Interview Zen

Home   Create an Interview

[redacted]

Total duration: 176:42

Question 1

```
Products of an Array

RESPONSE DETAILS:

• Code should be production quality – clearly written, runnable, and documented.
• Provide several examples that call your function and demonstrate that it works.
• You may use any functions or classes from the JDK, STL, or .Net Framework. Do not include extraneous code that is not
relevant to your solution.
• Provide the average runtime and space complexity (memory usage), and worst-case runtime and space complexity for your
solution, and a short explanation as to why.
• State any assumptions you make for your solution.

QUESTION:
Given an array of integers, produce an array whose values are the product of every other integer excluding the current index.

Example:
[4, 3, 2, 8] -> [3*2*8, 4*2*8, 4*3*8, 4*3*2] -> [48, 64, 96, 24]
```

```
#include <assert.h>
#include <vector>

/*
```

```
 * This function produce an array whose values
 * are the product of every other integer excluding the current index.
 */

typedef long long int64_t;
vector<int64_t> product(vector<int> & v) {
    int n = v.size();
    vector<int64_t> result(n, 0);
    if (n < 2) {
        return result;
    }

    vector<int64_t> f(n);
    vector<int64_t> g(n);

    // calculate array f from left to right
    f[0] = v[0];
    for (int i = 1; i < n; ++i) {
        f[i] = f[i-1] * v[i];
    }
    // calculate array g from right to left
    g[n-1] = v[n-1];
    for (int i = n - 2; i >= 0; --i) {
        g[i] = g[i+1] * v[i];
    }

    result[0] = g[1];
    result[n-1] = f[n-2];
    for (int i = 1; i < n - 1; ++i) {
        // calculate result by multiplying the left and the right
        result[i] = f[i-1] * g[i+1];
    }

    return result;
}

/*
 * Main function to call the above function and demonstrate that it works by assert
 */

int main(int argc, char** argv) {
    {
        int a[] = {};
        int b[] = {};
        vector<int> v(a, a + sizeof(a) / sizeof(int));
        vector<int> ans(b, b + sizeof(b) / sizeof(int));
        auto output = product(v);
        assert(ans.size() == output.size());
        int i = 0;
        for (auto k : ans) {
            assert(output[i++] == k);
        }
    }

    {
        int a[] = {4};
        int b[] = {0};
        vector<int> v(a, a + sizeof(a) / sizeof(int));
        vector<int> ans(b, b + sizeof(b) / sizeof(int));
        auto output = product(v);
        assert(ans.size() == output.size());
        int i = 0;
        for (auto k : ans) {
            assert(output[i++] == k);
        }
    }

    {
        int a[] = {4, 3};
        int b[] = {3, 4};
        vector<int> v(a, a + sizeof(a) / sizeof(int));
        vector<int> ans(b, b + sizeof(b) / sizeof(int));
        auto output = product(v);
        assert(ans.size() == output.size());
        int i = 0;
        for (auto k : ans) {
            assert(output[i++] == k);
        }
    }

    {
        int a[] = {4, 3, 2};
        int b[] = {6, 8, 12};
        vector<int> v(a, a + sizeof(a) / sizeof(int));
        vector<int> ans(b, b + sizeof(b) / sizeof(int));
        auto output = product(v);
        assert(ans.size() == output.size());
        int i = 0;
        for (auto k : ans) {
            assert(output[i++] == k);
        }
    }

    {
        int a[] = {4, 3, 2, 8};
        int b[] = {48, 64, 96, 24};
        vector<int> v(a, a + sizeof(a) / sizeof(int));
        vector<int> ans(b, b + sizeof(b) / sizeof(int));
        auto output = product(v);
        assert(ans.size() == output.size());
        int i = 0;
        for (auto k : ans) {
            assert(output[i++] == k);
        }
    }

    return 0;
}


Test case:
    [] -> []
    [4] -> [0]
    [4, 3] -> [3, 4]
    [4, 3, 2] -> [6, 8, 12]
    [4, 3, 2, 8] -> [48, 64, 96, 24]

Let N be the input array size

Avg runtime complexity:
    O(N), because there are 3 traversals with size N
Worst-case runtime complexity:
    Same as average

Avg space complexity:
    O(N), because there are 3 extra arrays with size N.
Worst-case space complexity:
    Same as average

Assumptions:
    No overflow when calculate the product
```

1x 2x 5x 0:00 / 86:09

Question 2

Pretty Number Formatting

```
RESPONSE DETAILS:

• Code should be production quality – clearly written, runnable, and documented.
• Provide several examples that call your function and demonstrate that it works.
• You may use any functions or classes from the JDK, STL, or .Net Framework. Do not include extraneous code that is not
relevant to your solution.
• Provide the average runtime and space complexity (memory usage), and worst-case runtime and space complexity for your
solution, and a short explanation as to why.
• State any assumptions you make for your solution.

QUESTION:

Write a function to convert an Integer representing a number of bytes (less than or equal to 1 Gigabyte) into an easy to read
format, defined as follows:
• Maximum of 3 digits (not counting a decimal point), and a single letter to signify the unit of measure.
• No leading zeroes, or trailing zeroes after a decimal point.
• Be as accurate as possible.
IMPORTANT DETAILS:
• Maximum of 3 digits (not counting a decimal point), and a single letter to signify the unit of measure.
• Round to the nearest valid values.

Examples:
o 341 = 341B
o 34200 = 34.2K
o 5910000 = 5.91M
o 1000000000 = 1G
o No leading zeroes, or trailing zeroes after a decimal point.

Examples:
o 34K, not 034K
o 7.2M, not 7.20M
o Be as accurate as possible. Example:
o 54123B = 54.1K, not 54K
o Note: For this problem, 1000 bytes = 1 KB, and so on.
```

```
#include <assert.h>
#include <vector>
#include <string>
#include <iostream>
using namespace std;

/**
*
* This function returns the number of digits of an integer
*
**/
int digits(int n) {
    int i = 0;
    while (n) {
        n /= 10;
        ++i;
    }
    return i;
}

/**
*
* This function returns the easy to read format of an integer
*
**/
string formatInt(int n) {
    string units[] = {"B", "K", "M", "G"};
    int level = 0;
    int quotient = n, remainder = 0;

    // calculate the quotient and remainder
    while (quotient >= 1000) {
        remainder = quotient % 1000;
        quotient = quotient/1000;
        ++level;
    }

    int numFrac = 3 - digits(quotient);
    int numZero = 3 - digits(remainder);
    numZero = min(numZero, numFrac);

    // round the fraction part
    int d[] = {1000, 100, 10};
    if (remainder / d[numFrac]) {
        remainder += d[numFrac] / 2;
        remainder /= d[numFrac];
    } else {
        remainder = 0;
    }

    // generate output
    char tmp[8];
    if (0 == remainder) {
        sprintf(tmp, "%d", quotient);
    } else {
        // trim the tail 0 in fraction part
        while (0 == remainder % 10) {
            remainder /= 10;
        }
        string s(numZero, '0');
        sprintf(tmp, "%d.%s%d", quotient, s.c_str(), remainder);
    }

    string result(tmp);
    result += units[level];
    return result;
}

/**
*
* Main function to call the above function and demonstrate that it works using assert
*
**/
int main(int argc, char** argv) {
    int a[] = {341, 34000, 34200, 5910000, 7200000, 54123, 1024, 1002, 1120, 1158, 1000000000};
    string b[] = {"341B", "34K", "34.2K", "5.91M", "7.2M", "54.1K", "1.02K", "1K", "1.12K", "1.16K", "1G"};
    std::vector<int> v(a, a+sizeof(a)/sizeof(int));
    int j = 0;
    for(auto i : v) {
        auto r = formatInt(i);
        cout<<i<<" -> "<<r<<endl;
        assert(r == b[j++]);
    }
    return 0;
}


Test case:
    341 -> 341B
    34000 -> 34K
    34200 -> 34.2K
    5910000 -> 5.91M
    7200000 -> 7.2M
    54123 -> 54.1K
    1024 -> 1.02K
    1002 -> 1K
    1120 -> 1.12K
    1158 -> 1.16K
    1000000000 -> 1G

Let N be the input integer

Avg runtime complexity:
    O(Log(N)), because the loop is related with the number of digits of N in decimal
Worst-case runtime complexity:
    Same as average

Avg space complexity:
    O(1), because we only need constant extra space
```

```
Worst-case space complexity:
    Same as average

Assumptions:
    No negative input
```

1x 2x 5x

0:00 / 90:23