Задача 1.

Тип double гарантирует хранение без потерь 15 значащих десятичных разрядов. Операция сложения вещественных чисел разных порядков теряет точность в зависимости от разности порядков. Таким образом, когда в переменной a будет накоплено значение с разницей порядков = 15, то есть степень a = -3, то дальнейшие вычисления не дадут изменения ни в одном из значащих разрядов, следовательно, суммирование прекратится. Конкретно на моей конфигурации компьютера такая деградация произошла при порядке одной десятой(-1), что не меняет общей картины. К примеру, 1.0e-1 + 1.0e-18 уже не даст никакого прироста, так как в правом операнде все цифры будут сдвинуты вправо на расстояние, превышающее количество значащих разрядов. Таким образом, a никогда не приблизится к 1, то есть в каждом выводе, в том числе и на строке 2.0e18, будет округлено до 0.
Еще хочется упомянуть, что при длинных суммах чисел одного порядка следует использовать алгоритм Кэхэна, чтобы снизить погрешность вычисления.

Задача 2.

Расстояние между двумя точками в пространстве определяется формулой:
$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$
Результат вычислений не должен превышать eps, что также означает, что если хотя бы одна из разностей координат превосходит eps, то и результат всей формулы тоже превысит eps.
Значит, если в паре точек найти такую пару координат, то точки не повторяющиеся. Таким образом можно отсеять часть повторяющихся точек, не доходя до полного вычисления расстояния. Чтобы воспользоваться данным преимуществом, сделаем три прогона отсева точек, сравнивая только координаты x, y, z в каждом прогоне соответственно. Каждый прогон займет (n^2 + n)/2 операций, что не отличается порядком от простого сравнения всех координат со всеми, но улучшает лучший случай в три раза, а также не требует дополнительной операции извлечения корня, что я бы отметил важным преимуществом, так как мы избегаем потери в точности.
После отсева остаются только подозрительные на повторяемость точки, для которых придется посчитать полное расстояние по формуле, что также оценивается по сложности как O((n^2 + n) / 2) – сравнение каждой пары с каждой, исключая предыдущие.
Следует отметить, что операцию сравнения не следует производить с eps так, как мы бы делали в случае с целочисленными типами. Следует сравнивать eps с относительной погрешностью операции вычитания.

Задача 3.

ASCII-формат использует алфавит из 256 элементов, что совсем немного при объеме данных в 100 Гб. Используем сортировку подсчетом.
Выделим вспомогательный целочисленный массив длиной 256 элементов, заполним его нулями. Строку при таком объеме целесообразно читать с помощью потока, считывая один символ, то есть один байт за раз. Будем вычислять номер считанного символа в ASCII-таблице, обращаться по данному номеру во вспомогательный массив и увеличивать счетчик на 1. Временная сложность оценивается как линейная, то есть растет прямо

пропорционально увеличению объема данных. Сложность может возникнуть с переполнением счетчика символов, так в худшем случае строка состоит из одного и того же символа, что приведет к: 100 Гб = 100 * 2^30 байт, порядок приблизительно 2^37, то есть unsigned int (2^32) уже не подойдет, зато подойдет long long int (2^63). Если строка станет слишком большой для имеющихся типов данных, то следует использовать длинную арифметику.

Задача 4.

Все предложенные варианты могут быть использованы, но вот мои соображения: чтобы избавиться от копирования v по значению, будем передавать по ссылке, что может снизить затраты на память при объемных векторах. Но в таком случае вектор становится незащищенным для изменения. Так как не подразумевается изменение значения v в теле функции, то следует указать модификатор const, который предостережет от попытки его изменить, также сигнатура функции становится информативнее.
Ответ в.

Задача 5.

Можно ускорить прохождение цикла за счет итератора, также следует оставить только одно обращение к текущему элементу.

```
double length2(const std::vector<double> &v) {
	using vector_iterator = std::vector<double>::const_iterator;
	double res = 0;
	for (vector_iterator vi = v.begin(); vi != v.end(); vi++)
	{
		res += *vi * *vi;
	}
	return res;
}
```

Задача 6.

Тип double хранит данные в двоичном представлении, как и все в памяти. 1/4 является степенью двойки, а любые операции с числами-степенями двойки происходят без потери точности, так как все они представимы в double, следовательно, результатом суммирования (double)1/4 четыре раза будет ровно 1 без какой-либо погрешности. В цикле при проверке условия 1 < 1 вернется false, и результатом foo(4) станет 4.

Задача 7.

Переменной result не присвоено значение, объявлена она в теле функции, значит ее значение не определено. В цикле к ней прибавляется 1/3, что есть 0, так как это int с округлением до целой части. Можно сделать вывод, что результатом выполнения функции с любым параметром будет неопределенное значение, причем оно не изменится в теле цикла.

Задача 8.

Самый простой и очевидный пример такой функции вытекает из формулы суммы первых n элементов чисел натурального ряда.

```
int sum(int const &n) {
	return (n * n + n) / 2;
}
```

Это решение не очень изящно по причине того, что n оказывается ограничено сверху неравенством $n^2 < S$, где S – максимальное значение типа данных, чтобы избежать переполнения переменной. Можно также увеличить диапазон в $\sqrt{2}$ раз, если немного модифицировать функцию. Например, для int диапазон расширится с 46340 –> 65535.

```
int sum(int const &n) {
	int base = (n + 1) * (n / 2);
	return (n % 2 == 0) ? base : base + n / 2 + 1;
}
```

Однако качественно ситуацию может изменить только использование длинной арифметики.

Задача 9.

При наследовании первыми вызываются конструкторы базовых классов, затем наследующих, деструкторы в обратном порядке. Поля инициализируются при вызове конструктора, при этом раньше, чем тело конструктора. Вывод:

Foo Value Generator
Foo Constructor
Bar Value Generator
Bar Constructor

Задача 10.

Переменная tmp объявлена как Foo, но содержит в себе Bar, следовательно, при ее удалении будет вызван деструктор Foo, что ведет к утечке памяти m_data. Решение – модификатор virtual при деструкторе базового класса, таким образом гарантированно будет вызван деструктор наследника, и только затем деструктор базового класса.