```cpp
// shift the integer and static cast to unsigned char to get the 8 rightmost bits
#define RADIXBYTE(num, pass) static_cast<std::uint8_t>((num >> (pass << 3)))

void RadixSort(std::vector<std::uint32_t> &vector){
    // create counters
    bool sortedleft = true;
    bool sortedright = true;
    std::array<std::uint64_t, 256 * 4> counters = { 0 }; // store for all 4 passes in one array and use offset
    for(std::uint64_t i = 0; i < vector.size(); ++i){
        // check if array is already fully sorted in either direction
        if(i > 0 && ( vector.at(i - 1) > vector.at(i))) sortedleft = false;
        if(i > 0 && ( vector.at(i - 1) < vector.at(i))) sortedright = false;
        // passes 0, 1, 2, 3
        counters.at(256 * 0 + RADIXBYTE(vector.at(i), 0))++;
        counters.at(256 * 1 + RADIXBYTE(vector.at(i), 1))++;
        counters.at(256 * 2 + RADIXBYTE(vector.at(i), 2))++;
        counters.at(256 * 3 + RADIXBYTE(vector.at(i), 3))++;
    }
    // abort if sorted
    if(sortedleft || sortedright) return;
    // calculate prefixsum in 4 passes
    std::array<bool, 4> skips = { false };
    for(std::uint8_t offset = 0; offset < 4; ++offset){
        // check if all elements in this pass are zero
        if(counters.at(256 * offset) == vector.size()){
            skips.at(offset) = true;
            continue;;
        }
        for(std::uint16_t i = 1; i < 256; ++i){
            counters.at(256 * offset + i) += counters.at(256 * offset + i - 1);
        }
    }
    //rebuilt in 4 passes
    std::vector<std::uint32_t> output(vector.size());
    for(std::uint8_t pass = 0; pass < 4; ++pass){
        // check skip
        if(skips.at(pass)){
            continue;
        }
        // one iteration
        for(std::uint64_t i = vector.size(); i-- > 0;){
            std::uint8_t radix = RADIXBYTE(vector.at(i), pass);
            // decrement counter to make it point to right index
            counters.at(256 * pass + radix)--;
            output.at( counters.at(256 * pass + radix) ) = vector.at(i);
        }
        // swap references
        std::swap(vector, output);
    }
}
```