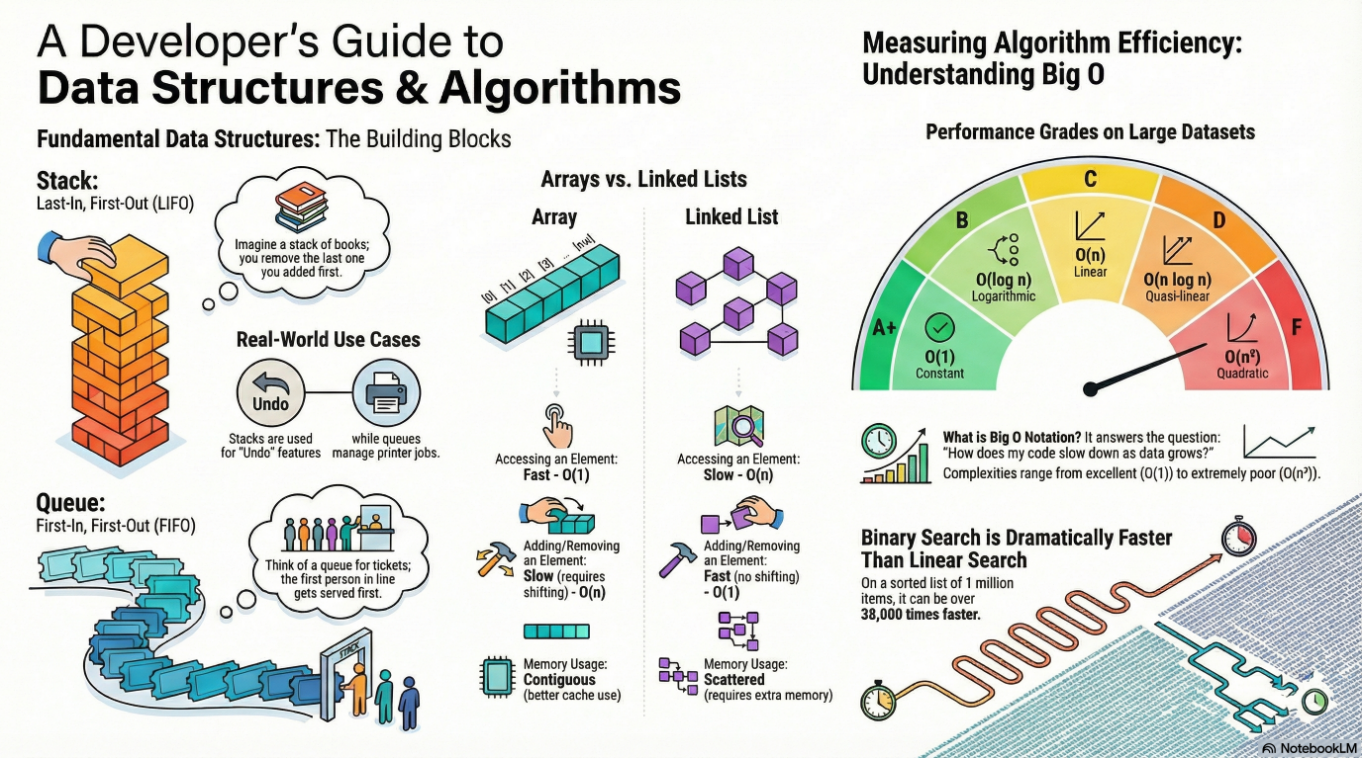
kintsugi-stack-dsa-cpp: LEETCODE

> “Talk is cheap. Show me the time complexity.”

- Author: Kintsugi-Programmer



> Disclaimer: The content presented here is a curated blend of my personal learning journey, experiences, open-source documentation, and invaluable knowledge gained from diverse sources. I do not claim sole ownership over all the material; this is a community-driven effort to learn, share, and grow together.

Table of Contents

Array & Hashing

TotalCompanyTags

> Baidu, Airbnb, Netease, Cisco, Amazon, Aetion, Box, Mathworks, Zoom, Google, Cloudera, Intel, Indeed, Godaddy, Walmart Global Tech, Salesforce, Didi, Affirm, Vmware, Yandex, Microsoft, Adobe, Alibaba, Jpmorgan, Linkedin, Citadel, Emc, Groupon, Intuit, Twitter, Nvidia, Twilio, Valve, Expedia, Yahoo, Zoho, Bookingcom, Wish, Zillow, Morgan-stanley, Drawbridge, Paypal, Huawei, Dropbox, Radius, Zomato, Roblox, Accenture, Goldman-sachs, Lyft, Yelp, Splunk, Bloomberg, Samsung, Bytedance, Servicenow, Quora, Goldman Sachs, Blackrock, Ebay, Ge-digital, Oracle, Qualcomm, Tencent, Uber, Tableau, Spotify, Morgan Stanley, American Express, Sap, Ibm, Deutsche-bank, Snapchat, Dell, Apple, Visa, Works-applications, Facebook, Factset, Audible, Google, Adobe, Facebook, Twilio, Salesforce, Affirm, Docusign, Yahoo, Cisco, Servicenow, Blackrock, Goldman Sachs, Ebay, Vmware, Tiktok, Bookingcom, Electronic-arts, Amazon, Wish, Microsoft, Yandex, Oracle, Qualtrics, Bloomberg, Adobe, Alation, Uber, Nutanix, Jpmorgan, Tesla, Mathworks, Zulily, Google, Hulu, Ibm, Snapchat, Apple, Intuit, Visa, Goldman-sachs, Yelp, Facebook, Walmart Global Tech, Bytedance, Yahoo, Cisco, Vmware, Ebay, Pocket-gems, Amazon, Microsoft, Oracle, Adobe, Uber, Spotify, Google, Linkedin, Hulu, Snapchat, Apple, Goldman-sachs, Yelp, Facebook, Bloomberg, Microsoft, Google, Linkedin, Square, Facebook, Twitter, Bloomberg, Amazon, Meta, Oracle, CrowdStrike, OpenAI, Snowflake, Salesforce, ByteDance, Yahoo, ServiceNow, Blackrock, eBay, VMware, Amazon, Microsoft, Oracle, Qualtrics, Adobe, Uber, Nutanix, Tableau, Google, Linkedin, SAP, PayPal, Groupon, Intel, Asana, Snapchat, Grab, Apple, Visa, Evernote, Goldman-sachs, Lyft, Facebook, Splunk, Bloomberg, Cisco, Goldman Sachs, Ibm, Meta, Tiktok, Walmart Global Tech, Zoho, Accenture, Autodesk, CEDCOSS, Disney, Docusign, Flipkart, Infosys, Intuit, Makemytrip, Paytm, Ripple, Sigmoid, Snap, TCS, Tekion, Turing, Unity, WarnerMedia, Wells Fargo, Yandex, ZS Associates, Spotify, Wish, Amazon, Apple, Microsoft, Google, Visa, PayPal, Oracle, Qualtrics, Adobe, Nutanix, Uber, Facebook, eBay, Bloomberg, Cisco, Goldman Sachs, Ibm, Meta, SAP, ServiceNow, Tiktok, Walmart Global Tech, Yahoo, Zoho, Atlassian, ByteDance, D. E. Shaw, Deltax, Epam Systems, Flipkart, Infosys, Lyft, Paytm, PhonePe, Roblox, Swiggy, TCS, Turing, UKG, Wissen Technology, Yandex, Zepto

---

PreReqs

- Read Question Multiple times pls

```
    // Read Question Multiple times pls
    // Read Question Multiple times pls
```

```
    // Read Question Multiple times pls
    // return {}; // No solution found, return an empty vector
    // else, this error comes: Line 10: Char 5: error: non-void function
```

```
does not return a value in all control paths [-Werror,-Wreturn-type] 10 |
}^1 error generated.
```

- STL Lib & namespace std

```
    // #include<bits/stdc++.h> // STL Lib
    // using namespace std;
```

- Vector Array

```
    // Vector Array
    // a way to store and access the list of elements.
    // dynamic array in C++ is std::vector
    // Why It's Useful: You don't need to know the exact number of elements
you'll be storing ahead of time
    // #include <vector> // lib
    std::vector<int> arr1; // empty vector
    std::vector<int> arr2={1,2,3}; // initialised vector
    std::vector<int> arr3(5); // size =5, each element value=default
value=0
    std::cout<<arr3[0]<<std::endl; //0 // Direct Access (0based)
    std::vector<int> arr4(5,90); // size =5, each element value=90
    std::cout<<arr4[0]<<std::endl; //90
    std::cout<<arr1.size()<<std::endl; //0 // Get Length
    arr4.push_back(5); // Append // btw each push in dynamic array is O(1)
    std::cout<<arr4[5]<<std::endl; //5
    for ( int i =0 ; i<arr2.size(); i++) { std::cout<<arr2[i]<<"    ";} //
Vector array traversal
    std::cout<<std::endl;
    // std::vector it's just a dynamic array that you can access with
integer indices (like tempo[0], tempo[1], etc.). It doesn't have a .find()
method that takes a string, nor can you use a string inside the square
brackets [].
    // The perfect tool for this is a std::unordered_map. It allows you to
store key-value pairs, which is exactly what you need: the sorted string as
the key and the list of anagrams as the value.
```

- Bucket/2D Vector

```
    // std::vector<std::vector<int>> (Bucket/2D Vector)
    // What It Is: This is simply a vector where each element is another
vector. You can visualize it as a 2D grid or a list of lists.
    // Why It's Useful: It's great for grouping items into "buckets."
    //      For instance, you could have a main vector where the index
represents a frequency.
```

```
    //      The element at my_buckets[5] would then be a vector containing
all the numbers that appeared 5 times.
    //      This technique, often called Bucket Sort, is a powerful way to
organize data by a specific property (like frequency).
    // Create a vector of 5 "buckets", where each bucket is an empty vector
of ints.
    int num_buck =5 ;
    vector < vector<int> > buckets(num_buck);
    // Let's put numbers into buckets based on some property.
    // For example, put numbers 10 and 11 into bucket 2.
    buckets[2].push_back(10);
    buckets[2].push_back(11);
    // Put number 25 into bucket 4.
    buckets[4].push_back(25);
    // Now, let's see what's in bucket 2.
    std::cout<< "Items in bucket 2: ";
    for (int items: buckets[2]) // buckets[2], not buckets
    {
        std::cout << items << " "; // Output: 10 11
    }
    std::cout << std::endl;
    // Items in bucket 2: 10 11
```

- For Loop

```
    // For Loop
    for ( int i=0; i<5; i++) std::cout<<i;
    // 01234
```

- Range-Based for Loop

```
    // Range-Based for Loop
    // What It Is: A cleaner, more modern syntax for a for loop that
iterates through all the elements of a container (like a vector or map)
without needing to manage indices or iterators manually.
    // Why It's Useful: It makes code much easier to read and write, and it
reduces the chance of off-by-one errors with indices. You just declare a
variable that will take the value of each element in the container, one by
one.
    vector<int> numbers = { 1,2,3 };
    cout<< "nos:";
    // => 'num' will take the value of each element in 'numbers'.
    for (int num: numbers){
        cout<<num<<" "; // 1 2 3
    }
    cout<<endl;
    // nos:1 2 3
    // => It works with maps, too. 'const auto&' is efficient here.
```

```
    unordered_map<char, int> meow = { {'a',3}, {'b',5 }}; // "c" NO CHAR,
'c' YES CHAR
    cout<< "nos:";
    for ( const auto& pair : meow) {cout<<"{"<<pair.first<<","
<<pair.second<<"}";}  // 'pair' is a key-value pair.
    cout<<endl;
    // nos:{b,5}{a,3}
```

- Hashset

```
    // Hashset
    std::unordered_set<int> hashSet;
    hashSet.insert(1);
    std::cout<<(
        hashSet.find(1)
        !=
        hashSet.end() // i.e if find give hashSet.end() ptr, then element
doesnt exist
    )<<std::endl; // true as 1 exists in hashset
    // 1
    // re-insert array elements in hashmap & checker while insertion:
number not existed in hashmap tip
    //  // given: vector<int>& nums
    //     unordered_set<int> hashbrown; // re-insert array elements in
hashmap
    //     for (int i=0; i<nums.size(); i++){
    //         if ( hashbrown.find(nums[i]) != hashbrown.end() ) // checker
while insertion: number not existed in hashmap
    //         {
    //         }
    //     }
    // ( hashbrown.find(nums[i] ) only returns ptr, not index, tip
```

- Sorting

```
    // Sorting
    // #include <algorithm> // Required for std::sort

    // Sorting a string
    std::string word = "cab";
    std::sort(word.begin(), word.end());
    std::cout << "Sorted word: " << word << std::endl;
    // Sorted word: abc

    // std::sort with Reverse Iterators
    std::vector<int> data = {40, 10, 50, 20, 30};

    // 1. Sort in default ascending order
    std::sort(data.begin(), data.end());
```

```
    std::cout << "Ascending: ";
    for (int x : data) std::cout << x << " "; // Output: 10 20 30 40 50
    std::cout << std::endl;
    // Ascending: 10 20 30 40 50

    // 2. Sort in descending order using reverse iterators
    sort(data.rbegin(), data.rend());
    std::cout << "Descending: ";
    for (int x : data) std::cout << x << " "; // Output: 50 40 30 20 10
    std::cout << std::endl;
    // Descending: 50 40 30 20 10

    // Sorting in Vector
    // (TC): O(nlogn) on average and in the worst case
    // (SC): O(logn)
    // std::sort sorts the vector in-place and does not return a sorted
vector. Its return type is void
    std::vector<int> v1 = {1,4,2,0};
    std::cout<<"v1: "<<v1[0]<<" "<< v1[1] << " " << v1[2] << " " << v1[3]
<< std::endl;
    // v1: 1 4 2 0
    std::sort(v1.begin(), v1.end()); // asc default
    std::cout<<"v1(sorted): "<<v1[0]<<" "<< v1[1] << " " << v1[2] << " " <<
v1[3] << std::endl;
    // v1(sorted): 0 1 2 4
    std::sort(v1.begin(), v1.end(), std::greater<int>()); // dsc
    std::cout<<"v1(sorted dsc): "<<v1[0]<<" "<< v1[1] << " " << v1[2] << "
" << v1[3] << std::endl;
    // v1(sorted dsc): 4 2 1 0
    // - Downside of This STL Sort
    //     - Differs from lang to lang
    //     - good sort algos do O(nlogn)
    //     - bad sort algos do O(n**2)
    //     - uncertain of SC
    //     - uncertain of TC
    //     - lenghty at cases where STL is not available
```

- unordered sets

```
    // unordered sets in c++ contains unique elements
    // if we even insert duplicate, it will reject it
    // we can make unord set with method to copy full the vector array,
rejecting the duplicates
    v1.push_back(1);
    v1.push_back(1);
    std::unordered_set<int> hashSet2(v1.begin(), v1.end()) ;// TC: O(n) avg
tc, O(n**2) worst rare tc where glitches happen &SC: O(k) Avg, O(n) worst,
k is number of unique elements
    std::cout<< hashSet2.size() << std::endl; // 4 // unique elements in
unord set
    std::cout<< v1.size() << std::endl; // 6 // duplicates elements in vect
arr
```

- string equality

```
    string s = "aba" ;
    string t = "aab" ;
    // string equality
    cout<<
    ( s==t ) // SC: O(min(N,M)), TC: O(1)
    << endl; // 0
    // cout<<s==t<< endl; // NO, always use explicit brackets to avoid
glitches
    sort(s.begin(),s.end());
    sort(t.begin(),t.end());
    cout<<( s==t )<< endl; // 1
```

- Storing Character Frequencies

```
    // Storing Character Frequencies
    // Storing Character Frequencies Way 1: Dynamic Array/ Vector
    vector<int> counts(26,0); // Creates a vector of size 26, with all
elements initialized to 0.
    // Since the problem states the strings only contain lowercase English
letters, you don't need a flexible map. A simple array of size 26 is
faster.
    // Here idx 0 -> 'a' -> 97
    // 25 -> 'z' -> 122
    char ch = 'a';
    counts[ch-97]++;
    counts[ch-97]++;
    cout<<counts[0]<<endl; //2

    // Storing Character Frequencies Way 2: Hashmap/ Unordered Map
    unordered_map<char,int> freq_map;
    freq_map['a']++;
    // If 'a' isn't in the map, it's added with value 1.
    // If it is, its value is incremented.
    // it has capability of direct comparison
        // freq_map == freq_map2 is valid
    string text = "hello world";
    unordered_map<char,int> char_freq;
    for ( char c: text){
        char_freq[c]++;
    }
    int charindex = 2;
    cout<< "Char "<<text[charindex]<<" comes "<<char_freq[text[charindex]]
<<" times\n";
    // Char l comes 3 times
```

- Character Arithmatics

```
    // Character Arithmatics
    // char types are internally represented as numbers (like ASCII
values). This allows you to perform math on them
    char c1 = 'a';
    char c2 = 'b';
    char c3 = 'c';
    int i1 = (c3 - c1); //2
    cout<<i1<<endl; //2
    // ASCII Values
    // 'a' = 97
    // 'z' = 122
    // 'A' = 65
    // 'Z' = 90
    // '0' = 48
    // '9' = 57

    // Calculate the 0-based index for any character
    char mychar = 'e';
    int index = mychar - 'a';
    std::cout << "The character '" << mychar << "' has an index of: " <<
index << std::endl;
    // The character 'e' has an index of: 4

    // General Looping tip
    // for (int i=0; i<s.size(); i++){
    //     counts[s[i]-97]++;
    // }
    // for (int i=0; i<s.size(); i++){
    //     counts[t[i]-97]--;
    // }
    // // you can just put them in one loop na
    // for (int i=0; i<s.size(); i++){
    //     counts[s[i]-97]++;
    //     counts[t[i]-97]--;
    // }
    // // You may forget this while focusing on knowledge
```

- Hashmap/ Unordered Map

```
    // Hashmap/ Unordered Map
    // This is the most powerful tool.
    // An unordered_map (or hash map) is like a super-fast dictionary.
    // You give it a key and it stores an associated value.
    // Its superpower is checking if a key exists or retrieving its value
in an average of constant time, $O(1)$, which is incredibly fast.
    // Why It's Useful: It provides incredibly fast lookups, insertions,
and deletions of elements. If you know the key, you can find its value
almost instantly, no matter how many items are in the map. This is perfect
```

```
for grouping items or counting frequencies.
    // For a problem where you're looking for target - nums[i], a hash map
is perfect for instantly checking: "Have I seen the number I need before?"
    unordered_map<string,int> std_scores;
    std_scores["Bali"]=101;
    std_scores["Bali"]=100;
    cout<<std_scores["Bali"]<<endl;// 100


    std::vector<int> nums1 = {3, 4, 5, 6, 4, 5, 4};
    std::unordered_map<int,int> momos;

    // Application : freq count for array containing duplicates, and index
find for array containing unique elements

    // Application : freq count for array containing duplicates
    for (int i: nums1)
    // special iterator
    // here i is not i , it's actually nums[i] of traditional for loop
    // faster
    // but can only use nums[i](value only), not i (index)

    {
        momos[i]++ ;// If num is not in the map, it's added. Then its count
is incremented.
    }
    // Fast Lookups in hashmap
    int target =4;
    if ( momos.find(target) != momos.end()){
        std:: cout << target << "'s momo :"<< momos[target] << std::endl ;
    }
    // 4's momo :3

    // Application :index find for array containing unique elements
    std::unordered_map<int,int> momos2;
    for ( int i=0 ; i<nums1.size(); i++){
        momos2[nums1[i]]=i; // not momos2[nums[i]]++ as it dont make sense
        // or momos2.insert( { nums[i] , i } );
    }
```

- Pair

```
    // Pair
    // you need to link two pieces of information together.
    // For example, what if you need to sort the numbers but not lose their
original positions ?
    // A std::pair is perfect for this. It holds exactly two items, which
you can access with .first and .second.
    // #include <utility> // lib
    std::pair<std::string, int> student1= {"Bali",101}; // {} way in cpp
    std::pair<std::string, int> student2{"Bhati",100}; // classic way
    auto student3 = std::make_pair("Bhaskar",100); // way of
```

```
std::make_pair() helper function ,This is useful because the compiler can
often figure out the types for you.
    student1.second=100;
    std::cout<<student1.first<<"\t"<<student1.second<<std::endl;// Bali
100

    // Vector of Pairs
    // value_index_pairs_vector
    std::vector<std::pair<int,int>> pv ; // vector of value_index_pairs
    std::vector<int> nums = {15, 8, 22, 5}; // random vect array, given in
problem
    for (int i=0; i <nums.size(); i++){ // Store each number with its
original index
        pv.push_back( { nums[i] , i } );
        // push_back() method
        // , with to push pairs: push_back({})
        // & our case :  push_back({value,index})
    }
    std::sort(pv.begin(),pv.end()); // Sort the pairs. By default, it sorts
by the first element (the value).
    for (int i=0; i<pv.size(); i++){
        std::cout<<pv[i].first<<"<-"<<pv[i].second<<std::endl;
    }
    // 5<-3
    // 8<-1
    // 15<-0
    // 22<-2
```

- String

```
    // String
    // #include <string>
    // String Concatenation
    // std::to_string: A simple function that converts a number (like an
int) into its string representation.
    // String Concatenation: The process of joining two or more strings
together to form a new, single string. In C++, this is easily done with the
+ or += operator
    // Why They're Useful: You often need to build a string from different
pieces of data, including numbers. std::to_string allows you to seamlessly
integrate numbers into your strings.
    std::string greet = "Hello World";
    std::string name= "Bali";

    // 1. String Concatenation using '+'
    std::string message = greet + ", I am " + name + " !!!";
    cout<<message<<endl;
    // Hello World, I am Bali !!!

    // 2. Using std::to_string
    int version = 5;
    string appname = " KintsugiDev.Studio ver."+ to_string(version); // We
```

```
must convert the number '5' to a string before we can join it
    cout<< appname << endl;
    //  KintsugiDev.Studio ver.5

    // 3. Building a key from parts using '+='
    std::string key = "";
    key+="user";
    key+=":";
    key+="Bali";
    key+=",";
    key+="id";
    key+=":";
    key+=to_string(2022496);
    cout<<key<<endl;
    // user:Bali,id:2022496
```

- priority_queue (Heap)

```
    // std::priority_queue (Heap)
    // priority_queue is a container that organizes elements based on
priority.
    // By default, it's a max-heap, meaning that whenever you look at the
top() element, it's always the largest one in the container.
    // When you pop() an element, the largest one is removed.
    // Why It's Useful: It's perfect for problems where you need to keep
track of the "top K" items without sorting the entire collection. For
example, you can maintain a priority queue of size k. As you process new
items, you can compare them to the smallest item in your "top K" set and
replace it if the new item is larger. This is much more efficient than
sorting everything.
    // #include <queue>
    // 1. Default priority_queue (Max-Heap)
    priority_queue<int> max_heap;
    max_heap.push(10);
    max_heap.push(20);
    max_heap.push(30);
    // The top element is always the largest.
    cout<< "Largest of max_heap :"<<max_heap.top()<<endl; //30
    // Largest of max_heap :30
    // 2. Min-Heap (keeps the smallest element at the top)
    // You must provide extra template arguments to change its behavior.
    priority_queue<int, vector<int>, greater<int>> min_heap;
    min_heap.push(10);
    min_heap.push(20);
    min_heap.push(30);
    // Now, the top element is always the smallest.
    cout<< "Smallest of min_heap :"<<min_heap.top()<<endl; //10
    // Smallest of min_heap :10
```

- typecasting(not parsing, parsing is dynamic translation)

```
    vector<string> strs = {}; // initialise too to avoid garbage
    string s = ""; // initialise too to avoid garbage


    // To convert an int to a std::string, use
    // std::string str = std::to_string(your_int);.

    // To convert a std::string to an int, use
    // int num = std::stoi(your_str);.
```

- Prefix & Suffix Multiplication

```
    // Prefix & Suffix Multiplication
    // The Core Idea:
    // To find the product of all elements EXCEPT index 'i', we can break
the problem into two parts:
    // 1. Product of everything to the LEFT of 'i' (Prefix Product)
    // 2. Product of everything to the RIGHT of 'i' (Suffix Product)
    // Formula: Result[i] = Prefix[i] * Suffix[i]

    // Visualizing the Arrays (Example: [1, 2, 3, 4])
    // -----------------------------------------------------------
    // Index:       0    1    2    3
    // Nums:      [ 1,   2,   3,   4 ]
    vector<int> nums2 = {1,2,3,4};
    vector<int> prefix_nums2(nums2.size());
    vector<int> suffix_nums2(nums2.size());


    // 1. Prefix Array (scans Left -> Right)
    //    Stores product of elements BEFORE current index.
    //    prefix[0] is always 1 (nothing to the left).
    //    prefix[1] = nums[0]
    //    prefix[2] = prefix[0] * nums[1]
    //    Prefix: [ 1,   1,   2,   6 ]
    prefix_nums2[0]=nums2[0];//obv
    for (int i=1; i<nums2.size(); i++){
        prefix_nums2[i]=prefix_nums2[i-1]*nums2[i-1];
    }
    // {1,1,2,6}

    // 2. Suffix Array (scans Right -> Left)
    //    Stores product of elements AFTER current index.
    //    suffix[3] is always 1 (nothing to the right).
    //    suffix[2] = suffix[3] * nums[3]
    //    suffix[1] = suffix[2] * nums[2]
    //    Suffix: [ 24,  12,  4,   1 ]
```

```
    suffix_nums2[nums2.size()-1]=1;// obv
    suffix_nums2[nums2.size()-2]=nums2[nums2.size()-1];//obv

    for (int i=nums2.size()-3; i>=0; i--){
        suffix_nums2[i]=suffix_nums2[i+1]*nums2[i+1];
    }// {24,12,4,1}

    // 3. Final Calculation
    //    Result[i] = Prefix[i] * Suffix[i]
    //    i=0: 1  * 24 = 24
    //    i=1: 1  * 12 = 12
    //    i=2: 2  * 4  = 8
    //    i=3: 6  * 1  = 6
    //    Result: [ 24,  12,  8,   6 ]
    vector<int> res_nums2(nums2.size());
    for (int i=0; i<nums2.size();i++) {
        res_nums2[i]=prefix_nums2[i]*suffix_nums2[i];
    }
    // {24,12,8,6}

    cout<<"arr :\t| ";
    for (int i=0; i<nums2.size();i++) {
        cout<<nums2[i]<<" | ";
    }
    cout<<endl;
    cout<<"pre :\t| ";
    for (int i=0; i<nums2.size();i++) {
        cout<<prefix_nums2[i]<<" | ";
    }
    cout<<endl;
    cout<<"suff :\t| ";
    for (int i=0; i<nums2.size();i++) {
        cout<<suffix_nums2[i]<<" | ";
    }
    cout<<endl;
    cout<<"res :\t| ";
    for (int i=0; i<nums2.size();i++) {
        cout<<res_nums2[i]<<" | ";
    }
    cout<<endl;
    // arr :   | 1 | 2 | 3 | 4 |
    // pre :   | 1 | 1 | 2 | 6 |
    // suff :  | 24 | 12 | 4 | 1 |
    // res :   | 24 | 12 | 8 | 6 |

    // Space Optimization Theory (O(N) -> O(1) space)
    // We don't actually need 3 separate arrays.
    // Step 1: Calculate Prefix products directly into the 'Result' array.
    //         Result is now: [1, 1, 2, 6]
    // Step 2: Iterate backwards. Instead of storing a full Suffix array,
    //         keep a running integer variable 'rightProduct'.
    //         Update Result[i] = Result[i] * rightProduct
    //         Then update rightProduct = rightProduct * nums[i]
    vector<int> res_nums2_1(nums2.size());
```

```
    res_nums2_1[0]=nums2[0];
    for ( int i=1; i<nums2.size(); i++){
        res_nums2_1[i]=res_nums2_1[i-1]*nums2[i-1];
    }// {1,1,2,6}
    // pre impose on res done

    int rightProd = 1;
    // res_nums2_1[nums2.size()-1]=res_nums2_1[nums2.size()-1]*rightProd;
    for ( int i=nums2.size()-1; i>=0 ;i--){
        res_nums2_1[i]=res_nums2_1[i]*rightProd;
        rightProd*=nums2[i];
    }// {24,12,8,6}
    // suff impose on res done


    cout<<"res2 :\t| ";
    for (int i=0; i<nums2.size();i++) {
        cout<<res_nums2_1[i]<<" | ";
    }
    cout<<endl;
    // res2 :  | 24 | 12 | 8 | 6 |
    // &btw res :   | 24 | 12 | 8 | 6 |
```

---

1 Concatenation of Array [Easy]

- https://leetcode.com/problems/concatenation-of-array/

> Google, Adobe, Facebook

Ques

You are given an integer array nums of length n. Create an array ans of length 2n where ans[i] == nums[i] and ans[i + n] == nums[i] for 0 <= i < n (0-indexed).

Specifically, **ans is the concatenation of two nums arrays**.

Return the array ans.

Example 1:

```
Input: nums = [1,4,1,2]

Output: [1,4,1,2,1,4,1,2]
```

Example 2:

```
Input: nums = [22,21,20,1]
```

```
Output: [22,21,20,1,22,21,20,1]
```

Constraints:

- 1 <= nums.length <= 1000.
- 1 <= nums[i] <= 1000

Solutions

- Basically We have to make final array of 2x input array
- **We can't do just Vector Concatnation like strings**
- Solution 1
    - assume nums as input vector
    - n = size of vector
    - **make new arr1 of size 2n default 0**
    - for loop 0 to n-1
        - arr1[i]= nums[i] **(insert 1 to n)**
    - for loop 0 to n-1
        - arr1[i+n]=arr[i] **(insert n+1 to 2n)**
    - return arr1

```cpp
// Solution 1
class Solution {
public:
    vector<int> getConcatenation(vector<int>& nums) {
        int n = nums.size();
        vector<int> ans(2*n);
        for (int i=0; i<n; i++){
            ans[i]= nums[i];
        }
        for (int i=0; i<n; i++){
            ans[n+i]= nums[i];
        }
        return ans;
    }
};
// Time complexity:
// O(n)
// Space complexity:
// O(n) for the output array.
```

- Solution 2
    - **Iteration (One Pass): Same Insertion in New Array, but in 1 loop, 2ops per iteration**
    - assume nums as input vector
    - n = size of vector
    - make new arr1 of size 2n default 0

    - for loop 0 to n-1
        - **arr1[i]= nums[i]**
        - **arr1[i+n]=arr[i]**
    - return arr1

```cpp
// Solution 2
class Solution {
public:
    vector<int> getConcatenation(vector<int>& nums) {
        int n = nums.size();
        vector<int> ans(2*n);
        for (int i=0; i<n; i++){
            ans[i]= nums[i];
            ans[i+n]= ans[i];
            // ans[i] = ans[i + n] = nums[i]; // we can write this too !!!

        }
        return ans;
    }
};
// Time complexity:
// O(n)
// Space complexity:
// O(n) for the output array.
```

- Solution 3
    - **just append in orginal array and return it**
    - **not making any new array ,saving space**
    - assume nums as input vector
    - n = size of vector
    - for loop 0 to n-1
        - nums.push_back(nums[i])
    - return nums
- **btw each push in dynamic array is O(1)**

```cpp
// Solution 3
class Solution {
public:
    vector<int> getConcatenation(vector<int>& nums) {
        int n = nums.size();

        for (int i=0; i<n; i++){
            nums.push_back(nums[i]);
        }

        return nums;
    }
};
```

```
// Time complexity:
// O(n)
// Space complexity:
// O(n) for the output array.
```

- Solution 4 **-- optimal**
    - Iteration (Two Pass)
    - **generic sol. with no. of times** var, here =2
    - assume nums as input vector
    - n = size of vector
    - for loop 1 to times-1
        - for loop 0 to n-1
            - nums.push_back(nums[i])
    - return nums
- btw each push in dynamic array is O(1)

```
// Solution 4
class Solution {
public:
    vector<int> getConcatenation(vector<int>& nums) {
        int n = nums.size();
        int times= 2;
        for (int j=1;j<times; j++)// j is 1 because ones occurence is
already here
        {
            for (int i=0; i<n; i++){
                nums.push_back(nums[i]);
            }
        }

        return nums;
    }
};

// Time complexity:
// O(n)
// Space complexity:
// O(n) for the output array.
```

---

2 Contains Duplicate [Easy]

- https://leetcode.com/problems/contains-duplicate/

> Airbnb, Amazon, Apple ,Microsoft, Tcs, Google, Yahoo, Oracle, Palantir-technologies, Adobe, Uber, Facebook, Bloomberg

---

Ques

Given an integer array nums, return true if any value appears more than once in the array, otherwise return false.

Example 1:

```
Input: nums = [1, 2, 3, 3]

Output: true
```

Example 2:

```
Input: nums = [1, 2, 3, 4]

Output: false
```

Constraints:

1 <= nums.length <= 105 -109 <= nums[i] <= 109

Recommended Time & Space Complexity You should aim for a solution with O(n) time and O(n) space, where n is the size of the input array.

Hint 1 A brute force solution would be to check every element against every other element in the array. This would be an O(n^2) solution. Can you think of a better way?

Hint 2 Is there a way to check if an element is a duplicate without comparing it to every other element? Maybe there's a data structure that is useful here.

Hint 3 We can use a hash data structure like a hash set or hash map to store elements we've already seen. This will allow us to check if an element is a duplicate in constant time.

Solutions

- So in this ques array
  - if any any element occurence >1 or **have duplicates**
    - **return true**
  - else
    - then the array have distinct elements
    - return false
- Solution 1 **-- brute force**
  - we are **checking each element to find it's same value by traversing each array eachtime**
  - given nums vect array
  - let n = nums vect array length
  - let counter = 0
  - if counter become more than 1 , i.e. item has multiple occurence

    - for i 0->n-1
        - for j 0->n-1
            - if nums[i]==nums[j] (not i==j)
                - counter++
                - **if counter>1**
                    - **return true**
        - counter = 0 // reset counter for next elements turn
    - return false // at case where counter didnt inc from 1 , i.e. not even onces occurence

```cpp
// Solution 1
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        int n = nums.size();
        int counter = 0;
        for ( int i=0; i<n; i++){
            for ( int j=0; j<n; j++){
                if (nums[i]==nums[j]){
                    counter++;
                    if (counter> 1){
                        return true;
                    }
                }
            }
            counter=0;
        }
        return false;
    }
};
// // Time & Space Complexity
// // Time complexity:
// // O(n**2)
// // Space complexity:
// // O(1)
```

- Solution 2 **-- optimal**
    - hashset
    - efficient tc O(n)
    - directly checking if element's occurence > 1
    - using another ds hash set
    - we are **inserting in hash set one by one** & **parallely checking if incoming element already exists in hash set**, if it already exists, so it mean ; at that point of time its duplicate is incoming
        - so this hashset contain duplicates
        - return true and exit, no need to continue, we got our ans
    - else try until any occurence is duplicate
    - if not true at any case and didn't exited earlier
        - return false
    - **now only 1 loop**, which is even just insertion in ds is only req.

    - thought:
        - i thought for an approach where we can remove that element in an array and still find if it's exist as duplicate or not.
        - NO
        - **this Solution similar acts , by not deleting it but checking during genesis of array**

```
// Solution 2
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        int n = nums.size();
        unordered_set<int> hashSet;
        for ( int i=0; i<n; i++){ // or // for (int num : nums) {
            if (hashSet.find(nums[i])!=hashSet.end()){ // or // if
(find.count(num)) {
                return true;
            }
            else {
                hashSet.insert(nums[i]);
            }
        }
        return false;
    }
};
// Time & Space Complexity
// Time complexity:
// O(n)
// Space complexity:
// O(n)
```

- Solution 3
    - **2 Pointer Approach & Sorting**
    - **no need of new space**
    - **sort the array**
    - O(nlogn)
    - then the duplicates would be adjacent to each other
    - even even once 2 adjacent elements are same, then we got our duplicate array proof
    - do **one loop** to just **check if arr[i]==arr[i+1]**
        - if true, return true
    - if loops ends without returning true;then array is distinct
        - return false

```
// Solution 3
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        int n = nums.size(); // or just use the method in loop too
        sort(nums.begin(), nums.end());
```

```
        for ( int i=0; i<n-1; i++){ // we took n-1 as to not going to case
(where i=n-1 index(last index), i+1=n index(exceeding limit)), and now at
i= n-2 (2nd last element index), then i+1 = n-1(last element index) :)
            if( nums[i]==nums[i+1]) return true;
        }
        return false;

    }
};
// Time & Space Complexity
// Time complexity:
// O(nlogn)
// Space complexity:
// O(1) or O(n) depending on the sorting algorithm.
```

- Solution 4 **-- optimal**
    - Hash Set Length
    - we can just **make set of distinct elements and compare size of old array**, if same, then false, else true
    - **unordered sets in c++ contains unique elements**
        - if we even insert duplicate, it will reject it
    - we can make unord set with method to copy full the vector array, rejecting the duplicates
    - if set size < vector size, it means that vec had duplicate elements
        - return true
    - else return false

```
// Solution 4
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        return (
            unordered_set<int>(nums.begin(), nums.end())
            .size()
            <
            nums.size()
        );

    }
};
// Time & Space Complexity
// Time complexity:
// O(n)
// Space complexity:
// O(n)
```

---

3 Valid Anagram [Easy]

- https://leetcode.com/problems/valid-anagram

- Expedia, Affirm, Docusign, Yahoo, Cisco, Servicenow, Goldman Sachs, Amazon, Microsoft, Oracle, Morgan-stanley, Uber, Spotify, Zulily, Google, Paypal, Snapchat, Apple, Goldman-sachs, Yelp, Facebook, Bloomberg

---

Ques

Given two strings s and t, return true if the two strings are anagrams of each other, otherwise return false.

An anagram is a string that contains the exact same characters as another string, but the order of the characters can be different.

> Anagram : Multiple String Arrays where each have
>
> 1. Same Elements
> 2. Same Length
> 3. Different Order

Example 1:

```
Input: s = "racecar", t = "carrace"

Output: true
```

Example 2:

```
Input: s = "jar", t = "jam"

Output: false
```

Constraints:

```
s and t consist of lowercase English letters.
```

Recommended Time & Space Complexity You should aim for a solution with O(n + m) time and O(1) space, where n is the length of the string s and m is the length of the string t.

Hint 1 A brute force solution would be to sort the given strings and check for their equality. This would be an O(nlogn + mlogm) solution. Though this solution is acceptable, can you think of a better way without sorting the given strings?

Hint 2 By the definition of the anagram, we can rearrange the characters. Does the order of characters matter in both the strings? Then what matters?

Hint 3 We can just consider maintaining the frequency of each character. We can do this by having two separate hash tables for the two strings. Then, we can check whether the frequency of each character in

string s is equal to that in string t and vice versa.

Solutions

- "anagram" has {a:3, n:1, g:1, r:1, m:1 } , "nagaram" has same {a:3, n:1, g:1, r:1, m:1 } thus anagram
- So basically if both string array have same elements despite any order , they are anagram
- thought:
    - so we have to check count of alphabets of both strings and compare
    - sounds like hashmap stuff
    - YES
- Solution 1
    - Sort Both String Arrays
    - check 1 if both lenght are equal or not
    - check 2 traverse and check each element of both array for equality
    - Downside of This STL Sort
        - Differs from lang to lang
        - good sort algos do O(nlogn)
        - bad sort algos do O(n**2)
        - uncertain of SC
        - uncertain of TC
        - lenghty at cases where STL is not available

```cpp
// Solution 1
class Solution {
public:
    bool isAnagram(string s, string t) {
        sort(s.begin(),s.end());
        sort(t.begin(),t.end());
        if (s.size() != t.size()) { return false; }
        for(int i=0; i<s.size(); i++){
            if (s[i]!=t[i]){
                return false;
            }
        }
        return true;


    }
};
// Time & Space Complexity
// Time complexity:
// O(nlogn+mlogm)
// Space complexity:
// O(1) or
// O(n+m) depending on the sorting algorithm.
// Where n is the length of string s and m is the length of string t.
```

- Solution 2

    - sort both s & t
    - return s==t ?

```
// Solution 2
class Solution {
public:
    bool isAnagram(string s, string t) {
        sort(s.begin(),s.end());
        sort(t.begin(),t.end());
        if (s.size() != t.size()) { return false; }
        return (s==t) ;
    }
};
// Time & Space Complexity
// Time complexity:
// O(nlogn+mlogm)
// Space complexity:
// O(1) or
// O(n+m) depending on the sorting algorithm.
// Where n is the length of string s and m is the length of string t.
```

- thought: Acc to ASCII, lets assume each char as a number bars
    - just add all ascii numbers as score 1 and score 2
    - if they are same then the strings are anagrams
    - NO, as 1+4 = 2+3
- Solution 3
    - The Hash Map Approach COUNTING
    - make char freq hashmap
    - traverse through str1 , inc freq. of freq hashmap
    - traverse through str1 , dec freq. of same freq hashmap
    - if both strs are anagrams, then second string traversal would cancel out all increasing of freq in freq hashmap

```
// Solution 3
class Solution{
    public:
        bool isAnagram(string s, string t){
            if (s.size() != t.size()) { return false; }
            unordered_map<char, int> freq_map;
            for (int i=0; i<s.size(); i++){
                freq_map[s[i]]++;
            }
            for (int i=0; i<s.size(); i++){
                freq_map[t[i]]--;
            }
            for (int i=0; i<s.size(); i++){
                if(freq_map[s[i]]>0) {
                    return false;
                }
```

```
            }
            return true;
        }
};
// Time & Space Complexity
// Time complexity:
// O(n+m)
// Space complexity:
// O(1) or O(k) since we have at most 26 different characters.
// The space is proportional to the number of unique characters, let's call
it k
// Where n is the length of string s and m is the length of string t.
```

- Solution 4 -- optimal
  - The Array-as-Counter Approach
    - SAME "The Hash Map Approach COUNTING" as Array
  - make char freq array with index
  - it dont stores character like hashmaps, indexes play save some space
  - traverse through str1 , inc freq. of arr
  - traverse through str1 , dec freq. of same arr
  - if both strs are anagrams, then second string traversal would cancel out all increasing of freq in arr

```
// Solution 4
class Solution{
    public:
        bool isAnagram(string s, string t){
            if (s.size() != t.size()) { return false; }
            vector<int> counts(26,0);
            for (int i=0; i<s.size(); i++){
                counts[s[i]-97]++;
            }
            for (int i=0; i<s.size(); i++){
                counts[t[i]-97]--;
            }
            // or merge them
            // for (int i = 0; i < s.length(); i++) {
            //     count[s[i] - 'a']++;
            //     count[t[i] - 'a']--;
            // }
            for (int i=0; i<s.size(); i++){
                if(counts[s[i]-97]>0) {
                    return false;
                }
            }
            // or traverse like this
            // for (int val : count) {
            //     if (val != 0) {
            //         return false;
```

```
            //     }
            // }
            return true;
        }
};
// Time & Space Complexity
// Time complexity:
// O(n+m)
// Space complexity:
// O(1) since we have at most 26 different characters.
// Where n is the length of string s and m is the length of string t.
```

- Solution 5
    - The Hash Map Approach COUNTING WAY 2
        - make char freq hashmap
        - traverse through str1 , inc freq. of freq hashmap
        - traverse through str1 , inc freq. of another freq hashmap2
        - return freq_map == freq_map2

```
// Solution 5
class Solution{
    public:
        bool isAnagram(string s, string t){
            if (s.size() != t.size()) { return false; }
            unordered_map<char, int> freq_map;
            unordered_map<char, int> freq_map2;
            for (int i=0; i<s.size(); i++){
                freq_map[s[i]]++;
            }
            for (int i=0; i<s.size(); i++){
                freq_map2[t[i]]++;
            }

            return freq_map == freq_map2 ;
        }
};
// Time & Space Complexity
// Time complexity:
// O(n+m)
// Space complexity:
// O(1) since we have at most 26 different characters.
// Where n is the length of string s and m is the length of string t.
```

---

4 Two Sum [Easy]

- https://leetcode.com/problems/two-sum

> Baidu, Airbnb, Netease, Cisco, Amazon, Aetion, Box, Mathworks, Zoom, Google, Cloudera, Intel, Indeed, Godaddy, Walmart Global Tech, Salesforce, Didi, Affirm, Vmware, Yandex, Microsoft, Adobe, Alibaba, Jpmorgan, Linkedin, Citadel, Emc, Groupon, Intuit, Twitter, Nvidia, Twilio, Valve, Expedia, Yahoo, Zoho, Bookingcom, Wish, Zillow, Morgan-stanley, Drawbridge, Paypal, Huawei, Dropbox, Radius, Zomato, Roblox, Accenture, Goldman-sachs, Lyft, Yelp, Splunk, Bloomberg, Samsung, Bytedance, Servicenow, Quora, Goldman Sachs, Blackrock, Ebay, Ge-digital, Oracle, Qualcomm, Tencent, Uber, Tableau, Spotify, Morgan Stanley, American Express, Sap, Ibm, Deutsche-bank, Snapchat, Dell, Apple, Visa, Works-applications, Facebook, Factset, Audible, Expedia, Affirm, Docusign, Yahoo, Cisco, Servicenow, Goldman Sachs, Amazon, Microsoft, Oracle, Morgan-stanley, Uber, Spotify, Zulily, Google, Paypal, Snapchat, Apple, Goldman-sachs, Yelp, Facebook, Bloomberg

---

Ques

Given an array of integers nums and an integer target, return the indices i and j such that nums[i] + nums[j] == target and i != j.

You may assume that every input has exactly one pair of indices i and j that satisfy the condition.

Return the answer with the smaller index first.

Example 1:

```
Input:
nums = [3,4,5,6], target = 7

Output: [0,1]
Explanation: nums[0] + nums[1] == 7, so we return [0, 1].
```

Example 2:

```
Input: nums = [4,5,6], target = 10

Output: [0,2]
```

Example 3:

```
Input: nums = [5,5], target = 10

Output: [0,1]
```

Constraints:

```
2 <= nums.length <= 1000
-10,000,000 <= nums[i] <= 10,000,000
```

```
-10,000,000 <= target <= 10,000,000
```

Recommended Time & Space Complexity You should aim for a solution with O(n) time and O(n) space, where n is the size of the input array.

Hint 1 A brute force solution would be to check every pair of numbers in the array. This would be an O(n^2) solution. Can you think of a better way? Maybe in terms of mathematical equation?

Hint 2 Given, We need to find indices i and j such that i != j and nums[i] + nums[j] == target. Can you rearrange the equation and try to fix any index to iterate on?

Hint 3 we can iterate through nums with index i. Let difference = target - nums[i] and check if difference exists in the hash map as we iterate through the array, else store the current element in the hashmap with its index and continue. We use a hashmap for O(1) lookups.

Solutions

- so basically,
  - given
    - Array
    - target
  - to find
    - index i &index j of array
    - && arr[i]+arr[j]=target
    - && i != j
    - Return the answer with the smaller index first.
    - You may assume that every input has exactly one pair of indices i and j that satisfy the condition.
- Solution 1 -- brute force
  - loop 1 : i 0 to n-1
    - loop 2 : j i+1 to n-1
      - if arr[i]+arr[j]=target
        - return {i,j}
  - if not found ,return {}

```
// Solution 1
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        for ( int i=0; i<nums.size(); i++){
            for ( int j=i+1; j<nums.size(); j++){
                if (nums[i]+nums[j] == target) { return {i,j} ; }
            }
        }
        return {};
    }
};
// Time & Space Complexity
```

```
// Time complexity:
// O(n**2)
// Space complexity:
// O(1)
```

- thought: let's just make checks of `(nums[i]+nums[nums.size()-i-1] == target`
  - NO, as ans can be `[0,1]`
- thought: arr[i]+arr[j]=target
  - arr[j]=target-arr[i]
  - we know target, if we just do hardwork to know arr[i] then, arr[j] will automatically come
  - YES, but then we can see arr[j] but need to extract index j
- thought: change the data stucture
  - hashset: NO, `( hashbrown.find(nums[i] )` only returns ptr, not index, tip
  - hashmap: YES, with < value, index >
  - pairs: YES, with < value, index >
  - YES
- Solution 2
  - Hash Map (Two Pass)
  - here we are finding i then j
    - { i,hash_map[target-nums[i]] }
  - make hashmap
    - where hashmap[value]=index
  - now loop traverse, first add all array elements with index nos. in hashmap
  - then another loop, to check
    - if element's Right Side Complement exists
      - i.e. target - arr[i] exists
      - , i.e. arr[j] && target - arr[i] != arr[i] (i.e. not point back to same element)
      - we are checking if the element 1's complement to sum the target exists to the RIGHT SIDE
      - i.e. element + Right Side Complement = target
    - if YES, then return the i ,then j as {i,j}
  - if nothing found, it didn't exist, return empty {}

```
// Solution 2
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int,int> momos;
        for ( int i = 0; i<nums.size(); i++ ) { momos[nums[i]]=i; }
        for ( int i = 0; i<nums.size(); i++ )
        {
            if ( momos.find(target-nums[i]) != momos.end() // or
momos.count(target-nums[i])
            && momos[target-nums[i]]!= i // to avoid the case of j = i
            )
            {
                return  { i,momos[target-nums[i]]};
```

```
            }
            // or
            // int diff = target - nums[i];
            // if (momos.count(diff) && momos[diff] != i) {
            //     return {i, momos[diff]};
            // }
        }
        return {};
    }
};
// Time & Space Complexity
// Time complexity:
// O(n)
// Space complexity:
// O(n)
```

- Solution 3 -- optimal
  - Hash Map (One Pass)
  - Solution 2's Clever/Smart Way
  - One Pass
  - Free from Edge cases
  - here we are finding j then i
    - { hash_map[target-nums[j]] , j }
  - now in one loop,
    - first check
      - if element's Left Side Complement Exists
        - i.e. Left Side Complement + Element = target
        - now we are finding LEFT SIDE, not right side; because this time we are checking before insertion
          - i.e. there is no right side elements of selected element
      - if exists then return i,then j as {i,j}
      - else array element with index no. in hashmap
  - if nothing found, it didn't exist, return empty {}
  - here we already avoide the case of j = i

```
// Solution 3
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int,int> momos;
        for ( int j = 0; j<nums.size(); j++ ){
            if ( momos.find(target-nums[j]) != momos.end() )
            {
                return  {momos[target-nums[j]] ,j}; // not {
j,momos[target-nums[j]]};
            }
            momos[nums[j]]=j; // not momos[nums[j]]++ as it dont make sense
            // or momos.insert( nums[j], j );
            // insertion should be second
```

```
            // else Let's say nums = [3, 2, 4] and target = 6
                // On the first loop (i=0, nums[0]=3), it calculates the
complement: 6 - 3 = 3.
                // Because you just added nums[0] to the map, the code
finds 3 in the map and incorrectly matches the element with itself,
returning {momos[3], 0} which is {0, 0}. This violates the i != j rule.
        }
        return {};
    }
};
// Time & Space Complexity
// Time complexity:
// O(n)
// Space complexity:
// O(n)
```

- thought: let's just sort the array and play first and last in one loop
  - NO, as we lost the indexes of array
  - thought: if we use value_index_pairs_vector instead of vector array
    - our for loop checks pairs at symmetrical indices:
      - When i = 0, it checks the 1st smallest (pv[0]) and the 1st largest (pv[size-1]).
      - When i = 1, it checks the 2nd smallest (pv[1]) and the 2nd largest (pv[size-2]).
      - ...and so on.
    -

```
// // WRONG
// class Solution {
// public:
//     vector<int> twoSum(vector<int>& nums, int target) {
//         vector< pair<int,int> > pv;
//         for (int i=0; i< nums.size(); i++){
//             pv.push_back( {nums[i],i} );
//         }
//         sort(pv.begin(), pv.end());
//         for (int i=0 ; i<pv.size(); i++)
//             if (
//                 pv[i].first
//                 +
//                 pv[pv.size()-i-1].first
//                 ==
//                 target
//             )
//             {
//                 return (
//                     {
//                         pv[i].second
//                         ,
//                         pv[pv.size()-1-i].second
//                     }
//                 );
//             }
//         return {};
```

```
//     }
// };
```

    - NO
        - The problem is that the correct pair isn't always symmetrical. The solution might be the 1st smallest and the 3rd largest
- Solution 4
    - Sorting
    - This is the two-pointer technique. I just need to put this logic into a while (left < right) loop. It's guaranteed to find the solution.
    - thought: if we use value_index_pairs_vector instead of vector array. Let's go back to the sorted list and think again. I have one pointer at the start (left) and one at the end (right).
        - The Core Question: I have sum = left_value + right_value. How does this sum guide my next move?
        - Case 1: sum > target: The total is too big. To make it smaller, the only logical move is to decrease the larger number. So, I must move the right pointer inward (right--).
        - Case 2: sum < target: The total is too small. To make it bigger, the only logical move is to increase the smaller number. So, I must move the left pointer inward (left++).
        - YES
        - textbook-quality solution !!!

```
// Solution 4
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        vector< pair<int,int> > pv;
        for (int i=0; i< nums.size(); i++){
            pv.push_back( {nums[i],i} );
        }
        sort(pv.begin(), pv.end());
        int i =0;
        int j = nums.size() -1;
        while (i<j){
            if ( pv[i].first + pv[j].first == target )
            {
                return  { min(pv[i].second , pv[j].second) ,
max(pv[i].second , pv[j].second) } ;
                // NOT return ( { min(pv[i].second , pv[j].second) ,
max(pv[i].second , pv[j].second) } );
                // The syntax ( { ... } ) is not the standard way to return
a newly created vector. The compiler misinterprets it and complains that it
can't convert the result to a std::vector<int>
            }
            else if ( pv[i].first + pv[j].first < target ) { i++; }
            else if ( pv[i].first + pv[j].first > target ) { j--; }

        }

        return {};
```

```
    }
};
// Time & Space Complexity
// Time complexity:
// O(nlogn)
// Space complexity:
// O(n)
```

---

5 Group Anagrams [Medium]

- https://leetcode.com/problems/group-anagrams/

> Twilio, Salesforce, Affirm, Docusign, Yahoo, Cisco, Servicenow, Blackrock, Goldman Sachs, Ebay, Vmware, Tiktok, Bookingcom, Electronic-arts, Amazon, Wish, Microsoft, Yandex, Oracle, Qualtrics, Bloomberg, Adobe, Alation, Uber, Nutanix, Jpmorgan, Tesla, Mathworks, Zulily, Google, Hulu, Ibm, Snapchat, Apple, Intuit, Visa, Goldman-sachs, Yelp, Facebook, Walmart Global Tech

---

Ques

Given an array of strings strs, group all anagrams together into sublists. You may return the output in any order.

An anagram is a string that contains the exact same characters as another string, but the order of the characters can be different.

Example 1:

```
Input: strs = ["eat","tea","tan","ate","nat","bat"]

Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
```

Explanation:

```
There is no string in strs that can be rearranged to form "bat".
The strings "nat" and "tan" are anagrams as they can be rearranged to form
each other.
The strings "ate", "eat", and "tea" are anagrams as they can be rearranged
to form each other.
```

Example 2:

```
Input: strs = ["x"]

Output: [["x"]]
```

Example 3:

```
Input: strs = ["a"]

Output: [["a"]]
```

Constraints:

```
1 <= strs.length <= 1000
0 <= strs[i].length <= 100
strs[i] consists of lowercase English letters.
```

Recommended Time & Space Complexity You should aim for a solution with O(m * n) time and O(m) space, where m is the number of strings and n is the length of the longest string.

Hint 1 A naive solution would be to sort each string and group them using a hash map. This would be an O(m * nlogn) solution. Though this solution is acceptable, can you think of a better way without sorting the strings?

Hint 2 By the definition of an anagram, we only care about the frequency of each character in a string. How is this helpful in solving the problem?

Hint 3 We can simply use an array of size O(26), since the character set is a through z (26 continuous characters), to count the frequency of each character in a string. Then, we can use this array as the key in the hash map to group the strings.

Solutions

- basically
  - given array of strs
  - return array of sublists of anagram string groups
- thought: You're trying to use a std::vector like a dictionary or a hash map, where you can look up a value using a string key.
  - NO
  - However, a std::vector in C++ doesn't work that way; it's just a dynamic array that you can access with integer indices (like tempo[0], tempo[1], etc.). It doesn't have a .find() method that takes a string, nor can you use a string inside the square brackets [].
- thought: The perfect tool for this is a std::unordered_map. It allows you to store key-value pairs, which is exactly what you need: the sorted string as the key and the list of anagrams as the value.
  - YES
- Solution 1 -- brute force
  - Sorting
  - thought:
    - i grouped them based on id as identity of sorted string

        - so i need to do one sort operation each traversal to find id which is sorted str
        - YES
    - make unord hashmap tempo
        - contain stuff as pairs {"abc", {"acb", "bac"}}
            - "abc"
            - {"acb", "bac"}
    - traverse through array of strs
        - each element
        - sort it using sort stl
        - if sorted string exists in tempo
            - append the element in sorted pair
        - else
            - make pair of
                - sorted element key
                - empty array
    - make new vector of vector of string fin
    - traverse through tempo
        - append second of pair into fin
    - return fin
    - TC
        - for each str : n log n from sort
        - for arra of m elements : m* n log n

```
// Solution 1
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map< string, vector<string> > tempo;

        for ( int i=0 ; i<strs.size() ; i++){
            // one string iteration "string1" strs[i]
            string s = strs[i];
            sort(s.begin(),s.end()); // s is sorted strs[i]

            // Add the original string to the vector associated with the
sorted key.
            // If the key doesn't exist, C++ creates it automatically!
            tempo[s].push_back(strs[i]);
        //     if ( tempo.find(s) != tempo.end ) // if s exist in tempo
        //     {
        //         // append strs[i] in tempo[s]
tempo[s].push_back(strs[i])
        //         tempo[s].push_back(strs[i]);
        //     }
        //     else
        //     {
        //         // make new str,{} in tempo
        //         tempo[s].push_back(strs[i]);
        //     }
```

```
        }
        vector<vector<string>> fin;
        for ( auto const& pairs: tempo){ // eg: pair { "abc", {array of
anagrams} }
            fin.push_back(pairs.second);
        }
        return fin;
    }
};
// Time & Space Complexity
// Time complexity:
// O(m*nlogn)
// Space complexity:
// O(m*n)
// Where
// m is the number of strings and
// n is the length of the longest string.
```

- Solution 2 -- optimal
  - Hash Map/Table
  - same working of Solution 1
  - but another ID
  - thought:
    - i grouped them based on id as identity
    - so i did earlier to do one sort operation each traversal to find id which is sorted str
    - is there any other way to make ids and group play??
    - idea: make an id
    - made of 0s and 1s of freq array of 26 alphabet & one seperator
      - eg: abc => 1.1.1.0.0.0.0.....0
        - bca => 1.1.1.0.0.....0
          - same as abc, cab, bac etc.
    - YES, really like adj matrix
  - TC:
    - just using hashmap
    - just counting stuff
    - O(m * n * 26)
      - n is elements size
      - m is array size
      - 26 is Alphabets iteration

```
// Solution 2
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map< string, vector<string> > tempo;
        for ( const string& s : strs)
        {
            // Create a character count array for each string
```

```
            // An array of 26 integers, one for each letter 'a' through
'z'.
            vector<int> freq(26,0);
            for (char c: s)
            {
                freq[c-'a']++;
            }
            // Build a unique key string from the count array.
            // For "eat": count['a'-'a']=1, count['e'-'a']=1, count['t'-
'a']=1
            // Key might look like: "1.0.0.0.1.....1...."
            string _ID = "";
            for ( int i =0; i<26; i++)
            {
                _ID+= to_string(freq[i]);
                _ID+="."; // as a seperator
            }
            tempo[_ID].push_back(s);
        }
        vector< vector<string>> fin={};
        for (auto const& [key,val]: tempo)
        {
            fin.push_back(val);
        }
        return fin;
    }
};
// Time & Space Complexity
// Time complexity:
// O(m*n)
// Space complexity:
// O(m) extra space.
// O(m*n) space for the output list.
// Where
// m is the number of strings and
// n is the length of the longest string.
```

---

6 Top K Frequent Elements [Medium]

- https://leetcode.com/problems/top-k-frequent-elements/description/

> Bytedance, Yahoo, Cisco, Vmware, Ebay, Pocket-gems, Amazon, Microsoft, Oracle, Adobe, Uber, Spotify, Google, Linkedin, Hulu, Snapchat, Apple, Goldman-sachs, Yelp, Facebook, Bloomberg

---

Ques

Given an integer array nums and an integer k, return the k most frequent elements within the array.

The test cases are generated such that the answer is always unique.

You may return the output in any order.

Example 1:

```
Input: nums = [1,2,2,3,3,3], k = 2

Output: [2,3]
```

Example 2:

```
Input: nums = [7,7], k = 1

Output: [7]
```

Constraints:

```
1 <= nums.length <= 10^4.
-1000 <= nums[i] <= 1000
1 <= k <= number of distinct elements in nums.
```

Recommended Time & Space Complexity You should aim for a solution with O(n) time and O(n) space, where n is the size of the input array.

Hint 1 A naive solution would be to count the frequency of each number and then sort the array based on each element's frequency. After that, we would select the top k frequent elements. This would be an O(nlogn) solution. Though this solution is acceptable, can you think of a better way?

Hint 2 Can you think of an algorithm which involves grouping numbers based on their frequency?

Hint 3 Use the bucket sort algorithm to create n buckets, grouping numbers based on their frequencies from 1 to n. Then, pick the top k numbers from the buckets, starting from n down to 1.

Solutions

- this question seems to be easy
- but this is quite hard to solve once
- basically
    - Given an integer array nums and an integer k
    - return the k most frequent elements within the array.
    - The test cases are generated such that the answer is always unique.
- thought :
    - was blackout, full algo didn't came at first try
    - origin thinking was to make a map containing freq, element

    - sort them desc
    - make fin arr
    - and while k--
        - append fin freq's element
    - return
    - NO
        - was beginner, other way too as iteration
    - Solution 1 came out of incapability of using just unordered_map or vector or vector pair
        - i tried here to use their synergy
        - YES
    - Will try this thought inspiration in Sol 2
        - YES

- Solution 1 -- optimal

    - Bucket Sort
    - Need to Revisit after Getting Grip
    - thought: the "bucket" array, where the index represents the frequency (count) Advantage

```
eg:
[1,1,2,2,2,3,10]
- this is unbounded
  - i.e. maxelement of this array exceed size of array
  - i.e. any element 100000,99999,etc. can be in this array
  - COUNT IS BTW TIMES OF OCCURENCE
- way 1 to store
  - i :       [0,1,2,3,4,5,6,7,8,9,10] IDX
  - count :   [0,2,3,1,0,0,0,0,0,0,1] VALUES
- way 2 to store -- optimal
  - count : [0,1,2,3,4,5,6] IDX
  - i : [[null],[10,3],[1],[2],X,X,X] VALUES
  - very much TC & SC Efficient
  - + new way to process
    - also even if array is big, still stop way 2 bucket till k ,
ignore after that
      - starting from reverse obv.
      - i.e. if k=2, so
        - 6,5,4 is out ,no data
        - process 3,2
        - ignore rest !!!
    - linear time
```

    - The two-step process is the standard, efficient pattern.
    - Use a map for what it's best at: fast lookups and counting.
    - Use a vector for what it's best at: storing data in a sequence that can be sorted.
    - make map named tempo
        - {{number,freq},{number,freq},{number,freq}}
        - Count the frequency of each number.
    - make bucket vector tempo2

        - {{number,number,number},{number,number,number},{number,number,number}}
    - here 0 index vector have 0 freq nos, 1 index nos have 1 freq nos, etc
    - 0 would be {} obv
        - useage
    - eg: buckets[5] = a list of all numbers that appeared exactly 5 times.
    - eg: buckets[4] = a list of all numbers that appeared exactly 4 times.
    - now traverse tempo
        - store tempo2[freq].push_back(number)
    - done
    - now to just append to fin array and return fin
    - actual_size_bucket = tempo2.size() -1
    - travers tempo2 reverse order
        - You can't sort the buckets vector itself, because the frequency information is stored in its indices, not in the values it holds. Sorting it would scramble this crucial relationship.
        - so now
        - traverse i lastFreq -> 0 && fin<=k
    - for nums in tempo[i]
        - fin.push_back(nums)
    - check if fin==k
        - The test cases are generated such that the answer is always unique
        - break
    - else
        - continue
    - return fin

```
// Solution 1
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        unordered_map<int,int> tempo; // COUNT
        // 1 Count the frequency of each number.
        for ( const int i : nums )
        {
            tempo[i]++;
        }

        // The two-step process is the standard, efficient pattern.
        // Use a map for what it's best at: fast lookups and counting.
        // Use a vector for what it's best at: storing data in a sequence
that can be sorted.

        // 2 Create buckets. The index is the frequency.
        // The size is nums.size() + 1 because a number can appear at most
nums.size() times.
        vector<vector<int>> tempo2(nums.size() + 1); // FREQ
        // Connecting Frequency to Array Indexing To store the number 5 in
our buckets vector, we need to place it at the index corresponding to its
frequency. So, we need to access buckets[4].
        // Because C++ vectors are 0-indexed, to have a valid index at 4,
```

```
the vector must have a size of at least 5 (to contain indices 0, 1, 2, 3,
and 4).
        // Therefore, if the maximum possible frequency is nums.size(), the
required size for the buckets vector is nums.size() + 1.
        for ( const auto& [key,value] : tempo) // key,value are num, freq
res.
        {
            tempo2[value].push_back(key);
        }

        // NO sort(tempo2.rbegin(),tempo2.rend());
        // You can't sort the buckets vector itself, because the frequency
information is stored in its indices, not in the values it holds. Sorting
it would scramble this crucial relationship.
        // You can't sort the buckets vector itself, because the frequency
information is stored in its indices, not in the values it holds. Sorting
it would scramble this crucial relationship.
        // The Role of the buckets Vector
        //     Think of the buckets vector like a series of filing cabinet
drawers. The number on each drawer is the frequency (the index), and inside
the drawer are the numbers that appeared that many times.
        //     eg: buckets[5] = a list of all numbers that appeared exactly
5 times.
        //     eg: buckets[4] = a list of all numbers that appeared exactly
4 times.
        vector<int> fin;
        // 3 Iterate backwards from the highest possible frequency.
        // Add elements to the result until we have k elements.
        int actual_size_bucket = tempo2.size() -1;
        for ( int i = actual_size_bucket ; i >=0 && fin.size()<k ; --i )
        {
            for ( int num: tempo2[i]){
                fin.push_back(num);
                if ( fin.size() == k){
                    break;
                }
            }
        }
        return fin;


    }
};
// Time & Space Complexity
// Time complexity:
// O(n)
// Space complexity:
// O(n)
```

- Solution 2 -- brute force
- Sorting

- Count frequencies
  - make unord map <int,int> freqMap
  - for nums in nums
    - freqMap num ++
      - it will inc freq of existing fre in freq map
      - or if new so make new freq
- Convert to a vector of pairs { freq, number}
  - make vect of pairs freqVec
  - traverse map_pair in freqMap
    - freqVec.push_back({ map_pair.second, map_pair.first })
- Sort the vector by frequency in descending order
  - use sort(freqVec.rbegin(), freqVec.rend())
- Extract the top k elements
  - make fin which is vect int array
  - i 0->k-1
    - fin.push_back freqVec [i] Second
- return fin

```
// Solution 2
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        unordered_map<int,int> freqMap;
        for ( const int i : nums )
        {
            freqMap[i]++;
        }
        vector<pair<int,int>> freqVect;
        for ( const auto& [k,v]: freqMap){
            freqVect.push_back({v,k});
        }
        sort(freqVect.rbegin(),freqVect.rend());
        vector<int> fin;
        int idx=0;
        while(k--)
        {
            fin.push_back(freqVect[idx].second);
            idx++;
        }
        return fin;
    }
};
// Time & Space Complexity
// Time complexity:
// O(nlogn)
// Space complexity:
// O(n)
```

- Solution 3

- Min-Heap
- Need to Revisit After Getting grip on Heap
- We don't really need to sort whole arr
- We just need to find k freq. so extract k times too..
- heap push O(n)
- pop only k times O(klogn)
  - each pop O(logn)

```
// Solution 3
// Defines the 'Solution' class, which holds our function
class Solution {public:
// Declares the function 'topKFrequent' which takes a vector of integers
'nums'
// and an integer 'k', and returns a vector of integers.
     vector<int> topKFrequent(vector<int>& nums, int k) {
// 1. --- Frequency Counting ---

// Create an unordered_map (hash map) to store the frequency of each
number.
// The 'key' will be the number from 'nums', and the 'value' will be its
count.
    unordered_map<int,int> count;
// Loop through each number ('num') in the input vector 'nums'.
// Increment the count for the current 'num' in the map.
// If 'num' is not in the map yet, it's automatically added with a count of
1.
    for (int num: nums) {count[num]++;}

// 2. --- Min-Heap for Top K ---

// Create a min-heap (priority_queue). This is the clever part!
// It stores pairs: {frequency, number}.
// 'greater<...>' makes it a min-heap, meaning the pair with the *smallest*
// frequency will always be at the top.
    priority_queue<pair<int,int> , vector<pair<int,int>>
,greater<pair<int,int>>> hp; // heap
// Loop through each 'entry' (a {number, frequency} pair) in our 'count'
map.
// Note: 'entry.first' is the number, 'entry.second' is its frequency.
    for ( auto& i : count){

// Push the {frequency, number} pair onto the min-heap.
// We put frequency *first* so the heap sorts by frequency.
        hp.push( {i.second,i.first} ) ;


// This is the key optimization:
// If the heap's size exceeds 'k', we remove the smallest element.
// Since it's a min-heap, 'heap.pop()' removes the element with the
// *lowest frequency* currently in the heap.
        if ( hp.size() > k) {hp.pop();}
    }
```

```
// After the loop, the heap contains exactly the 'k' elements
// with the highest frequencies.

// 3. --- Final Result ---

// Create a vector 'res' to store our final result.
    vector<int> fin;

// Loop 'k' times to extract all elements from the heap.
// (Alternatively, you could use 'while (!heap.empty())')
    for( int i = 0; i<k ; i++){

// Get the top element from the heap (which is a {frequency, number} pair).
// 'heap.top().second' accesses the 'number' part of the pair.
        fin.push_back(hp.top().second);
        // NO fin.push_back(hp.top().second());
        // NO fin.push_back(hp.top.second());
        // NO fin.push_back(hp.top.second);

// Remove the top element from the heap to access the next one.
        hp.pop();
    }

// Return the 'res' vector containing the top k frequent numbers.
// Note: The order in 'res' isn't guaranteed (e.g., it might be
// from k-th most frequent to 1st most frequent), but the problem
// usually allows any order.
    return fin;
    }
};
// Time & Space Complexity
// Time complexity:
// O(nlogk)
// Space complexity:
// O(n+k)
// Where
// n is the length of the array and
// k is the number of top frequent elements.
```

---

7 Encode and Decode Strings [Medium]

- https://neetcode.io/problems/string-encode-and-decode
- https://leetcode.com/problems/encode-and-decode-strings/description/
- https://www.lintcode.com/problem/659/description

> Microsoft, Google, Linkedin, Square, Facebook, Twitter, Bloomberg, Amazon, Meta, Oracle, CrowdStrike, OpenAI, Snowflake

---

Ques

Design an algorithm to encode a list of strings to a single string. The encoded string is then decoded back to the original list of strings.

Please implement encode and decode

Example 1:

```
Input: ["neet","code","love","you"]

Output:["neet","code","love","you"]
```

Example 2:

```
Input: ["we","say",":","yes"]

Output: ["we","say",":","yes"]
```

Constraints:

```
0 <= strs.length < 100
0 <= strs[i].length < 200
strs[i] contains only UTF-8 characters.
```

Recommended Time & Space Complexity You should aim for a solution with O(m) time for each encode() and decode() call and O(m+n) space, where m is the sum of lengths of all the strings and n is the number of strings.

Hint 1 A naive solution would be to use a non-ascii character as a delimiter. Can you think of a better way?

Hint 2 Try to encode and decode the strings using a smart approach based on the lengths of each string. How can you differentiate between the lengths and any numbers that might be present in the strings?

Hint 3 We can use an encoding approach where we start with a number representing the length of the string, followed by a separator character (let's use # for simplicity), and then the string itself. To decode, we read the number until we reach a #, then use that number to read the specified number of characters as the string.

Solutions

- basically
    - design algo ques
    - we have to make 2 funcs
        - 1st convert list to strings

        - 2nd convert strings to list
- WRONG Solution 1 -- brute force
    - encoder
        - string s = ""
        - traverse list
            - append each element in s
                - with special unique seperator "#" help identify element breaks in decoder
        - return s
    - decoder
        - list l = {}
        - string buffer = ""
        - traverse letter in string
            - if element == "#"
                - list.append (buffer)
                - buffer = "" // reset
            - else
                - buffer + = letter
        - return l
    - Problem
        - fails at case : `["we","say",":","yes","!@#$%^&*()"]`
        - our special character would be in input
        - if we do combination like "#HEHE", then character traversal would be pointless

```
// WRONG Solution 1
class Solution {
public:

    string encode(vector<string>& strs) {
        string s = ""; // initialise too to avoid garbage
        for ( auto element : strs ){
            s+=element;
            s+="#";
        }
        return s;
    }

    vector<string> decode(string s) {
        vector<string> strs = {}; // initialise too to avoid garbage
        string buffer = "";
        for ( auto letter : s ){
            if (letter == '#'){ strs.push_back(buffer); buffer="";}
            else {buffer += letter;}
        }
        return strs;

    }
};
```

- Solution 2 -- brute force
    - tweaked solution 1
        - with special seperator would be unique combination of letters MORE THAN 1 CHARACTER
        - eg : "#0#"
        - encoder
            - same as solution 1
            - but
                - "#0#" as seperation
        - decoder
            - same as solution 1
            - but
                - 3 pointer run at same time
                    - `i+2<s.size()` bound safety
                    - `s[i] == '#' && s[i+1] == '0' && s[i+2] == '#'`
                    - also append to buffer and i + = 2
                        - to avoid insertion of #,0s of our special seperator side
    - Passed but, Potential Problem
        - fails at case where input is having same unique seperator by chance

```
// Solution 2
class Solution {
public:

    string encode(vector<string>& strs) {
        string s = ""; // initialise too to avoid garbage
        for ( auto element : strs ){
            s+=element;
            s+="#0#";
        }
        return s;
    }

    vector<string> decode(string s) {
        vector<string> strs = {}; // initialise too to avoid garbage
        string buffer = "";
        for ( int i = 0; i < s.size(); i++ ){
            if ( i+2<s.size()
                && ( s[i] == '#' && s[i+1] == '0' && s[i+2] == '#' )
            ){ strs.push_back(buffer); buffer="";
            i+=2;// nice move by me :)
            }
            else {buffer += s[i];}
        }
        return strs;

    }
};
  // - Passed but, Potential Problem
  //   - fails at case where input is having same unique seperator by
```

```
chance
// Time & Space Complexity
// Time complexity: O(m) for each encode() and decode() function calls.
// Space complexity: O(m+n) for each encode() and decode() function calls.
// Where m is the sum of lengths of all the strings and n is the number of
strings.
```

- Solution 3 -- Illegal
    - using hint 2
    - make a array storing frequencies of each string
    - and at decoder
        - index=0, seperate as in loop we could return to 0 , pointless stuff
        - traversing through array,
            - using its element as for range to traverse
                - append to buffer
            - and append to list
            - buffer reset
            - index++
    - Why illegal ?
        - Breaks the Problem Contract
        - This solution is incorrect because it relies on shared state (a class member variable) to pass information from encode to decode
        - WE HAVE TO CODE STATELESS

```cpp
// Illegal Solution 3
class Solution {
public:
    vector<int> freq = {};

    string encode(vector<string>& strs) {
        string s = ""; // initialise too to avoid garbage
        for ( auto element : strs ){
            s+=element;
            freq.push_back(element.size());
        }
        return s;
    }

    vector<string> decode(string s) {
        vector<string> strs = {}; // initialise too to avoid garbage
        string buffer = "";
        int index=0;
        for ( auto times : freq){
            for ( int i= 0; i< times; i++) {
                buffer+=s[index];
                index++;
            }
            strs.push_back(buffer);
            buffer="";
```

```
        }
        return strs;

    }
};
```

- Solution 4 -- optimised
    - we can store state inside string
        - with our dilimiter
        - and it doesn't matter if another edge case comes
            - as with state
            - we know its not part of state var
    - thought
        - mix of all ways
        - ["neet","4#co#de2"] => "4#neet8#4#co#de2"
            - "4#neet"
                - '4' -> string size
                - '#' to notify that the number is freq
        - then at decoder traverse but
            - if
                - s[i] == number
                - && s[i+1] == '#'
                    - buffer add
                    - index inc
        - NO
            - fails at case of '10#'

```
// Solution 4
class Solution {
public:

    string encode(vector<string>& strs) {
        string s = "" ;
        for ( auto element : strs ) {
            s+=to_string(element.size()) + "#" + element;
        }
        return s;
    }

    vector<string> decode(string s) {
        vector<string> strs = {}; // initialise too to avoid garbage
        string buffer = "";
        int start_idx = 0;
        int freq_buff = 0;
        int freq_check = 1; // 1 = true, 0 = false
        for ( int i = 0 ; i<s.size() ; i++){
            if ( freq_check==1 ){
            if ( s[i] == '#'  ) {
                int end_idx = i-1;
```

```
                freq_buff = stoi( s.substr(start_idx, end_idx-start_idx+1)
);
                freq_check = 0;
            }}
            else if (freq_buff > 0){
                buffer+=s[i];
                freq_buff--;
            }
            else if ( freq_buff == 0){
                freq_check=1;
                strs.push_back(buffer);
                buffer="";
                start_idx=i;
            }

        }
        // --- THIS IS THE FIX ---
        // After the loop, the last word ("you") is still in the buffer.
        // Push it to the vector.
        strs.push_back(buffer);
        return strs;

    }
};

// Time & Space Complexity
// Time complexity: O(m) for each encode() and decode()function calls.
// Space complexity: O(m+n) for each encode() and decode() function calls.
// Where
// m is the sum of lengths of all the strings and
// n is the number of strings.
```

```
class Solution {
public:
    string encode(vector<string>& strs) {
        string res;
        for (const string& s : strs) {
            res += to_string(s.size()) + "#" + s;
        }
        return res;
    }

    vector<string> decode(string s) {
        vector<string> res;
        int i = 0;
        while (i < s.size()) {
            int j = i;
            while (s[j] != '#') {
                j++;
            }
            int length = stoi(s.substr(i, j - i));
```

```
            i = j + 1;
            j = i + length;
            res.push_back(s.substr(i, length));
            i = j;
        }
        return res;
    }
};
```

---

8 Product of Array Except Self [Medium]

- https://leetcode.com/problems/product-of-array-except-self/description/

> Salesforce, ByteDance, Yahoo, ServiceNow, Blackrock, eBay, VMware, Amazon, Microsoft, Oracle, Qualtrics, Adobe, Uber, Nutanix, Tableau, Google, Linkedin, SAP, PayPal, Groupon, Intel, Asana, Snapchat, Grab, Apple, Visa, Evernote, Goldman-sachs, Lyft, Facebook, Splunk, Bloomberg, Cisco, Goldman Sachs, Ibm, Meta, Tiktok, Walmart Global Tech, Zoho, Accenture, Autodesk, CEDCOSS, Disney, Docusign, Flipkart, Infosys, Intuit, Makemytrip, Paytm, Ripple, Sigmoid, Snap, TCS, Tekion, Turing, Unity, WarnerMedia, Wells Fargo, Yandex, ZS Associates

---

Ques

Given an integer array nums, return an array output where output[i] is the product of all the elements of nums except nums[i].

Each product is guaranteed to fit in a 32-bit integer.

Follow-up: Could you solve it in O(n) time without using the division operation?

Example 1:

```
Input: nums = [1,2,4,6]

Output: [48,24,12,8]
```

Example 2:

```
Input: nums = [-1,0,1,2,3]

Output: [0,-6,0,0,0]
```

Constraints:

```
2 <= nums.length <= 1000
-20 <= nums[i] <= 20
```

Recommended Time & Space Complexity : You should aim for a solution as good or better than O(n) time and O(n) space, where n is the size of the input array.

Hint 1 A brute-force solution would be to iterate through the array with index i and compute the product of the array except for that index element. This would be an O(n^2) solution. Can you think of a better way?

Hint 2 Is there a way to avoid the repeated work? Maybe we can store the results of the repeated work in an array.

Hint 3 We can use the prefix and suffix technique. First, we iterate from left to right and store the prefix products for each index in a prefix array, excluding the current index's number. Then, we iterate from right to left and store the suffix products for each index in a suffix array, also excluding the current index's number. Can you figure out the solution from here?

Hint 4 We can use the stored prefix and suffix products to compute the result array by iterating through the array and simply multiplying the prefix and suffix products at each index.

Solutions

- so given
  - array of integers
- to find
  - another array
  - of integers
  - where each index has are product of all other indexes multiply
- edge cases
  - at array containing 0
    - [1,-2,0] => [0,0,-1]
  - at array only having 0
    - [0,0] => [0,0]
  - at array having more than 1 zero
    - [0,4,0] => [0,0,0]
- Solution 1 -- brute force
  - TLE AT LEETCODE
  - let result array int
  - traverse through array i 0 to n-1
    - let buffer int element = 1
    - at traversal each element, start another traversal j 0 to n-1
      - where j is not i
        - buffer * = element at j's index
    - put buffer in result array
    - automatic reset buffer to 1
  - return result array

```
// Solution 1
class Solution {
```

```
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        vector<int> prod(nums.size());
        for (int i=0; i<nums.size(); i++){
            int buff =1;
            for (int j=0; j<nums.size(); j++){
                if(j!=i){
                    buff*=nums[j];
                }
            }
            prod[i]=buff;
        }
        return prod;
    }
};

// Time & Space Complexity
// Time complexity:
// O(n^2)
// Space complexity:
//     O(1) extra space.
//     O(n) space for the output array.
```

- Solution 2 -- illegal
    - not allowed to use division operator
    - thought
        - int totalProd = 1
        - vector int result_array
        - traverse through array
            - totalProd * = array[index]
        - traverse through array
            - result_array[index] = totalProd / array[index]
        - WRONG, fails at case where array containing 0, as it just hides the prod,by multiplying with 0, and not show prod when index contains 0
            - eg
                - intput : [1,0,2,-3]
                - output : [0,-6,0,0]
                - thought's output : [0,0,0,0]
    - thought
        - int totalProd = 1
        - make checker var bool named containZero, default false
        - vector int result_array
        - traverse through array
            - totalProd * = array[index]
            - if array[index]==0 { containZero is true }
        - traverse through array
            - if containsZero is true
                - if at case where zero exist
                    - result_array[index] = totalProd

            - else
                - let non_zero indexes of given array have zero as new result_array
        - if containsZero is false
            - result_array[index] = totalProd / array[index]
    - WRONG, fails at only zero arrays, where we default set prod=1
- int totalProd = 1
- make checker var bool named containNonZeroNumbers, default false
- make checker var bool named containZero, default 0
- vector int result_array
- traverse through array
    - if array[index]!=0
        - totalProd * = array[index]
        - containNonZeroNumbers +=1
    - if array[index]==0 { containZero is true }
- traverse through array
    - if containNonZeroNumbers is ture
        - if containsZero is 1
            - if at case where zero exist
                - result_array[index] = totalProd
            - else
                - let non_zero indexes of given array have zero as new result_array
        - if containsZero is false
            - result_array[index] = totalProd / array[index]

```
// Ilegal Solution 2
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        vector<int> prod(nums.size());
        int totalProd= 1;
        int containZero = 0;
        int containNums = 0;
        for (int i=0; i<nums.size(); i++){
            if (nums[i]==0) {containZero+=1;}
            if (nums[i]!=0) {totalProd*= nums[i];containNums =1;}
        }
        for (int i=0; i<nums.size(); i++){
            if ( containNums ==1 ){
            if (containZero==1){
                if (nums[i]==0){ prod[i]=totalProd;}
                }
            if (containZero==0){
                prod[i]=totalProd/nums[i];
                }
            }

        }
        return prod;
    }
```

```
};

// Time & Space Complexity
// Time complexity:
// O(n)
// Space complexity:
//     O(1) extra space.
//     O(n) space for the output array.
```

```
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        int prod = 1, zeroCount = 0;
        for (int num : nums) {
            if (num != 0) {
                prod *= num;
            } else {
                zeroCount++;
            }
        }

        if (zeroCount > 1) {
            return vector<int>(nums.size(), 0);
        }

        vector<int> res(nums.size());
        for (size_t i = 0; i < nums.size(); i++) {
            if (zeroCount > 0) {
                res[i] = (nums[i] == 0) ? prod : 0;
            } else {
                res[i] = prod / nums[i];
            }
        }
        return res;
    }
};
```

- Solution 3
  - thought
    - the requirement is that, suppose [1,2,3,4] for element 3's result_array
      - we need [1,2, ,4] multiply
      - result_array[that index of 3] = mul. of all prefixes and suffixes of 3 of given array
      - prefix = 1,2
      - suffix = 4
    - so let's make 2 arrays out of [1,2,3,4]
      - product of prefixes => [1,2,6,24]
        - l to r
        - where this array's element = prev array same index element * prev element of this array

        - product of suffixes => [24,24,12,4]
            - r to l
            - from reverse oreder, starting from end, where this array's element = prev array same index element * prev element of this array
        - thus the result array[index] = prefixes[index-1]*suffixes[index+1]
        - result_array => [24,12,8,6]
        - observe it
            - [1,2,3,4] given
            - [1,2,6,24] pre
            - [24,24,12,4] suf
            - [24,12,8,6] res
    - YES, Implement it

```
// Solution 3
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        vector<int> res(nums.size());
        vector<int> pre(nums.size());
        vector<int> suff(nums.size());
        pre[0]=nums[0];
        for ( int i=1; i<nums.size(); i++ ){
            pre[i]=nums[i]*pre[i-1];
        }
        suff[nums.size()-1]=nums[nums.size()-1];
        for ( int i=nums.size()-2; i>0; i-- ){
            suff[i]=nums[i]*suff[i+1];
        }
        for ( int i=0; i<nums.size(); i++ ){
            int preN =1;
            int suffN =1;
            if (i-1>=0) {preN = pre[i-1];}
            if(i+1<nums.size()){suffN = suff[i+1];}
            res[i] = preN * suffN;
        }

        return res;
    }
};
// Time & Space Complexity
// Time complexity:
// O(n)
// Space complexity:
// O(n)
```

```
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        int n = nums.size();
```

```
        vector<int> res(n);
        vector<int> pref(n);
        vector<int> suff(n);

        pref[0] = 1;
        suff[n - 1] = 1;
        for (int i = 1; i < n; i++) {
            pref[i] = nums[i - 1] * pref[i - 1];
        }
        for (int i = n - 2; i >= 0; i--) {
            suff[i] = nums[i + 1] * suff[i + 1];
        }
        for (int i = 0; i < n; i++) {
            res[i] = pref[i] * suff[i];
        }
        return res;
    }
};
```

- Solution 4 -- optimal
  - tweaked version of Solution 3
  - same concept, just save the array space of prefixes and suffixes mul array
  - now
    - [1,2,3,4] given
    - [1,2,6,24(noneed)] pre => [1,2,6] needed nos => [1(obv),1(nums[0]),2,6] arr1
    - [24(noneed),24,12,4] suf => [24,12,4] needed nos => [24,12,4(nums.size()-1),1(obv)] arr2
    - [24,12,8,6] res
      - res[i] = arr1[i] * arr2[i]
      - we can just store res[i]=arr1[i] // correct till this
      - then res[i]=res[i]*arr2[i] // not this because to construct arr2 elements inside res , that process involves null beforehand to get replaced , and if we think of just multiplay the patteren with res updated non-null element instead of replacing, then old values just distrupt the construct
      - instead of making suffixes in array and iterating on it, make independent var suff
        - which updates to get multiplied with res simple

```
// Solution 4
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        vector<int> res(nums.size());
        // vector<int> pre(nums.size());
        // vector<int> suff(nums.size());
        // pre[0]=nums[0];
        res[0]=1;//obv
        res[1]=nums[0];//obv
        for ( int i=2; i<nums.size(); i++ ){
            res[i]=nums[i-1]*res[i-1];
        }
```

```
        // pre array inside res

        // no reverse fashion play like pre impose with arr, because to
construct arr2 elements inside res , that process involves null beforehand
to get replaced , and if we think of just multiplay the patteren with res
updated non-null element instead of replacing, then old values just
distrupt the construct
        // res[nums.size()-1]=res[nums.size()-1]*1; // obv // obv a*1=a
        // res[nums.size()-2]=res[nums.size()-1]*nums[nums.size()-1];//obv
        // // for ( int i=nums.size()-3; i>=0; i-- ){
        // //     res[i]=nums[i+1]*res[i+1]*res[i];
        // // }

        // suff array inside res without involvement of prearray imposed in
it
        int suffN = 1;
        for ( int i=nums.size()-1; i>=0; i-- ){
            res[i]=res[i]*suffN;
            suffN=suffN*nums[i];

        }

        // no need
        // for ( int i=0; i<nums.size(); i++ ){
        //     int preN =1;
        //     int suffN =1;
        //     if (i-1>=0) {preN = pre[i-1];}
        //     if(i+1<nums.size()){suffN = suff[i+1];}
        //     res[i] = preN * suffN;
        // }

        return res;
    }
};
// Time & Space Complexity
// Time complexity:
// O(n)
// Space complexity:
// O(1) extra space.
// O(n) space for the output array.
```

```
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        int n = nums.size();
        vector<int> res(n, 1);

        for (int i = 1; i < n; i++) {
            res[i] = res[i - 1] * nums[i - 1];
        }
```

```
        int postfix = 1;
        for (int i = n - 1; i >= 0; i--) {
            res[i] *= postfix;
            postfix *= nums[i];
        }
        return res;
    }
};
```

---

9 Longest Consecutive Sequence [Medium]

- https://leetcode.com/problems/longest-consecutive-sequence/description/

> Spotify, Wish, Amazon, Apple, Microsoft, Google, Visa, PayPal, Oracle, Qualtrics, Adobe, Nutanix, Uber, Facebook, eBay, Bloomberg, Cisco, Goldman Sachs, Ibm, Meta, SAP, ServiceNow, Tiktok, Walmart Global Tech, Yahoo, Zoho, Atlassian, ByteDance, D. E. Shaw, Deltax, Epam Systems, Flipkart, Infosys, Lyft, Paytm, PhonePe, Roblox, Swiggy, TCS, Turing, UKG, Wissen Technology, Yandex, Zepto

---

Ques

Given an array of integers nums, return the length of the longest consecutive sequence of elements that can be formed.

A consecutive sequence is a sequence of elements in which each element is exactly 1 greater than the previous element. The elements do not have to be consecutive in the original array.

You must write an algorithm that runs in O(n) time.

Example 1:

```
Input: nums = [2,20,4,10,3,4,5]

Output: 4
Explanation: The longest consecutive sequence is [2, 3, 4, 5].
```

Example 2:

```
Input: nums = [0,3,2,5,4,6,1,1]

Output: 7
```

Constraints:

```
0 <= nums.length <= 1000
-10^9 <= nums[i] <= 10^9
```

Recommended Time & Space Complexity

```
You should aim for a solution as good or better than O(n) time and O(n)
space, where n is the size of the input array.
```

Hint 1

```
A brute force solution would be to consider every element from the array as
the start of the sequence and count the length of the sequence formed with
that starting element. This would be an O(n^2) solution. Can you think of a
better way?
```

Hint 2

```
Is there any way to identify the start of a sequence? For example, in [1,
2, 3, 10, 11, 12], only 1 and 10 are the beginning of a sequence. Instead
of trying to form a sequence for every number, we should only consider
numbers like 1 and 10.
```

Hint 3

```
We can consider a number num as the start of a sequence if and only if num
- 1 does not exist in the given array. We iterate through the array and
only start building the sequence if it is the start of a sequence. This
avoids repeated work. We can use a hash set for O(1) lookups by converting
the array to a hash set.
```

Solutions

- given
  - array of nos
- to find
  - max no. of possible numbers which when arranged
    - consecutive
    - inc by exactly 1
- edge cases
  - array is empty
  - array containing negative same elements
  - array containing 2 consecutive sub arrays ,one subarray is larger than other
- Imp. Ques.
- Wrong Solution

- sort the array
- cons=0
- traverse
  - if difference bw 2 elements == 1
  - cons++
- return cons

```
// Wrong Solution
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        sort(nums.begin(),nums.end());
        int cons = 0;
        for (int i=1; i<nums.size(); i++){
            if (nums[i]-nums[i-1]==1){cons++;}
        }
        return cons+1;// +1 as we want nos. not number of differences (i.e.
at for loop we started with 1 :) )
    }
};
```

```
// Wrong Solution
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        if (nums.size()==0){return 0;}
        sort(nums.begin(),nums.end());
        int cons = 0;
        for (int i=1; i<nums.size(); i++){
            if (nums[i]-nums[i-1]==1){cons++;}
        }
        return cons+1;// +1 as we want nos. not number of differences (i.e.
at for loop we started with 1 :) )
    }
};
```

```
// Wrong Solution
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        if (nums.size()==0){return 0;}
        sort(nums.begin(),nums.end());
        int cons = 0;
        for (int i=1; i<nums.size(); i++){
            if (abs(nums[i])-abs(nums[i-1])==1){cons++;}
        }
        return cons+1;// +1 as we want nos. not number of differences (i.e.
```

```
at for loop we started with 1 :) )
    }
};
```

```
// Wrong Solution
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        if (nums.size()==0){return 0;}
        sort(nums.begin(),nums.end());
        int cons = 0;
        for (int i=1; i<nums.size(); i++){
            if (abs(abs(nums[i])-abs(nums[i-1]))==1){cons++;}
        }
        return cons+1;// +1 as we want nos. not number of differences (i.e.
at for loop we started with 1 :) )
    }
};
```

```
// Wrong Solution
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        if (nums.size()==0){return 0;}
        sort(nums.begin(),nums.end());
        int cons = 0;
        vector<int> con;
        for (int i=1; i<nums.size(); i++){
            if (nums[i]-nums[i-1]==1){cons++;}
            if (nums[i]-nums[i-1]>1){con.push_back(cons);cons=0;}
        }
        sort(con.rbegin(),con.rend());
        if(con.size()==0){return cons+1;}// either there is null in the
name of consecutive, or array must be blanck because array was never filled
at case of only one sub array all done in 1st if check
        return con[0]+1;// +1 as we want nos. not number of differences
(i.e. at for loop we started with 1 :) )
    }
};
```

- Wrong Solution
    - store diffrences frequencies
    - return freq of '1'

```
// Wrong Solution
class Solution {
public:
```

```
    int longestConsecutive(vector<int>& nums) {
        if (nums.size()==0){return 0;}
        // vector<int> diff;
        sort(nums.begin(),nums.end());
        unordered_map<int,int> freq;
        for ( int i=1; i<nums.size(); i++){
            // diff.push_back(nums[i]-nums[i-1]);
            freq[nums[i]-nums[i-1]]++;
        };
        return freq[1]+1;
    }
};
```

- Solution 1 -- brute force
  - TLE at LC
  - A consecutive sequence grows by checking whether the next number (num + 1, num + 2, …) exists in the set.
  - The brute-force approach simply starts from every number in the list and tries to extend a consecutive streak as far as possible.
  - For each number, we repeatedly check if the next number exists, increasing the streak length until the sequence breaks.
  - Even though this method works, it does unnecessary repeated work because many sequences get recomputed multiple times.
  - Algorithm
    - Convert the input list to a set for O(1) lookups.
    - Initialize res to store the maximum streak length.
    - For each number num in the original list:
      - Start a new streak count at 0.
      - Set curr = num.
      - While curr exists in the set:
        - Increase the streak count.
        - Move to the next number (curr += 1).
    - Update res with the longest streak found so far.
    - Return res after checking all numbers.

```
// Solution 1 -- brute force

class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        int res=0; // max streak length
        unordered_set<int> store(nums.begin(),nums.end());// set with O(1)
lookups
        for (int num: nums){ // traverse
            int streak =0;//streak
            int curr = num; // current num
            while(store.find(curr) != store.end()){ // if number existe in
set
```

```
                // if yes
                streak++; // inc streak
                curr++; // let curr=cur+1, move to next
            }
            res = max(res,streak);// check with max

        }
        return res;

    }
};
// Time & Space Complexity
//     Time complexity: O(n2)O(n2)
//     Space complexity: O(n)O(n)
```

- Solution 2
  - Sorting
  - Intuition
    - If we sort the numbers first, then all consecutive values will appear next to each other.
    - This makes it easy to walk through the sorted list and count how long each consecutive sequence is.
    - We simply move forward while the current number matches the expected next value in the sequence.
    - Duplicates don’t affect the result—they are just skipped—while gaps reset the streak count.
    - This approach is simpler and more organized than the brute force method because sorting places all potential sequences in order.
  - Algorithm
    - If the input list is empty, return 0.
    - Sort the array in non-decreasing order.
    - Initialize:
      - res to track the longest streak,
      - curr as the first number,
      - streak as 0,
      - index i = 0.
    - While i is within bounds:
      - If nums[i] does not match curr, reset:
        - curr = nums[i]
        - streak = 0
      - Skip over all duplicates of curr by advancing i while nums[i] == curr.
      - Increase streak by 1 since we found the expected number.
      - Increase curr by 1 to expect the next number in the sequence.
      - Update res with the maximum streak found so far.
    - Return res after scanning the entire list.

```

```

```
// Solution 2
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        if (nums.size()==0) return 0;
        sort(nums.begin(),nums.end());
        int
        res = 0,
        curr = nums[0],
        streak = 0,
        i = 0;

        while ( i<nums.size()){
            if(curr!=nums[i]){
                curr=nums[i];
                streak=0;
            }
            while(
                i<nums.size()
                &&
                nums[i] == curr
            ){i++;}
            streak++;
            curr++;
            res=max(res,streak);


        }return res;

    }
};
// Time & Space Complexity

//     Time complexity: O(nlogn)O(nlogn)
//     Space complexity: O(1)O(1) or O(n)O(n) depending on the sorting
algorithm.
```

```
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {

        // Edge case: no elements → no sequence
        if (nums.empty()) return 0;

        // Step 1: Sort the array so consecutive numbers come together
        sort(nums.begin(), nums.end());

        int res = 0;          // stores the longest sequence length found
        int curr = nums[0];   // current expected number in the sequence
        int streak = 0;       // length of the current consecutive sequence
```

```
        int i = 0;            // index pointer

        // Step 2: Traverse the sorted array
        while (i < nums.size()) {

            // If current number does NOT match the expected number,
            // this means the previous sequence is broken
            if (curr != nums[i]) {
                curr = nums[i];   // start a new sequence from nums[i]
                streak = 0;       // reset streak
            }

            // Skip all duplicate values
            // duplicates should not increase streak
            while (i < nums.size() && nums[i] == curr) {
                i++;
            }

            // We successfully found the expected number → extend sequence
            streak++;

            // Update the maximum result
            res = max(res, streak);

            // Now expect the next consecutive number
            curr++;
        }

        return res;
    }
};
// Time & Space Complexity

//     Time complexity: O(nlogn)O(nlogn)
//     Space complexity: O(1)O(1) or O(n)O(n) depending on the sorting
algorithm.
```

- Solution 3
  - TLE at LC
  - Hashset
  - Intuition
    - Avoid recounting the same consecutive sequence multiple times.
    - Start counting only when a number is the beginning of a sequence.
    - A number is a valid start if (num - 1) does not exist in the set.
    - This ensures each sequence is counted exactly once.
    - From a valid start, keep checking num + 1, num + 2, and so on.
    - Each number contributes to the sequence only one time.
  - Algorithm
    - Convert the input list into a set for O(1) lookups.

- Initialize longest to store the maximum sequence length.
- For each number num in the set:
    - If num - 1 is not present in the set:
        - This number is the start of a sequence.
        - Initialize length = 1.
        - While num + length exists in the set:
            - Increment length.
        - Update longest with the maximum value.
- Return longest as the final answer.

```
// Solution 3
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        unordered_set<int> numSet(nums.begin(), nums.end());
        int longest=0;
        for (int num: nums){
            if (numSet.find(num-1) == numSet.end()){
                int length = 1;

                while(numSet.find(num+length)!=numSet.end()){
                    length++;
                }

                longest = max(longest, length);
            }
        }
        return longest;
    }
};

// Time & Space Complexity

//     Time complexity: O(n)O(n)
//     Space complexity: O(n)O(n)
```

```
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {

        // Store all numbers in a hash set for O(1) lookup
        unordered_set<int> numSet(nums.begin(), nums.end());

        int longest = 0;  // stores the maximum consecutive sequence length

        // Iterate through each unique number in the set
        for (int num : numSet) {
```

```
            // Check if this number is the START of a sequence
            // A number is a start only if (num - 1) does not exist
            if (numSet.find(num - 1) == numSet.end()) {

                int length = 1;  // current sequence length starts at 1

                // Extend the sequence forward as long as consecutive
numbers exist
                while (numSet.find(num + length) != numSet.end()) {
                    length++;
                }

                // Update the longest sequence found so far
                longest = max(longest, length);
            }
        }

        // Return the length of the longest consecutive sequence
        return longest;
    }
};

// Time & Space Complexity

//     Time complexity: O(n)O(n)
//     Space complexity: O(n)O(n)
```

- Solution 4 -- optimal
  - hashmaps
  - now using sort() leads to `O(n(log(n)))`
  - here we approach for more optimial way
  - suppose `[100,4,200,1,3,2]`
  - number line visualize
    - <-....1,2,3,4.....100.....200....->
    - 3 sequences of consecutive stuff
      - elements : 4,1,1
    - how would human find a answer?
      - find the bigger sequence
    - how to distinct 3 seqences?
      - observe that all sequences first element don't have it's left neightbour (i.e. element-1)
      - it's obvious
      - use this fact
    - store numbers as set and play with it
    - so process is this :
      - [100,4,200,1,3,2]
      - traverse array

            - is 100 is first element
                - check 100-1 => 99 exists
                    - no
                - Yes
                    - append in sequence as 1st
                    - search for 2nd element
                        - does 100+1 => 101 exists
                            - NO
                            - i.e. sequence ends
            - 4 check
                - 3 yes
                    - its not 1st element
            - 200 check
                - 199 no
                - 201 no
                - single sequence
            - 1 check
                - 0 no
                - 2 yes
                    - 3 yes
                        - 4 yes
                - another sequence
            - total seqence
                - [100],1
                - [200],1
                - [1,2,3,4],4 => longest sequence
    - this algo is effient as it iterate the same element as max twice
    - TC: O(n)
    - SC: O(n)
    - Intuition
        - When inserting a new number, it may extend a sequence on the left, extend a sequence on the right, or merge two sequences together.
        - Instead of scanning forward or backward, we only inspect neighbors.
        - mp[num - 1] gives the length of the sequence ending just before num.
        - mp[num + 1] gives the length of the sequence starting just after num.
        - Adding left length, right length, and 1 gives the merged sequence length.
        - Only boundary values are updated to avoid repeated work.
    - Algorithm
        - Create a hash map mp to store sequence lengths at boundaries.
        - Initialize res = 0 to track the longest sequence.
        - For each number num in the input:
            - If num already exists in mp, skip it.
            - Get left = mp[num - 1].
            - Get right = mp[num + 1].
            - Compute length = left + right + 1.
            - Store length at mp[num].

- Update left boundary: mp[num - left] = length.
- Update right boundary: mp[num + right] = length.
- Update res with the maximum value.

- Return res.

```
// Solution 4
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        unordered_map<int,int> mp;
        int res=0;
        for (int i=0; i<nums.size(); i++){
            int num = nums[i];
            if (!mp[num]){
                mp[num] = mp[num - 1] + mp[num + 1] + 1;
                mp[num - mp[num - 1]] = mp[num];
                mp[num + mp[num + 1]] = mp[num];
                res = max(res, mp[num]);
            }
        }
        return res;
    }
};
// Time & Space Complexity

//     Time complexity: O(n)O(n)
//     Space complexity: O(n)O(n)
```

```
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {

        // Hash map that stores the length of consecutive sequences
        // Only meaningful at sequence boundaries
        unordered_map<int, int> mp;

        int res = 0;  // stores the longest sequence length

        for (int num : nums) {

            // If num already exists, skip it (avoids duplicates)
            if (mp[num]) continue;

            // Length of sequence ending just before num
            int left = mp[num - 1];

            // Length of sequence starting just after num
            int right = mp[num + 1];
```

```
            // New sequence length after inserting num
            int len = left + right + 1;

            // Store the sequence length for current number
            mp[num] = len;

            // Update the LEFT boundary of the merged sequence
            mp[num - left] = len;

            // Update the RIGHT boundary of the merged sequence
            mp[num + right] = len;

            // Update the global maximum
            res = max(res, len);
        }

        return res;
    }
};
// Time & Space Complexity
//     Time complexity: O(n)O(n)
//     Space complexity: O(n)O(n)
```

Two Pointers

TotalCompanyTags

> Wish, Zenefits, Amazon, Apple, Microsoft, Yandex, Google, American Express, Linkedin, Wayfair, Oracle, Cisco, Adobe, Uber, Facebook, eBay, Bloomberg, Deloitte, Goldman Sachs, Meta, PayPal, SAP, ServiceNow, Spotify, Tiktok, Visa, Yahoo, Zoho, Accenture, Axon, Bank Of America, Cadence, Capgemini, Epam Systems, Infosys, Intel, RBC, Shopee, Tinkoff, Toast, Turing, VK, Wipro

---

PreReqs

```
#include<bits/stdc++.h>
int main(){

    return 0;
}
```

- Character Arithmatics Again
  - to check if character is alphanumeric
    - check if character ASCII is under bounds of alphanumeric
    - use && instead of || in each check as it's logically incorrect and char bypass the check

        - check if lies in a to z
            - `( c >= 97 && c <= 122)`
        - else check if lies in A to Z
            - `( c >= 65 && c <= 90)`
        - else check if lies in 0 to 9
            - `( c >= 48 && c <= 57)`

```
bool isCharAlphaNum(char c){
    return (
        ( c>=97 && c<=122 ) ||
        ( c>=65 && c<=90 ) ||
        ( c>=48 && c<=57 )
    );
}
```

- to check if character is alphanumeric smarter way
    - just compare chars

```
bool isCharAlphaNum(char c){
    return (
        ( c>='A' && c<='Z' ) ||
        ( c>='a' && c<='z' ) ||
        ( c>='0' && c<='9' )
    );
}
```

- `isalnum()` checks if char is alphanumeric
- string ops like `tolower()`,`toupper()`, when applies to number chars , throw no errors and skip

---

10 Valid Palindrome [Easy]

- https://leetcode.com/problems/valid-palindrome/

> Wish, Zenefits, Amazon, Apple, Microsoft, Yandex, Google, American Express, Linkedin, Wayfair, Oracle, Cisco, Adobe, Uber, Facebook, eBay, Bloomberg, Deloitte, Goldman Sachs, Meta, PayPal, SAP, ServiceNow, Spotify, Tiktok, Visa, Yahoo, Zoho, Accenture, Axon, Bank Of America, Cadence, Capgemini, Epam Systems, Infosys, Intel, RBC, Shopee, Tinkoff, Toast, Turing, VK, Wipro

---

Ques

Given a string s, return true if it is a palindrome, otherwise return false.

A palindrome is a string that reads the same forward and backward. It is also case-insensitive and ignores all non-alphanumeric characters.

Note: Alphanumeric characters consist of letters (A-Z, a-z) and numbers (0-9).

Example 1:

```
Input: s = "Was it a car or a cat I saw?"

Output: true

Explanation: After considering only alphanumerical characters we have
"wasitacaroracatisaw", which is a palindrome.
```

Example 2:

```
Input: s = "tab a cat"

Output: false

Explanation: "tabacat" is not a palindrome.
```

Constraints:

```
    1 <= s.length <= 1000
    s is made up of only printable ASCII characters.
```

Recommended Time & Space Complexity

```
You should aim for a solution with O(n) time and O(1) space, where n is the
length of the input string.
```

Hint 1

```
A brute force solution would be to create a copy of the string, reverse it,
and then check for equality. This would be an O(n) solution with extra
space. Can you think of a way to do this without O(n) space?
```

Hint 2

```
Can you find the logic by observing the definition of pallindrome or from
the brute force solution?
```

Hint 3

```
A palindrome string is a string that is read the same from the start as
well as from the end. This means the character at the start should match
the character at the end at the same index. We can use the two pointer
algorithm to do this efficiently.
```

Solutions

- Analysis
  - The question asks to check whether a given string is a **palindrome**
    - A palindrome reads the same forward and backward
  - The comparison is **case-insensitive**
    - Uppercase and lowercase letters are treated as the same
  - All **non-alphanumeric characters** must be ignored
    - Spaces, punctuation, and symbols do not affect the result
    - Only letters (A–Z, a–z) and digits (0–9) are considered
  - After ignoring invalid characters and normalizing case
    - The remaining characters form the actual string to check
  - The task is to return:
    - `true` if the cleaned string is a palindrome
    - `false` otherwise
  - eg
    - Input string: "Was it a car or a cat I saw?"
    - Step 1: Remove non-alphanumeric characters → "WasitacaroracatIsaw"
    - Step 2: Convert all letters to lowercase → "wasitacaroracatisaw"
    - Step 3: Compare characters from both ends
      - First and last characters match
      - Second and second-last match
      - This continues until the middle
    - Since all mirrored characters are equal → The string is a palindrome → Output is `true`
- edge cases
  - uppercase lowercase char, mean the same
  - special characters need to ignore
  - spaces need to ignore
  - consider the numbers and letters only
  - at odd cases need to ignore the middle char
- Solution 1 -- brute force
  - similar to 2 pointer approach
  - thought: to check if character is alphanumeric
    - check if character ASCII is under bounds of alphanumeric
    - use && instead of || in each check as it's logically incorrect and char bypass the check
    - check if lies in a to z
      - `( c >= 97 && c <= 122)`
    - else check if lies in A to Z
      - `( c >= 65 && c <= 90)`

        - else check if lies in 0 to 9
            - `( c >= 48 && c <= 57)`
        - YES
    - Initialize an empty string `ss`
        - Traverse each character of the input string `s`
            - If character is a letter (`A–Z` or `a–z`)
                - Convert to lowercase
                - Append to `ss`
            - Else if character is a digit (`0–9`)
                - Append to `ss`
            - Else
                - Ignore the character
    - Check if `ss` is a palindrome
        - Compute `half = length(ss) / 2`
            - For each index `i` from `0` to `half - 1`
                - Compare `ss[i]` with `ss[length(ss) - 1 - i]`
                    - If mismatch found
                        - Return false
    - If all characters match
        - Return true
    - Time Complexity (TC)
        - O(n)
            - One pass to filter characters
            - One pass to compare palindrome
    - Space Complexity (SC)
        - O(n)
            - Extra string used to store alphanumeric characters

```
// Solution 1
class Solution {
public:
    bool isPalindrome(string s) {
        string ss ="";
        for (int i=0; i<s.size(); i++){
            if (
                // s[i]!=' '
                // &&
                    // (
                        (s[i]>=97 && s[i]<= 122) // a to z check
                        // NO to s[i]>=97 || s[i]<= 122 as it's logically
incorrect and char bypass the check
                    ||
                        (s[i]>=65 && s[i]<= 90) // A to Z check
                        // && instead of ||
                    // )
                )
                {
                ss+=tolower(s[i]);
```

```
            }
            else if (s[i]>=48 && s[i]<= 57){
                                        // && instead of ||
                ss+=s[i]; // nos
            }
        }
        // int half = ss.size()%2 == 0 ? (ss.size()/2) : ((ss.size()/2));
        int half = ss.size() / 2 ;
        for (int i=0; i<half; i++){
            if(ss[i]!=ss[ss.size()-1-i]) return false; // remember
ss.size()-1 is last index, not ss.size()
        }

        // if (ss.size%2==0){ //even alphanumeric chars

        // }
        // else if (ss.size%2==0){ //odd alphanumeric chars

        // }
        return true;
    }
};

//   - Time Complexity (TC)
//     - O(n)
//       - One pass to filter characters
//       - One pass to compare palindrome
//   - Space Complexity (SC)
//     - O(n)
//       - Extra string used to store alphanumeric characters
```

- `isalnum()` checks if char is alphanumeric
- string ops like `tolower()`,`toupper()`, when applies to number chars , throw no errors and skip

```
// Solution 1
class Solution {
public:
    bool isPalindrome(string s) {
        string ss ="";
        for (int i=0; i<s.size(); i++){
            if (
                isalnum(s[i]) //`isalnum()` checks if char is alphanumeric
                )
                {
                ss+=tolower(s[i]);
            }
        }
        // int half = ss.size()%2 == 0 ? (ss.size()/2) : ((ss.size()/2));
        int half = ss.size() / 2 ;
        for (int i=0; i<half; i++){
            if(ss[i]!=ss[ss.size()-1-i]) return false; // remember
```

```
ss.size()-1 is last index, not ss.size()
        }

        // if (ss.size%2==0){ //even alphanumeric chars

        // }
        // else if (ss.size%2==0){ //odd alphanumeric chars

        // }
        return true;
    }
};

//   - Time Complexity (TC)
//     - O(n)
//       - One pass to filter characters
//       - One pass to compare palindrome
//   - Space Complexity (SC)
//     - O(n)
//       - Extra string used to store alphanumeric characters
```

- Solution 2
  - Reverse String
  - Initialize an empty string `ss`
    - Traverse each character of the input string `s`
      - If the character is alphanumeric
        - Convert it to lowercase
        - Append it to `ss`
  - Create a reversed copy of the cleaned string
    - Copy `ss` into `rss`
    - Reverse `rss`
  - Compare both strings
    - Traverse from index `0` to `ss.size() - 1`
      - If any character in `ss` does not match the corresponding character in `rss`
        - Return false
  - If all characters match
    - Return true
  - Time Complexity (TC)
    - O(n)
  - Space Complexity (SC)
    - O(n)

```
// Solution 2
class Solution {
public:
    bool isPalindrome(string s) {
        string ss ="";
        for (int i=0; i<s.size(); i++){if (isalnum(s[i]))
ss+=tolower(s[i]);}
```

```
        string rss = ss;
        reverse(rss.begin(),rss.end());
        for (int i=0; i<ss.size(); i++){if ( ss[i] != rss[i]) return
false;}
        return true;
    }
};
//   - Time Complexity (TC)
//     - O(n)
//   - Space Complexity (SC)
//     - O(n)
```

```
class Solution {
public:
    bool isPalindrome(string s) {
        string newStr = "";
        for (char c : s) {
            if (isalnum(c)) {
                newStr += tolower(c);
            }
        }
        return newStr == string(newStr.rbegin(), newStr.rend());
    }
};
```

- Solution 3 -- optimal
  - Two Pointers
  - Solution 3 is better than Solution 1
    - Solution 1 may have slight better time complexity but solution 3 have better memory complexity
    - thought
      - even though this seems as Solution 1
        - but, this is better
        - because even you go till half of solution 1
        - but, this is better, as it goes till the step where mismatch comes
        - but thinking of that its same, goes till the step where mismatch comes in Solution 1
          - but Solution 1 is strict after lot of alterations
          - Solution 3 is Dynamic
        - YES
    - another thought
      - this Solution 3 is better than Solution 1 as it doesn't uses the another array
      - and it doesn't play on the half cal
        - as using `i<j`
          - at `even` case,
            - then `s[i=leftmiddle]` and `s[j=rightmiddle]` are compared

        - & at `odd` case,
            - then `s[i=middle-1]` and `s[j=middle+1]` are compared
            - and `s[middle]` is leave out as there is no need to check it in odd array for palindrome
    - and it doesn't uses outside func like `isalnum()`
    - YES
- Algorithm: Valid Palindrome (Two Pointers)
    - Initialize pointers i = 0 (start) and j = n - 1 (end)
    - While i < j
        - While i < j and s[i] is non-alphanumeric, increment i
        - While i < j and s[j] is non-alphanumeric, decrement j
        - If lowercase(s[i]) != lowercase(s[j]), return false
        - Increment i and decrement j
    - Return true
    - Time Complexity: O(n)
    - Space Complexity: O(1)

```
// Solution 3
class Solution {
public:
    bool isPalindrome(string s) {
        int i =0;
        int j =s.size() -1;

        while (i<j){
            while(i<j && !isCharAlphaNum(s[i])){i++;}// NOT IF. Because if skips only one invalid character, but while skips all consecutive invalid characters before comparison. and if you proceed after if thinking that something inc. that is logical error as it inside inc by 1 and then at end too execute and inc by 1
            while(i<j && !isCharAlphaNum(s[j])){j--;}
            if(isCharAlphaNum(s[i]) && isCharAlphaNum(s[j]))
{if(tolower(s[i])!=tolower(s[j])) return false;}

            i++;j--;// after checking and found out no mismatch, we ought to move to next
        }
        return true;
    }

    // bool isCharAlphaNum(char c){
    //     return (
    //         ( c>=97 && c<=122 ) ||
    //         ( c>=65 && c<=90 ) ||
    //         ( c>=48 && c<=57 )
    //     );
    // }

    bool isCharAlphaNum(char c){
        return (
```

```
            ( c>='A' && c<='Z' ) ||
            ( c>='a' && c<='z' ) ||
            ( c>='0' && c<='9' )
        );
    }

};

    // - Time Complexity: O(n)
    // - Space Complexity: O(1)
```

```cpp
class Solution {
public:
    bool isPalindrome(string s) {
        int l = 0, r = s.length() - 1;

        while (l < r) {
            while (l < r && !alphaNum(s[l])) {
                l++;
            }
            while (r > l && !alphaNum(s[r])) {
                r--;
            }
            if (tolower(s[l]) != tolower(s[r])) {
                return false;
            }
            l++; r--;
        }
        return true;
    }

    bool alphaNum(char c) {
        return (c >= 'A' && c <= 'Z' ||
                c >= 'a' && c <= 'z' ||
                c >= '0' && c <= '9');
    }
};
```

Template

Topic

TotalCompanyTags

> Total Company Tags

PreReqs

```cpp
#include<bits/stdc++.h>
int main(){

    return 0;
}
```

Sr.No. Question [Easy/Medium/Hard]

- Link

> Company Tags, Ctrl+Shift+Alt+ArrowKeys

Ques

Content

Solutions

```

```

- Prompt for PreReqs of My Multiple Resources(courses, notes, papers)

```
You are an expert C++ tutor. Your goal is to teach me the necessary
concepts to solve a specific coding problem, but without revealing the
final answer's logic. I will provide you with both the problem description
and the complete, working C++ solution. Based on this information, you must
first identify all the core programming concepts, data structures, and C++
features used in the solution. Then, for each of these topics, you must
explain it to me from a beginner's perspective. Each explanation should
cover what the concept is, why it's useful in general, its basic C++
syntax, and a small, self-contained code snippet that demonstrates only
that single concept in isolation. It is crucial that you do not explain the
line-by-line logic of the solution I provided or combine your examples into
the final answer. Your entire purpose is to give me the individual building
blocks so that I can construct the final solution myself.
```

- Prompts for Notes of My Multiple Resources(courses, notes, papers)

```
Create super depth notes in Markdown (.md) format with 100% information
preserved, no loss. Use simple grammar and keep everything clear, direct,
and well-structured. using headings, subheadings,paragraphs, statements and
code blocks when needed. Include every detail, definition, example, and
step exactly from the source. transform the given content into clean,
```

```
readable .md format.
and no #, just nested - lines plaintext
```

TotalCompanyTags

TotalCompanyTags

> Baidu, Airbnb, Netease, Cisco, Amazon, Aetion, Box, Mathworks, Zoom, Google, Cloudera, Intel, Indeed, Godaddy, Walmart Global Tech, Salesforce, Didi, Affirm, Vmware, Yandex, Microsoft, Adobe, Alibaba, Jpmorgan, Linkedin, Citadel, Emc, Groupon, Intuit, Twitter, Nvidia, Twilio, Valve, Expedia, Yahoo, Zoho, Bookingcom, Wish, Zillow, Morgan-stanley, Drawbridge, Paypal, Huawei, Dropbox, Radius, Zomato, Roblox, Accenture, Goldman-sachs, Lyft, Yelp, Splunk, Bloomberg, Samsung, Bytedance, Servicenow, Quora, Goldman Sachs, Blackrock, Ebay, Ge-digital, Oracle, Qualcomm, Tencent, Uber, Tableau, Spotify, Morgan Stanley, American Express, Sap, Ibm, Deutsche-bank, Snapchat, Dell, Apple, Visa, Works-applications, Facebook, Factset, Audible, Google, Adobe, Facebook, Twilio, Salesforce, Affirm, Docusign, Yahoo, Cisco, Servicenow, Blackrock, Goldman Sachs, Ebay, Vmware, Tiktok, Bookingcom, Electronic-arts, Amazon, Wish, Microsoft, Yandex, Oracle, Qualtrics, Bloomberg, Adobe, Alation, Uber, Nutanix, Jpmorgan, Tesla, Mathworks, Zulily, Google, Hulu, Ibm, Snapchat, Apple, Intuit, Visa, Goldman-sachs, Yelp, Facebook, Walmart Global Tech, Bytedance, Yahoo, Cisco, Vmware, Ebay, Pocket-gems, Amazon, Microsoft, Oracle, Adobe, Uber, Spotify, Google, Linkedin, Hulu, Snapchat, Apple, Goldman-sachs, Yelp, Facebook, Bloomberg, Microsoft, Google, Linkedin, Square, Facebook, Twitter, Bloomberg, Amazon, Meta, Oracle, CrowdStrike, OpenAI, Snowflake, Salesforce, ByteDance, Yahoo, ServiceNow, Blackrock, eBay, VMware, Amazon, Microsoft, Oracle, Qualtrics, Adobe, Uber, Nutanix, Tableau, Google, Linkedin, SAP, PayPal, Groupon, Intel, Asana, Snapchat, Grab, Apple, Visa, Evernote, Goldman-sachs, Lyft, Facebook, Splunk, Bloomberg, Cisco, Goldman Sachs, Ibm, Meta, Tiktok, Walmart Global Tech, Zoho, Accenture, Autodesk, CEDCOSS, Disney, Docusign, Flipkart, Infosys, Intuit, Makemytrip, Paytm, Ripple, Sigmoid, Snap, TCS, Tekion, Turing, Unity, WarnerMedia, Wells Fargo, Yandex, ZS Associates, Spotify, Wish, Amazon, Apple, Microsoft, Google, Visa, PayPal, Oracle, Qualtrics, Adobe, Nutanix, Uber, Facebook, eBay, Bloomberg, Cisco, Goldman Sachs, Ibm, Meta, SAP, ServiceNow, Tiktok, Walmart Global Tech, Yahoo, Zoho, Atlassian, ByteDance, D. E. Shaw, Deltax, Epam Systems, Flipkart, Infosys, Lyft, Paytm, PhonePe, Roblox, Swiggy, TCS, Turing, UKG, Wissen Technology, Yandex, Zepto, Wish, Zenefits, Amazon, Apple, Microsoft, Yandex, Google, American Express, Linkedin, Wayfair, Oracle, Cisco, Adobe, Uber, Facebook, eBay, Bloomberg, Deloitte, Goldman Sachs, Meta, PayPal, SAP, ServiceNow, Spotify, Tiktok, Visa, Yahoo, Zoho, Accenture, Axon, Bank Of America, Cadence, Capgemini, Epam Systems, Infosys, Intel, RBC, Shopee, Tinkoff, Toast, Turing, VK, Wipro

---

End-of-File

The KintsugiStack repository, authored by Kintsugi-Programmer, is less a comprehensive resource and more an Artifact of Continuous Research and Deep Inquiry into Computer Science and Software Engineering. It serves as a transparent ledger of the author's relentless pursuit of mastery, from the foundational algorithms to modern full-stack implementation.

> Made with 💚 Kintsugi-Programmer