Предложите исправление представленого ниже кода

FILE *f = fopen(name, “w”);

// Тут код, при котором может быть выброшено исключение

fclose(f);

Предлагаемые исправления

```
#include <cstdio>
#include <iostream>
#include<errno.h>

try {
		FILE *f = fopen("file", "w");
		if (f == NULL)
			throw errno;

		try {
			int err_code;
			//Тут код, при котором может быть выброшено исключение
			if(err_code==-1)
				 throw errno;
		}

		catch (int err)
		{
			//Вероятные ошибки при записи файла
			if(err == ENOSPC)
				std::cout << "Недопустимый аргумент." << std::endl;
			if(err == EINVAL)
				std::cout << "Нет места на выведенном устройстве" << std::endl;
			else
				std::cout << err << std::endl;
			//Вероятных ошибок может быть и больше, всё зависит от используемых
функций
		}

		int e = fclose(f);
		if (errno == EBADF || errno == EIO || e == -1)
			throw e;
	}

	catch (int a)
	{
		if(a == EMFILE)
			std::cout << "Открыто максимальное количество файлов" << std::endl;
		if(a == ENOENT)
			std::cout << "Неверный путь" << std::endl;
		if(a == EACCES)
			std::cout << "Доступ запрещен" << std::endl;
		if(a == EBADF)
			std::cout << "Неверный дескриптор файла" << std::endl;
		if(a == EIO)
			std::cout << "Произошла ошибка ввода / вывода" << std::endl;
		else
			std::cout << "Произошла неизвестная ошибка" << std::endl;
	}
```

Предложите алгоритм для удаления дубликатов (или выбора уникальных элементов) из вектора. Оцените временную и пространственную сложность.

Самый наивный алгоритм удаление дубликатов – это полный перебор и копирование уникальных элементов в новый вектор. Его временная сложность $O(N^2)$, сложность по памяти $(N + K)$, N – размер исходного вектора, K – размер нового вектора с уникальными элементами.

Быстрее чем $O(N)$ выбрать уникальные элементы нельзя т.к. нужно проверить весь вектор. Для ускорения поиска можно использовать бинарное дереве поиска, а именно сбалансированное по высоте АВЛ-дерево. Поиск по дереву равен $O(logN)$.

Предлагается следующий алгоритм:

1. Инициализируем дерево, корень 0 элемент вектора
2. Идём по вектору и ищем каждый элемент в дереве
3. Если элемент не найден, то добавляем его в дерево
4. Если элемент найден, то пропускаем его и переходим к шагу 2
5. Продолжаем пока не переберём все элементы вектора

На выходе мы получаем дерево, в котором все элементы уникальны, далее можно сохранить все элементы дерева в вектор.

Оценка сложности по времени $O(NlogN)$, по памяти $O(N + K)$, где N – размер исходного вектора, K – размер дерева.

Сколько различных изображений можно получить из изображения NxN, если каждое следующее получается путем уменьшения предыдущего в 2 раза? В n раз?

$[\log_n N]$, где n – во сколько раз уменьшаем изображение, а N – начальный размер изображения.

Докажем:

В ходе получения каждого нового изображения производится деление на 2, пока T не равен 2 или меньше 2.

$T = N/2$

$T = (N/2)/2$

$T = ((N/2)/2)/2$

При известном количестве итераций можно восстановить исходное или приблизительное N.

$$N = \big((T * 2) * 2\big) * 2$$

$$N = (T * 2) * 2$$

$$N = T * 2$$

Из приведённых вычислений выше можно заметить, что вычисления выше нечто иное как возведение в степень, из этого следует что приведение к приблизительному исходному размеру равно,

$N \sim T * 2^i$, где T – минимальный размер изображения, i – количество итераций (полученных изображений)

Обратной же операцией возведению в степень является логарифмирование, из чего следует что

$$i = [\log_2 N]$$

Данный способ позволяет быстро вычислить приблизительное количество получаемых изображений, можно использовать следующий алгоритм:

```
i = log(2,N)
temp = N-(N/pow(2,l))
if(temp>2)
{
   do{
       temp=temp/2
       i++
   }while(temp>2)
}
return i
```

В место 2 можно использовать любое другое число, не превышающее N.