$, in connection to the sum-of-square-roots problem, a famous open problem in computational geometry. The current best lower bound and upper bound are far apart. In this paper, we prove an upper bound of $2^{O(n/\log n)}$ for $R(n,k)$, which is better than the best known result $O(2^{2k}\log n)$ whenever $n \le ck\log k$ for some constant c. In particular, our result implies an algorithm *subexponential* in k (i.e. with time complexity $2^{o(k)}(\log n)^{O(1)}$) to compare two sums of k square roots of integers of value $o(k\log k)$. We then present an algorithm to find $r(n,k)$ *exactly* in $n^{k+o(k)}$ time and in $n^{\lceil k/2\rceil+o(k)}$ space. As an example, we are able to compute $r(100,7)$ exactly in a few hours on a single PC. The numerical data indicate that the root separation bound is very far away from the true value of $r(n,k)$.

Keywords: Sum of square roots problem, Geometric computing, Computational number theory.

1. Introduction

In computational geometry, one often needs to compare lengths of two polygonal paths, whose nodes are on an integral lattice, and whose edges are measured according to the Euclidean norm. The geometrical question can be reduced to a

☆This research is partially supported by NSF grant CCF-0830522 and CCF-0830524. Part of the results, in their preliminary form, appeared in [5] and [6].

numerical problem of comparing two sums of square roots of integers. In computational geometry one sometimes assumes a model of real-number machines, where one memory cell can hold one real number. It is then assumed that an algebraic operation, taking a square root as well as a comparison between real numbers can be done in one operation. There is a straight-forward way to compare sums of square roots in real-number machines. But this model is not realistic, as shown in [14, 12].

If we consider the problem in the model of the Turing machine, then we need to design an algorithm to compare two sums of square roots of integers with low bit complexity. One approach would be approximating the sums by decimal numbers up to a certain precision, and then hopefully we can learn which one is larger. Formally define $r(n,k)$ to be the minimum positive value of

$$|\sqrt{a_1} + \cdots + \sqrt{a_k} - \sqrt{b_1} - \cdots - \sqrt{b_k}|$$

where $a_1, a_2, \cdots, a_k, b_1, b_2, \cdots, b_k$ are positive integers no larger than n. Define $R(n,k)$ to be $-\log r(n,k)$. The time complexity of the approximation approach is polynomial on $R(n,k)$, since an approximation of a sum of square roots of integers can be computed in time polynomial in the number of precisions. One would like to know if $R(n,k)$ is bounded from above by a polynomial function in k and $\log n$. If so, the approximation approach to compare two sums of square roots of integers runs in polynomial time. Note that even if the lower bound of $R(n,k)$ is exponential, it does not necessarily rule out a polynomial time algorithm.

Although this problem was put forward during the 1980s [7], not many results have been reported. In [3], it is proved that

$$R(n,k) = O(2^{2k} \log n)$$

using the root separation method. Qian and Wang [11] presented a constructive upper bound for $r(n,k)$ at $O(n^{-2k+\frac{3}{2}})$. The constant hidden in the big-O can be derived from their paper and it depends on k. Taking it into account, one can show that

$$R(n,k) \geq 2k \log n - 8k^2 + O(\log n + k \log k).$$

See Section 3 for details. Hence the bound is nontrivial only when $n \geq 2^{4k}$. There is another upper bound for $r(n,k)$ using solutions for the Prouhet–Tarry–Escott problem [11]. However, the Prouhet–Tarry–Escott problem is hard to solve by itself.

There is a wide gap between the known upper bound and lower bound of $r(n,k)$. For example by the root separation method, one has a lower bound

$$r(100,7) \geq (14 * \sqrt{100})^{-2^{13}} \approx 10^{-17581}.$$

One can not derive a nontrivial upper bound for $r(100,7)$ either from Qian and Wang's method, or from the Prouhet–Tarry–Escott method.

2. Our contribution

We first prove a simple lower bound for $R(n,k)$ based on a pigeonhole argument:

$$R(n,k) \geq k\log n - k\log k + O(k+\log n).$$

In comparison to Qian–Wang's bound, it is weaker when n is very large, but it is better when n is polynomial on k, hence it has wider applicability. For example, when $n = 100$ and $r = 7$, it can give us a meaningful upper bound:

$$r(100,7) \leq 7.2 \times 10^{-8}.$$

From the known upper bound of $R(n,k)$, we conclude that there is a polynomial time algorithm to compare sum of square roots if k is fixed. We consider the case in the other end of the spectrum when k grows (almost) linear with n.

Definition 1. *An integer n is called square free, if there does not exist a prime p such that p^2 divides n. We use $s(n)$ to denote the number of positive square free integers less than n, e.g. $s(100) = 61$.*

It is well known that [10]

$$s(n) = \frac{6n}{\pi^2} + O(\sqrt{n}).$$

Proposition 1. *Suppose that $s_1, s_2, \cdots, \cdots$, and s_k are distinct positive square-free integers. Then $\sqrt{s_1}, \sqrt{s_2}, \cdots$, and $\sqrt{s_k}$ are linear independent over $\mathbf{Q}$.*

So it is possible that k and n are linearly related. This case is also practically interesting. We often need to compare paths whose nodes are on an $l \times l$ integral grid. The distance between the lattice points are square roots of integers of size $O(l^2)$. There are l^2 many nodes in the grid, and if a path passes a dense subset out of the grid points, its length is a sum of k square roots of integers less than n where n is linear in k.

We obtain a lower bound of difference of two sums of square roots, which beats the root separation bound as long as $n \leq ck\log k$ for some constant c. The corresponding upper bound of

$$R(n,k) = 2^{O(n/\log n)}$$

becomes *subexponential* in k when $n = o(k\log k)$, or more generally, if the square free parts of the numbers grow at rate $o(k\log k)$. Our bound implies a subexpontial algorithm, i.e., an algorithm with time complexity $2^{o(k)}(\log n)^{O(1)}$, to compare two sums of square roots of small integers.

Moreover, we show that the root separation lower bound $2^{O(k)}\log n$ of $R(n,k)$ is *not* tight, at least, when n is linear in k. It is still possible that when n is much larger than k, the root separation bound becomes tight. Our result indicates that to achieve the root separation bound, it is important to select the numbers $a_1, a_2, \cdots, a_k, b_1, b_2, \cdots, b_k$ such that they are pairwise relatively prime.

The lack of strong bounds for $r(n,k)$ after many years of study indicates that finding a tight bound is likely to be very hard. We feel that the situation warrants an extensive numerical study of $r(n,k)$. So far only a few toy examples have been reported and they can be found easily using an exhaustive search:

$$r(20,2) \approx .0002 = \sqrt{10} + \sqrt{11} - \sqrt{5} - \sqrt{18}.$$

$$r(20,3) \approx .000005 = \sqrt{5} + \sqrt{6} + \sqrt{18} - \sqrt{4} - \sqrt{12} - \sqrt{12}.$$

Computing power has gradually increased which makes it feasible for us to go beyond toy examples. In addition, there are other motivations for a numerical study of the sum-of-square-roots problem:

1. The numerical data shed light on the type of integers whose square roots summations are extremely close.
2. In many practical situations, especially in the exact geometric computation, n and k are small. Explicit bounds like one we produce here help to speed up the computation, as they are better than the bounds predicted by the root separation method.
3. Since the upper bound is so far away from the lower bound, the numerical data may provide us some hints on which bound is closer to the truth and may inspire us to formulate a reasonable conjecture on a tight bound of $r(n,k)$.

How can we find the exact value of $r(n,k)$? The naive exhaustive search uses little space but requires n^{2k} time. If $n = 100$ and $k = 7$, the algorithm needs about $100^{14} \approx 2^{93}$ operations, which is prohibitive. A better approach would be first sorting all the summations of $\sqrt{a_1} + \cdots + \sqrt{a_k}$ ($1 \leq a_i \leq n$ for all $1 \leq i \leq k$) and then going through the sorted list to find the smallest gap between two consecutive elements. It runs in time at least n^k and in space at least n^k. If $n = 100$ and $k = 7$, then the approach would use at least $100^7 = 10^{14} \approx 10000$ Gbytes of space, under an overly optimistic assumption that we use only one byte to hold one value of the summation. The space complexity makes the computation of $r(100,7)$ very expensive, to say the least.

We present an algorithm to compute $r(n,k)$ exactly based on the idea of enumerating summations using heap. Our algorithm uses much less space than the sorting approach while preserving the time complexity, which makes computing $r(100,7)$ feasible. Indeed it has the space complexity at most of $n^{\lceil k/2 \rceil + o(k)}$. Our search reveals that

$$r(100,7) = 1.88 \times 10^{-19},$$

which is reached by

$$\sqrt{7}+\sqrt{14}+\sqrt{39}+\sqrt{70}+\sqrt{72}+\sqrt{76}+\sqrt{85} = 47.42163068019049036900034846$$

and

$$\sqrt{13}+\sqrt{16}+\sqrt{46}+\sqrt{55}+\sqrt{67}+\sqrt{73}+\sqrt{79} = 47.42163068019049036881196876.$$

3. An upper bound of $R(n,k)$ from the pigeonhole principle

Qian-Wang's upper bound was derived from the inequality:

$$0 < \left|\sum_{i=0}^{2k-1} \binom{2k-1}{i}(-1)^i\sqrt{t+i}\right| \le \frac{1*3*5*\cdots*(4k-5)}{2^{2k-1}t^{2k-\frac{3}{2}}}.$$

Let $a_i = \binom{2k-1}{2i-2}^2(t+2i-2)$ for $1 \le i \le k$ and $b_i = \binom{2k-1}{2i-1}^2(t+2i-1)$ for $1 \le i \le k$, we have

$$0 < |\sum_{i=1}^{k}\sqrt{a_i} - \sum_{i=1}^{k}\sqrt{b_i}| \le \frac{1*3*5*\cdots*(4k-5)}{2^{2k-1}t^{2k-\frac{3}{2}}}.$$

Note that $\binom{2k-1}{i}$ can be as large as $\binom{2k-1}{k} \ge 2^{2k-1}/(2k)$. To get an upper bound for $r(n,k)$, assign

$$n = \binom{2k-1}{k}^2(t+k), \tag{1}$$

thus we have

$$R(n,k) \ge 2k\log n - 8k^2 + O(\log n + k\log k).$$

Hence Qian and Wang's result only applies when n is much greater than 2^{4k}. In particular it does not give a meaningful bound for $r(100,7)$.

Another interesting upper bound depends on the Prouhet–Tarry–Escott problem, which is to find a solution for a system of equations:

$$\sum_{i=1}^{k} a_i^t = \sum_{i=1}^{k} b_i^t, 1 \le t \le k-1$$

under the condition that $a_1 \le a_2 \cdots \le a_k$ and $b_1 \le b_2 \cdots \le b_k$ are distinct lists of integers. However no such solutions have been found for $k = 11$ and $k > 13$ [2]. Therefore the approach based on the Prouhet–Tarry–Escott problem is not scalable.

Here we present an upper bound based on the pigeonhole argument.

Theorem 1. *We have*

$$r(n,k) \le \frac{k\sqrt{n}-k}{\binom{s(n)+k-1}{k}-1}.$$

Proof: Consider the set

$$\{(a_1,a_2,\cdots,a_k)|a_i \text{ is squarefree }, 1 \le a_1 \le a_2 \le \cdots \le a_k \le n.$$

The set has cardinality $\binom{s(n)+k-1}{k}$. For each element $(a_1,a_2,\cdots,a_k)$ in the set, the sum $\sum_{i=1}^{k}\sqrt{a_i}$ is distinct by Proposition 1. Hence there are $\binom{s(n)+k-1}{k}$

many distinct sums in the range $[k, k\sqrt{n}]$. There must be two points within the distance $\frac{k\sqrt{n}-k}{\binom{s(n)+k-1}{k}-1}$ from each other. The theorem follows. □

Plugging in $n = 100$ and $k = 7$, we have

$$r(100, 7) \leq \frac{(70-7)}{\binom{67}{61}-1} = 7.2 \times 10^{-8}.$$

Since $s(n) = \frac{6n}{\pi^2} + O(\sqrt{n})$, one can derive

Corollary 1.

$$R(n, k) \geq k \log n - k \log k + O(\log(nk))$$

Note that when n is much larger than k, then this bound is not as good as Qian and Wang's bound.

4. A lower bound of $R(n, k)$

We begin the presentation of our result by defining the notion of multiplicative generators.

Definition 2. *Given two sets of positive integers A and B, we say that B multiplicatively generates A if any number in A can be written as a product of numbers from B with repetition allowed.*

It is easy to see that A multiplicatively generates itself, but for many sets, there exist much smaller sets which multiplicatively generate them. For example, all the square free number less than n are generated by the set of primes less than n, whose cardinality is $O(n/\log n)$.

Theorem 2. *Let $c_1, c_2, \cdots, c_k, d_1, d_2, \cdots, d_k$ be positive integers. Let*

$$A = \{a_1, a_2, \cdots, a_k, b_1, b_2, \cdots, b_k\}$$

be the set of $2k$ positive square free integers. Assume that $c_i^2 a_i \leq n$ for all $1 \leq i \leq k$ and $d_i^2 b_i \leq n$ for all $1 \leq i \leq k$. Let B be a set which multiplicatively generates A. Then

$$|c_1\sqrt{a_1} + \cdots + c_k\sqrt{a_k} - d_1\sqrt{b_1} - \cdots - d_k\sqrt{b_k}| > (2k\sqrt{n})^{-2^{|B|}+1}.$$

Since A generates itself, so this result recovers the best known lower bound on $r(n, k)$. In many cases, this result improves that bound, since $|B|$ can be smaller than $|A| = 2k$. It is possible that the cardinality of B can be as small as $O(\log k)$, in which case, there is a polynomial time algorithm comparing two sums of square roots.

Our result shows that the multiplicative structure of A affects the minimum possible value of $|c_1\sqrt{a_1} + c_2\sqrt{a_2} + \cdots + c_k\sqrt{a_k} - d_1\sqrt{b_1} - d_2\sqrt{b_2} - \cdots - d_k\sqrt{b_k}|$, which appears to be unknown before.

Proof: We use α to denote $c_1\sqrt{a_1}+c_2\sqrt{a_2}+\cdots+c_k\sqrt{a_k}-d_1\sqrt{b_1}-d_2\sqrt{b_2}-\cdots-d_k\sqrt{b_k}$. Suppose that $B=\{h_1,h_2,\cdots,h_{|B|}\}$. Since B multiplicatively generates A,

$$\mathbf{Q}(\sqrt{a_1},\sqrt{a_2},\cdots,\sqrt{a_k},\sqrt{b_1},\sqrt{b_2},\cdots,\sqrt{b_k})\subseteq\mathbf{Q}(\sqrt{h_1},\sqrt{h_2},\cdots,\sqrt{h_{|B|}}).$$

By induction, one knows that $\mathbf{Q}(\sqrt{h_1},\sqrt{h_2},\cdots,\sqrt{h_{|B|}})$ is a Galois extension of $\mathbf{Q}$ and

$$[\mathbf{Q}(\sqrt{h_1},\sqrt{h_2},\cdots,\sqrt{h_{|B|}}):\mathbf{Q}]\le 2^{|B|}.$$

Let G denote the Galois group of $\mathbf{Q}(\sqrt{h_1},\sqrt{h_2},\cdots,\sqrt{h_{|B|}})$ over $\mathbf{Q}$. We have $|G|\le 2^{|B|}$. For any field isomorphism $\sigma\in G$, we have $\sigma(\sqrt{a_i})=\pm\sqrt{a_i}$ for $1\le i\le k$, and $\sigma(\sqrt{b_i})=\pm\sqrt{b_i}$ for $1\le i\le k$. So

$$\begin{aligned}&|\sigma(c_1\sqrt{a_1}+c_2\sqrt{a_2}+\cdots+c_k\sqrt{a_k}-d_1\sqrt{b_1}-d_2\sqrt{b_2}-\cdots-d_k\sqrt{b_k})|\\ \le\ & c_1\sqrt{a_1}+c_2\sqrt{a_2}+\cdots+c_k\sqrt{a_k}+d_1\sqrt{b_1}+d_2\sqrt{b_2}+\cdots+d_k\sqrt{b_k}\\ \le\ & 2k\sqrt{n}.\end{aligned}$$

The norm of α is a nonzero integer. So $|norm(\alpha)|\ge 1$, that is

$$\prod_{\sigma\in G}\sigma(\alpha)\ge 1.$$

Hence

$$|\alpha|\ge\frac{1}{\prod_{\sigma\in G,\sigma\ne 1}|\sigma(\alpha)|}\ge(2k\sqrt{n})^{-2^{|B|}+1}.$$

□

The proof relies on the fact that the norm is a nonzero integer, thus has absolute value greater than 1. Every factor in the definition of the norm is not too large (less than $2k\sqrt{n}$ in our case), so the smallest factor should not be too small. The technique has been used in several papers, for example, see [4]. The estimation depends primarily on the number of factors in the definition of the norm.

Corollary 2. *Let $c_1,c_2,\cdots,c_k,d_1,d_2,\cdots,d_k$ be positive integers. Let $a_1,a_2,\cdots,a_k$, $b_1,b_2,\cdots,b_k$ be distinct square free positive integers less than m. Assume that $c_i^2a_i\le n$ for all $1\le i\le k$ and $d_i^2b_i\le n$ for all $1\le i\le k$. Then*

$$|c_1\sqrt{a_1}+\cdots+c_k\sqrt{a_k}-d_1\sqrt{b_1}-\cdots-d_k\sqrt{b_k}|>(2k\sqrt{n})^{-2^{O(m/\log m)}}.$$

Proof: It is well known that the number of primes less than m is $O(m/\log m)$. The set of primes less than m generates all the positive integers less than m. □

Corollary 3. $R(n,k)=2^{O(n/\log n)}$

5. Algorithm for finding $r(n, k)$

5.1. Related work

The use of heaps to enumerate sums in a sorted order appeared quite early [9, Section 5.2.3]. Let P be a sorted list of p real numbers whose i-th element is denoted by $P[i]$. Let Q be another sorted list of q real numbers whose i-th element is denoted by $Q[i]$. Consider the following way of enumerating elements of form $P[i] + Q[j]$ in a sorted order:

Algorithm 1.

```
Build a heap for $P[i] + Q[1]$, $1 \leq i \leq p$;
while the heap is not empty do
      Remove the element $P[i] + Q[j]$ at the root from the heap
      if $j < q$
         then put $P[i] + Q[j + 1]$ at the root of the heap
      endif
      reheapify.
endwhile
```

Note that for the program to work, one needs to keep track of the indexes i and j for the summation $P[i] + Q[j]$. The algorithm uses space to store $p + q$ elements but produces a stream of pq elements in a sorted order. Schroeppel and Shamir [13] applied this idea to attack cryptosystems based on knapsack. Number theorists have been using this idea as a space-saving mechanism to test difficult conjectures on computers. For example, consider the following Diophantine equation:

$$a^4 + b^4 + c^4 = d^4.$$

Euler conjectured that the equation had no positive integer solutions. It was falsified with a explicit counterexample by Elkies [8] using the theory of elliptic curves with help from a computer search. Bernstein [1] was able to find all the solutions with $d \leq 2.1 \times 10^7$. His idea was to build two streams of sorted integers, one for $a^4 + b^4$ and another one for $d^4 - c^4$, and then look for collisions. To find solutions with $d \leq H$, the algorithm needs only $H^{1+o(1)}$ space and runs in time $H^{2+o(1)}$. A similar idea can be used to find integers which can be written in many ways as summations of certain powers. Our approach is inspired by this work. Essentially we use heap to enumerate all the summations of form $\sum_{i=1}^{k} \sqrt{a_i}$ ($1 \leq a_i \leq n$ for all $1 \leq i \leq k$) and try to find the smallest gap between two consecutive elements. In our case, equality (i.e. gap = 0) is not interesting in the view of Proposition 1, while in the power summation applications, only equality (collision) is desired. There are other important differences:

- In the power sum case it will deal with only integers, while in our case, we have to deal with float-point numbers. The precision of real numbers plays an important role. Sometimes two equal sums of square roots can result in different float point numbers. For example, using `double double` type to represent real numbers, the evaluation of

  $$(\sqrt{1}+\sqrt{8}+\sqrt{8})+(\sqrt{24}+\sqrt{83}+\sqrt{83}+\sqrt{89})$$

  differs from the evaluation of

  $$(\sqrt{1}+\sqrt{6}+\sqrt{6})+(\sqrt{32}+\sqrt{83}+\sqrt{83}+\sqrt{89})$$

  by about 8×10^{-28}, even though they are clearly equal to each other.

- In the power sum case, p-adic restriction can often be applied to speed up the search, while unfortunately we do not have it here.

5.2. The algorithm

We first sketch the algorithm. It takes two positive integers n and k as input. Assume that $k < n$.

Algorithm 2. *Input: Two positive integers n, k ($n > k$).*

```
Store all the lists $(a_1, a_2, \ldots, a_A)$, where $1 \le a_1 \le a_2 \le \cdots \le a_A \le n$, into
        an array $P$, and then sort the array $P$ according to the sum $\sum_{i=1}^{A} \sqrt{a_i}$.
        Assume that there are $p$ elements in the list;
Store all the lists $(a_1, a_2, \ldots, a_{k-A})$, where $1 \le a_1 \le a_2 \le \cdots \le a_{k-A} \le n$,
        into an array $Q$, and then sort the array $Q$ according to the sum $\sum_{i=1}^{k-A} \sqrt{a_i}$.
        Assume that there are $q$ many elements in $Q$;
current_small_gap $= \infty$;
previous_smallest_element $= k$;
Build a heap for $(P[i], Q[1])$, $1 \le i \le p$, where two lists are compared according to
        the sum of square roots of the integers in the lists;
While the heap is not empty do
        Let $(P[i], Q[j])$ be the element at the root of the heap;
        current_top_element $= \sum_{l=1}^{A} \sqrt{P[i][l]} + \sum_{l=1}^{k-A} \sqrt{Q[j][l]}$
        if $0 <$ current_top_element $-$ previous_top_element $<$ current_small_gap
           then current_small_gap $=$ current_top_element$-$ previous_top_element;
        endif
        remove $(P[i], Q[j])$ from the heap;
        previous_top_element $= (P[i], Q[j])$;
        if there exist integers $j'$ such that $j < j' \le q$ and $P[i][A] \le Q[j'][1]$
           let $j'$ be the smallest one and put $(P[i], Q[j'])$ at the root
        endif
        reheapify
endwhile
Ouput $r(n, k) =$ current_small_gap
```

Note that in the above algorithm, unlike in Algorithm 1, we replace $(P[i], Q[j])$ at the root by $(P[i], Q[j'])$, which is not necessarily $(P[i], Q[j+1])$. In many cases, j' is much bigger than $j+1$. This greatly improves the efficiency of the algorithm. Now we prove the correctness of the algorithm.

Theorem 3. *When the algorithm halts, it outputs $r(n,k)$;*

Proof: For any $1 \leq a_1 \leq a_2 \cdots \leq a_A \leq n$, define

$$S_{a_1,a_2,\cdots,a_A} = \{(a_1, a_2, \cdots, a_k) | a_A \leq a_{A+1} \leq a_{A+2} \leq \cdots \leq a_k \leq n\}$$

Partition the set

$$S = \{(a_1, a_2, \cdots, a_k) | 1 \leq a_1 \leq a_2 \leq \cdots \leq a_k \leq n\}$$

into subsets according to the first A elements, namely,

$$S = \bigcup_{1 \leq a_1 \leq a_2 \leq \cdots a_A \leq n} S_{a_1,a_2,\cdots,a_A}.$$

As usual, we order two lists of integers by their sums of square roots. Consider the following procedure: select the smallest element among all the the minimum elements in all the subsets, and remove it from the subset. If we repeat the procedure, we generate a stream of elements in S in a sorted order.

It can be verified that in our algorithm, the heap consists of exactly all the minimum elements from all the subsets. The root of the heap contains the minimum element of the heap. After we remove the element at the root, we put the next element from its subset into the heap. Hence the algorithm produces a stream of elements from S in a sorted order. The minimum gap between two consecutive elements in the stream is $r(n,k)$ by definition. □

Theorem 4. *The algorithm runs in time at most $n^{k+o(k)}$ and space at most $n^{\max(A,k-A)+o(k)}$.*

Proof: Using the root separation bound, we need at most $O(2^{2k} \log n)$ bit to represent a sum of square roots for comparison purposes. So comparing two elements takes time $(2^{2k} \log n)^{O(1)}$. Since every element in S appears at the root of the heap at most once and $|S| \leq n^k$, the main loop has at most n^k iterations. For each iteration, the time complexity is

$$(2^{2k} \log n)^{O(1)} \log(n^A).$$

The complexity of other steps are much smaller comparing to the loop. Hence the time complexity is $n^{k+o(k)}$. The space complexity is clearly $n^{\max(A,k-A)+o(k)}$.
□

6. Numerical Data and Observations

To implement our algorithm, the main issue is to decide the precision when computing the square roots and their summations. We need to pay attention to two possibilities:

- First, two summations may be different, but if the precision is set too small, then they appear to be equal numerically. Keep in mind that we have not ruled out that $r(n, k)$ can be as small as n^{-2^k}.
- Secondly two expressions may represent the same real number, but after the numerical calculation, they are different. This is the issue of numerical stability.

In either case, we may get a wrong $r(n, k)$. Our strategy is to set the precision at about $2k \log n$ decimal digits. For example, to compute $r(100, 7)$, we use the data type which has precision about 32 decimal digits. Whenever the difference of two summations is smaller than $k^2 n^{-2k}$, we call a procedure based on Proposition 1 to decide whether the two numbers are equal or not.

We produce some statistics data about the sums of square roots and the gaps between two consecutive sums. The computation takes about 18 hours on a high-end PC. There are 17940390852 real numbers in $[7, 100]$ which can be written as summations of 7 square roots of positive integers less than 100. Hence there are 17940390851 gaps between two consecutive numbers after we sort all the sums.

In Table 1, we list an integer $7 \leq a \leq 70$ with the number of reals in $[a, a+1)$ which can be represented as $\sqrt{a_1}+\sqrt{a_2}+\cdots+\sqrt{a_7}$ ($1 \leq a_1 \leq a_2 \cdots \leq a_7 \leq 100$). Note that if two summations have the same value, they are counted only once. From the table, we see that there are 1163570911 sums in the $[48, 49)$, which gives us a more precise pigeonhole upper bound for $r(n, k)$ at $1/1163570911 = 8.6 \times 10^{-10}$, which is still several magnitudes away from $r(n, k)$. On the other hand, it is known that

Proposition 2. *If one selects N random points uniformly from a line segment of length L, then two closest points has expected distance $\frac{L}{N^2-1}$.*

We can see that $r(n, k)$ is very close to the value predicted by the above proposition.

In Table 2, for each range, we list the number of gaps between consecutive numbers in the range. From the table, we see that there are 7 gaps which have magnitude at 10^{-19}.

7. Conclusion remarks

In this paper, we prove an upper bound of $2^{O(n/\log n)}$ for $R(n, k)$, by exploring the fact that the algebraic degree of sum of $2k$ square free positive integers can be much less than 2^{2k}. We suspect that $2^{O(k/\log k)} \log n$ type of upper bound

Table 1: Statistics on the summations of square roots

7	**8**	**9**	**10**	**11**	**12**
4	17	57	161	418	1003
13	**14**	**15**	**16**	**17**	**18**
2259	4865	10044	20061	38742	72903
19	**20**	**21**	**22**	**23**	**24**
133706	239593	420279	722739	1218852	2017818
25	**26**	**27**	**28**	**29**	**30**
3280805	5239096	8218857	12664315	19165803	28482325
31	**32**	**33**	**34**	**35**	**36**
41554376	59503519	83607939	115241837	155784865	206478894
37	**38**	**39**	**40**	**41**	**42**
268254403	341520055	425961992	520334126	622307266	728445926
43	**44**	**45**	**46**	**47**	**48**
834229563	934295227	1022797808	1093860379	1142175328	1163570911
49	**50**	**51**	**52**	**53**	**54**
1155526520	1117588507	1051539385	961294902	852549403	732208073
55	**56**	**57**	**58**	**59**	**60**
607649679	486014737	373475729	274666260	192383944	127511613
61	**62**	**63**	**64**	**65**	**66**
79264404	45637971	23914891	11119037	4410314	1398655
67	**68**	**69**	**70**		
316043	40172	1476	1		

Table 2: Statistics about the gaps

$\mathbf{10^{-19} \sim 10^{-18}}$	$\mathbf{10^{-18} \sim 10^{-17}}$	$\mathbf{10^{-17} \sim 10^{-16}}$	$\mathbf{10^{-16} \sim 10^{-15}}$	$\mathbf{10^{-15} \sim 10^{-14}}$
7	47	1245	14139	129248
$\mathbf{10^{-14} \sim 10^{-13}}$	$\mathbf{10^{-13} \sim 10^{-12}}$	$\mathbf{10^{-12} \sim 10^{-11}}$	$\mathbf{10^{-11} \sim 10^{-10}}$	$\mathbf{10^{-10} \sim 10^{-9}}$
1459473	13100265	132767395	1272832428	8256755966
$\mathbf{10^{-9} \sim 10^{-8}}$	$\mathbf{10^{-8} \sim 10^{-7}}$	$\mathbf{10^{-7} \sim 10^{-6}}$	$\mathbf{10^{-6} \sim 10^{-5}}$	$\mathbf{10^{-5} \sim 10^{-4}}$
7766837445	463570895	30415764	2314151	176109
$\mathbf{10^{-4} \sim 10^{-3}}$	$\mathbf{10^{-3} \sim 10^{-2}}$	$\mathbf{10^{-2} \sim 10^{-1}}$	$\mathbf{10^{-1} \sim 1}$	
14890	1300	80	5	

holds for much larger n, and leave it as an open problem. We have also proposed a space-efficient algorithm to compute $r(n,k)$ exactly. Our numerical data seem to suggest that the upper bound is closer to the truth than the root separation bounds. Further investigations, both experimental and theoretical, are needed.

References

[1] Daniel Bernstein. Enumerating solutions to $p(a)+q(b) = r(c)+s(d)$. *Math. of Comp.*, 70:389–394, 2001.

[2] Peter Borwein. *Computational Excursions in Analysis and Number Theory.* Springer-Verlag, 2002.

[3] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. A strong and easily computable separation bound for arithmetic expressions involving radicals. *Algorithmica*, 27(1):87–99, 2000.

[4] Zhi-Zhong Chen and Ming-Yang Kao. Reducing randomness via irrational numbers. *SIAM J. Comput.*, 29(4):1247–1256, 2000.

[5] Qi Cheng. On comparing sums of square roots of small integers. In *Proc. of 31st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 4162 of *Lecture Notes in Computer Science*, 2006.

[6] Qi Cheng and Yu-Hsin Li. Finding the smallest gap between sums of square roots. In *Proceedings of The 9th Latin American Theoretical Informatics Symposium (LATIN)*, volume 6034 of *Lecture Notes in Computer Science*, pages 446–455, 2010.

[7] Erik D. Demaine, Joseph S. B. Mitchell, and Joseph O'Rourke. The open problems project: Problem 33. http://maven.smith.edu/˜orourke/TOPP/.

[8] Noam Elkies. On $a^4 + b^4 + c^4 = d^4$. *Math. of Comp.*, 51:825–835, 1988.

[9] Donald Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1973.

[10] Francesco Pappalardi. A survey on k-power freeness. In *Proceeding of the Conference in Analytic Number Theory in Honor of Prof. Subbarao*, number 1 in Ramanujan Math. Soc. Lect. Notes Ser., pages 71–88, 2002.

[11] Jianbo Qian and Cao An Wang. How much precision is needed to compare two sums of square roots of integers? *Inf. Process. Lett.*, 100(5):194–198, 2006.

[12] Arnold Schönhage. On the power of random access machines. In *Proc. 6th Internat. Colloq. Automata Lang. Program.*, volume 71 of *Lecture Notes in Computer Science*, pages 520–529. Springer-Verlag, 1979.

[13] Richard Schroeppel and Adi Shamir. A $T = o(2^{n/2})$, $S = o(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM journal on Computing*, 10(3):456–464, 1981.

[14] A. Shamir. Factoring numbers in $O(\log n)$ arithmetic steps. *Information Processing Letters*, 1:28–31, 1979.