*CSC 495 – Problem Set 9 – Due Tuesday, March 27*

# Problem 9.R1: Caution While Merging

*Required Problem*
**Points:** 50 points
**20 Extra Ninja Points** for the fastest solution (several tests will be run with various sizes — if no solution is consistently fastest, the points will be split)

### Background

In a lot of cases, your goal is to sort data, and the data has some characteristics that you can take advantage of. For example, consider a scenario in which a company has many field offices collecting information, which they each provide in a sorted file to the central office. The central office wants to combine these reports into a single, sorted report. Clearly, you could just throw all of the data into a big array and sort, but can you do better?

Specifically, consider a case with $k = 8$ offices submitting reports that contain $m = 1,000,000$ records each, for a total of $n = mk = 8,000,000$ records. Storing all of those in a single array and sorting would take $\Theta(n \log n)$ time. However, since the branch offices have done a lot of the sorting work already, there is an algorithm that uses $O(n \log k)$ time. Since $\log_2 k = 3$ and $\log_2 n \approx 23$ (and the constants in the times are comparable), the better solution is almost 8 times faster.

### The Problem to Solve

You are to read in blocks of sorted data, where each record is a single line and sorting is done treating the lines as strings. You should output the entire collection of records in sorted order.

### Hints and Techniques

"Correctness" in this problem is defined simply by having the correct sorted output, so you could in fact get all correctness points simply by putting everything into an array and sorting. However, that does not take advantage of the characteristics of the input data and hence is not a particularly elegant solution — and it stands no chance of being one of the fastest solutions. For a better solution, think about mergesort — this should give an obvious idea for two branch offices, so how can you generalize this? Can you generalize the merge process so that it can merge $k$ lists with $O(\log k)$ cost per output element?

### Input and Output

Input will consist of an integer $n$ representing the number of blocks, followed by the $n$ blocks: block $i$ (for $i = 1, \ldots, n$) is given as an integer $m_i$ followed by $m_i$ lines that are the records from that office/block. Records within a block are sorted. You should output all records in sorted order.

**Sample Input**

```
4
3
Alpha
Beta
Gamma
2
Classy
Doughnut
3
Acid
Database
Operations
2
Computer
Fixer
```

**Sample Output**

```
Acid
Alpha
Beta
Classy
Computer
Database
Doughnut
Fixer
Gamma
Operations
```

## Problem 9.R2: How Quick is Quicksort

*Required Problem*
**Points:** 50 points

### Background

Column 11 in *Programming Pearls* describes "Lomuto Partitioning" for Quicksort, but always uses the first element in the array/subarray as the pivot value for partitioning. Unfortunately, if you pick the first element as the pivot, an array that is already sorted (or almost sorted) results in worst-case $\Theta(n^2)$ time behavior. Alternatively, you could pick the last element, but that exhibits worst-case behavior when the input is in reverse order. In order to avoid these well-structured worst-cases, one variant of Quicksort does the following: First, extract the first, last, and middle element of the array, where "middle" is the average of the first and last indices rounded down (so if you are looking at subarray $x[lo..hi]$, then the middle element is $x[\lfloor \frac{lo+hi}{2} \rfloor]$). Once the median of these three elements is found, it is swapped to the first position in the array, and then the regular Lomuto partitioning is performed, followed by the regular Quicksort recursion. Note that with this new algorithm, random inputs are still handled efficiently, and the previous worst-case inputs (sorted and reverse sorted inputs) exhibit best-case performance. Can you implement this and count how many comparisons are made for a given input?

In order to count precisely and consistently, everyone needs to use the same median-of-3 code, provided below:

```
int pivotIdx = lo;
int mid = (lo+hi)/2;
if (x[lo] < x[mid]) {
    if (x[mid] < x[hi])
        pivotIdx = mid;
    else if (x[lo] < x[hi])
        pivotIdx = hi;
    else
        pivotIdx = lo;
} else {
    if (x[lo] < x[hi])
        pivotIdx = lo;
    else if (x[mid] < x[hi])
        pivotIdx = hi;
    else
        pivotIdx = mid;
}
```

For Lomuto partitioning and Quicksort, see the pseudocode in *Programming Pearls* (pages 118 and 119). Note that you must have at least 3 elements in the array to pick the pivot this way, so the base case of your Quicksort recursion should handle sizes 1 and 2 directly.

### The Problem to Solve

You should read in a list of $n$ integers (guaranteed to all be distinct), and output the number of comparisons that this variant of Quicksort will make in processing this input. The output is just the number of comparisons (you do not have to output the sorted data). You are guaranteed that $n \leq 1,000,000$.

### Hints and Techniques

Be careful to use exactly the median selecting algorithm given above, and an accurate coding of Lomuto partitioning. Seemingly small changes can affect the number of comparisons made, resulting in the wrong answer, so be very careful with your coding and comparison counting.

### Input and Output

Input consists of an integer $n$, representing the number of items in the array to sort, followed by $n$ lines, each containing an integer value. The values are guaranteed to all be distinct. Your program should output a single line containing the number of comparisons made by this variant of Quicksort.

**Sample Input 1**

```
10
9
2
4
7
6
10
1
8
3
5
```

**Sample Output 1**

```
30
```

**Sample Input 2**

```
6
2
3
4
5
6
1
```

**Sample Output 2**

```
15
```

# Problem 9.C1: Evil Generator for Quicksort

*Challenge Problem*
**Ninja Points:** This challenge problem is worth up to 20 base ninja points

### Background

The Quicksort variant described in Problem 9.R2 makes it so that some typical worst-case behaviors of the most basic Quicksort implementation are avoided. However, the pivot selection rule is still deterministic, and every simple deterministic variant of Quicksort has a worst-case behavior that degenerates to $\Theta(n^2)$ time. Given a value $n$, can you find an input that exhibits this worst-case behavior for the Quicksort described in Problem 9.R2?

### The Problem to Solve

You are given an integer $n$, and need to compute a permutation of $1, \ldots, n$ that requires a worst-case number of comparisons when fed into the Quicksort variant described in Problem 9.R2. You are guaranteed that $n \leq 1,000,000$, and your program should be able to compute this worst-case permutation in under 15 seconds.

### Hints and Techniques

Think carefully about how to construct the worst-case input. To help you check your program, consider this additional information: Let $T(n)$ represent the function that gives this worst-case number of comparisons for any value $n$, and then

$$T(n) = \begin{cases} \frac{1}{4}n^2 + \frac{3}{2}n - 3 & \text{if } n \text{ is even;} \\ \\ \frac{1}{4}n^2 + \frac{3}{2}n - \frac{7}{4} & \text{if } n \text{ is odd.} \end{cases}$$

Note that even though these formulas contain fractions, the value of $T(n)$ always works out to be an integer for all integer values of $n$ (try it if you don't believe me!). Note also how this number of comparisons grows as $\Theta(n^2)$, reflecting the bad performance in the worst case.

You can use this formula with your solution to Problem 9.R2 in order to test your solution to this problem. Pick a value of $n$, run this program to generate a worst-case permutation, and then use that as input to your program for Problem 9.R2 to count the actual number of comparisons required for that permutation. Then calculate $T(n)$ using the formula above, and see if it matches the output of your Problem 9.R2 program. If it matches, that's great! If your output is smaller than the calculated $T(n)$ then it means you haven't found the worst case. If your output is *bigger* than $T(n)$ then either (a) you have a bug in your Problem 9.R2 program, or (b) I messed up in figuring out $T(n)$ or have another error in this problem statement. If you feel confident that (b) is the issue, contact me!

### Elegance Considerations

Put in some brief comments describing why your program produces a worst-case permutation. Don't just guess at a technique, code it up, and hope that you're lucky — give some reasoning!

### Input and Output

Input consists of a single line with an integer $n$. Your program should output $n$ lines, each containing an integer in the range $1, \ldots, n$, so that these lines give a permutation that forces worst-case performance of this Quicksort variant. Note that the output is not necessarily unique: there are multiple permutations that force worst-case behavior, and any of these will be judged correct.

In the sample below, note that the "Sample Output" matches the "Sample Input 2" in Problem 9.R2. And note that if you plug in $n = 6$ to the formula above, you get $T(6) = 15$, matching the "Sample Output 2" in Problem 9.R2. Isn't that cool?

**Sample Input**

```
6
```

**Sample Output**

```
2
3
4
5
6
1
```

# Problem 9.C2: Cubist Delight

*Challenge Problem*
**Ninja Points:** This challenge problem is worth up to 20 base ninja points

### Background

Fermat's Last Theorem is one of the great stories in mathematics. The statement of the theorem is simple: There is no solution to $a^n + b^n = c^n$ for integers $a, b, c \geq 1$ and $n > 2$. Fermat often wrote about mathematical topics and wrote proofs of the statements in the margins, but for this particular statement he noted that he had a proof, but it was too large to fit in the margin. That was in 1637, and for the next 350 years mathematicians tried to prove this statement. Finally, in 1995, Andrew Wiles (a mathematician at Princeton University) published a proof that has withstood scrutiny — it was actually his second proof, with the first one containing an error. Wiles' proof is over 100 pages long, and far from something that Fermat could have had in mind in 1637. Did Fermat really have a proof? Is there a simpler proof waiting to be discovered? These questions keep this story alive, even after the truth of the statement itself has been proved.

Several special cases of Fermat's Theorem were proven long before Wiles' general result. For example, in 1770 the famous mathematician Leonard Euler proved that no integers $a, b, c \geq 1$ could satisfy $a^3 + b^3 = c^3$ (there was an error in his original proof, but that's yet another story). There are many problems similar to Fermat's Last Theorem that are interesting to consider, some of which are unsolved to this day. In this problem we consider one related problem computationally. In particular, can we find $a, b, c, d \geq 1$, with $\{a, b\} \neq \{c, d\}$ such that

$$a^3 + b^3 = c^3 + d^3$$

(note the set notation — this excludes trivial solutions in which $a = d$ and $b = c$)? There are many such values: for example, if $a = 1$, $b = 12$, $c = 9$, and $d = 10$, we have

$$1^3 + 12^3 = 1729 = 9^3 + 10^3.$$

Can you find all such $\{a, b\}, \{c, d\}$ where $a^3 + b^3 \leq n$ for some bound $n$?

Note that there are some similar problems that are still unsolved, and could potentially be solved computationally. As an example, look up the "Beal Conjecture" — either a proof or a counter-example (which could found computationally) would earn the solver a prize of $100,000.

### The Problem to Solve

You are given an integer $n$ such that $1 \leq n \leq 10^{15}$, and your program must find all integers $x \leq n$ that can be expressed as the sum of two cubes of positive integers in multiple ways. The smallest such integer is 1729 (as shown above). Your program should handle $n = 10^{15}$ in under 60 seconds.

### Hints and Techniques

A common way to look for duplicate values in a list is to first sort the list so that duplicates are adjacent to one another. However, in some cases the list being dealt with is so large that storing

and sorting it would require too much resources (space and time). This is one of those cases. Expanding the bounds above a little, if $n$ could be $10^{18}$ then there are over a billion sums to consider, each requiring 64 bits (8 bytes) to store, requiring over 8 gigabytes of memory. And sorting such an array would take many minutes. However, my solution to this problem handles $n = 10^{18}$ in under 3 minutes using under 8 megabytes of memory.

The trick is to generate values in sorted order, rather than generating and then sorting. Think about how a heap (i.e., priority queue) might help in this situation. In fact, in a very general sense, this problem is similar to Problem 9.R1.

### Input and Output

The input consists of a single integer $n$, where $n \leq 10^{15}$. The output should consist of all integers $\leq n$ which can be expressed in more than one way as sums of cubes of integers. The output should be given in groups of formulas that have the same sum, separated by blank lines, and sorted as follows: first, groups are sorted by sum, with each formula expressed with the smaller integer first, and formulas within a group ordered by this first number. See the sample below for formatting and an example of ordering.

**Sample Input**

```
50000
```

**Sample Output**

```
1729 = 1^3 + 12^3
1729 = 9^3 + 10^3

4104 = 2^3 + 16^3
4104 = 9^3 + 15^3

13832 = 2^3 + 24^3
13832 = 18^3 + 20^3

20683 = 10^3 + 27^3
20683 = 19^3 + 24^3

32832 = 4^3 + 32^3
32832 = 18^3 + 30^3

39312 = 2^3 + 34^3
39312 = 15^3 + 33^3

40033 = 9^3 + 34^3
40033 = 16^3 + 33^3

46683 = 3^3 + 36^3
46683 = 27^3 + 30^3
```