Очень простая задача.

Решение.

Да, действительно, очень простая. Особенно, если бы диапазон возможных чисел состоял из тысячи-другой или даже из нескольких миллионов чисел. Тогда бы мы просто читали числа, отмечали бы в некотором массиве, что такое число уже было, и, если такое число раньше не было отмечено, то выводили бы его. Однако технические ограничения совсем не такие: диапазон возможных чисел растянулся на 4 миллиарда – это много, и вышеописанный подход не сработает. Даже если на метку «было/не было» некоторое число тратить не целый байт, а только один бит.

Что же делать? Чисто интуитивно кажется, что можно найти алгоритм, требующий асимптотически O(N) операций (где N – количество чисел). Нет, не получится. Чтобы убедить себя в этом воспользуемся одним известным утверждением: *любой алгоритм, определяющий, являются ли все элементы массива из N чисел различными, требует $\Omega(N \cdot \log N)$ проверок*. Запись $\Omega(N \cdot \log N)$ означает, что в асимптотике проверок потребуется не меньше, чем $O(N \cdot \log N)$, а само утверждение выполняется при очень общих предположениях о применяемых алгоритмах.

Этот факт применим очень часто, доказывать его здесь неуместно, но использовать его будем. В самом деле, допустим мы сумели решить нашу исходную задачу быстрее, чем за $O(N \cdot \log N)$ операций. Тогда выполним следующие действия:

1. подсчитаем количество чисел во входном файле
2. решим нашу исходную задачу
3. подсчитаем количество чисел в выходном файле.

Если два подсчитанных количества совпадают, то среди данных чисел нет повторов, иначе – повторы есть. Первый и третий шаги требуют O(N) операций, второй – меньше, чем $O(N \cdot \log N)$. Итого, получается, что мы можем определить, являются ли все введенные числа различными, быстрее, чем за $O(N \cdot \log N)$. А это не так. Полученное противоречие и показывает, что лучше, чем $O(N \cdot \log N)$, искать не надо.

Ну, а тогда все легко: отсортируем введенные числа за $O(N \cdot \log N)$, выбросим повторы и выведем оставшиеся числа.

Идея ясна, осталось продумать детали. Из нескольких повторяющихся чисел нам надо выводить то, которое не встречалось раньше во входном файле. Значит, чтобы отличать одинаковые числа, можно хранить не только сами числа, но и номера их мест во входном файле. Вот всё и встало на свои места:

1. Считываем числа и храним их вместе с номерами их мест во входном файле.
2. Сортируем числа.
3. Из всех одинаковых чисел оставляем только то, у которого номер места наименьший. Для этого при сортировке будем считаем, что если два числа одинаковы, то раньше должно идти число с меньшим номером места. Тогда при вычеркивании оставляем среди всех одинаковых чисел первое из них.
4. Сортируем оставшиеся числа по номерам мест.
5. Выводим полученный ряд чисел.

Разумеется, вторая сортировка совсем не обязательна. Также, как можно обойтись и без хранения номеров мест. Можно решать задачу, например, так: в одном массиве хранить числа в том порядке, в каком они идут во входном файле, в другом массиве - те же числа, но отсортированные; затем брать поочерёдно числа из первого массива и искать их во втором массиве с целью проверить, выводили ли мы уже такое число (понадобится ещё массив меток "было-не было", конечно). Каждый поиск требует O(log N) операций, т.е. сложность получается та же самая O(N logN).

Можно изогнуться, и вовсе обойтись без сортировки (при той же сложности алгоритма), однако это уже не очень естественно. Всё-таки, проверка уникальности (или проверка повторяемости) и сортировка идут рядом. Так что в данной задаче, видимо, применение сортировки - есть самое естественное решение. И очень-очень во многих других...

Я позволю себе привести не очень короткую цитату из очень интересной книги Steven S. Skiena*. The Algorithm Design Manual*:

By the time they graduate, computer science students are likely to have studied the basic sorting algorithms in their introductory programming class, then in their data structures class, and finally in their algorithms class. Why is sorting worth so much attention? There are several reasons:

- *Sorting is the basic building block around which many other algorithms are built. By understanding sorting, we obtain an amazing amount of power to solve other problems.*
- *Historically, computers have spent more time sorting than doing anything else. A quarter of all mainframe cycles are spent sorting data [Knu73b]. Although it is unclear whether this remains true on smaller computers, sorting remains the most ubiquitous combinatorial algorithm problem in practice.*
- *Sorting is the most throughly studied problem in computer science. Literally dozens of different algorithms are known, most of which possess some advantage over all other algorithms in certain situations. To become convinced of this, the reader is encouraged to browse through [Knu73b], with hundreds of pages of interesting sorting algorithms and analysis.*
- *Most of the interesting ideas used in the design of algorithms appear in the context of sorting, such as divide-and-conquer, data structures, and randomized algorithms.*

Под **Knu73b** здесь подразумевается ссылка на 3-й том "Искусства программирования" Д. Кнута (D. E. Knuth. *The Art of Computer Programming, Volume 3)*

Полностью HTML-версия книги Steven S. Skiena*. The Algorithm Design Manual.* доступна в сети по адресу:

http://www2.toki.or.id/book/AlgDesignManual/BOOK/BOOK/BOOK.HTM

Но вернёмся к нашей “Очень простой задаче”. В приведённой программе используется дважды один и тот же алгоритм - QuickSort. Ни в малейшей степени не отрицая достоинств других быстрых алгоритмов сортировки (например, MergeSort, или Сортировка слиянием, или HeapSort, она же Пирамидальная сортировка или

Сортировка кучей), скажу с редкой определённостью: глубокое понимание и уверенное применение QuickSort, и вообще такого "разделяющего" подхода, необходимы для любого программиста, и, уж безусловно, - для олимпиадника.

Я здесь не буду описывать QuickSort (и другие алгоритмы сортировки) – это ни к чему, имеется много прекрасных описаний. О них-то речь и пойдёт ниже...

Литературы о сортировках имеется угрожающе много. Много очень хороших и интересных книг и статей - выбрать что-то одно чрезвычайно трудно. Тем не менее, я рискну и выложу в учебных материалах несколько отрывков из классических книг:

1. Вирт Н. *Алгоритмы и структуры данных*. М.: Мир, 1989. Глава 2, пункты 1-3. – Вирт – это Вирт, и этим всё сказано.
2. Ахо А., Хопкрофт Дж., Ульман Дж. *Построение и анализ вычислительных алгоритмов.* М.: Мир, 1979. Глава 3, пункты 4-5. Здесь описаны два алгоритма сортировки: HeapSort и QuickSort.
3. Бентли Дж. *Жемчужины программирования*, 2-е изд. СПб.: Питер, 2002. Глава 11. Основное внимание в этой главе посвящено QuickSort'у – подробнейшее и блестящее изложение и обсуждение.
4. Bentley J*. Programming pearls*. Addison-Wesley, 1986. Column 10. – оригинальный текст из той же книги.

В интернет-океане проблема выбора ещё сложнее. Поиск по слову «сортировка» в Google дал мне только что "примерно 13 800 000" результатов, а по слову "sorting" - "примерно 42 100 000" результатов. Я ограничусь двумя интересными ссылками:

1. http://program.rin.ru/razdel/html/765.html
   Эта страничка приятно выделяется тщательностью проработки материала.

2. http://rain.ifmo.ru/cat/view.php/vis/sorts
   А здесь имеются визуализаторы работы очень многих алгоритмов, в том числе и сортировок.