**Data Structure notes**

**1. Arrays**

* **Array rotations -> rotate an array arr [ ] of size d by d elements (d = d % n): using temp array, rotating one by one rotation , juggling algo (gcd) , reversal algo.**
* **Kadane’s algo, Moore’s Voting algo, merge two sorted arrays, two-pointer approach problems, Dutch flag (0,1,2) ,**
* **Binary Search (Most important) –**
* **Number system -> binary to decimal , decimal to binary ,**
* **Try to use two-pointer approach in questions, where D.S. (mainly array) is sorted.**

**2. Linked Lists**

* **Reverse (iter. / recur.) , loop detect / remove , palindrome ,**
* **Linked lists are always made in heap , we access them through head pointer ( head is in stack ) . Save memory leaks (delete the node if not required) .**
* **Always check for memory out of bounds. Only do p = p -> next ,if p != nullptr .**
* **Always write case for n == 0 and n == 1 , required or not.**
* **Always make functions for specific things , otherwise one function becomes mess in large solution .**
* **Check for corner cases i.e. n == 0 , 1 and 2 . In palindrome type question check for odd / even length too .**
* **Avoid using extra space i.e. array or some other space .**
* **In a singly linked list , if we have to perform some operation which requires us to traverse the LL from tail to head , then reverse the linked list and then perform the operation .**
* **We use the concept of dummy node, when we have to keep a track of previous ptr, which can raise null ptr exceptions, if we keep it null. (see merge two sorted lists question)**
* **Problems -> reverse ( recursive + iterative ), find mid (first mid and second mid), detect cycle, find starting point of cycle, merge two sorted lists, palindromic linkedlist, fold-unfold, quicksort, mergesort,**

**2. Graphs**

* **Traversals -> BFS, DFS, Cycle detection (BFS and DFS), Bipartite graph coloring (BFS and DFS),**

**3. Recursion**

* **Two methods in recursion – 1. Faith - Expectation (basic) 2. Level and options**
* **Faith-Expectation method – a) High level thinking + b) Low level thinking**
* **High level thinking (Recursive case)-> Write your function’s output (Expectation) , write output with a smaller input (Faith) .Generally we write f(n) and f(n-1), but it can also be f(n-2), f(n/2), f(n-3) or something like that.**
* **Establish relation b/w Expectation and faith assuming faith already knows the answer (100% confidence) .**
* **Low level thinking (Base case)-> use stack, DRY run, or think of smallest valid input .**
* **Find out if we can solve the question at calling time or returning time .**
* **If we have both options , then check the function header and solve accordingly .**
* **If our favorable function header is not there, then we should make another utility function to proceed .**
* **In array you can use func(arr+1,n-1) method (we consider arr[0]th element in this and perform operations on it ), but not in vectors(as vec+1 operation is not allowed). In vectors we can use func(vec, n-1) and consider vec[n-1]th element. Also in vectors , we can use an index i which should reflect in function header .**
* **Recursion -> Ek step main le leta hu sare possible options ko consider karke , baki choti problem apne aap solve ho jaegi.**
* **Array -> sum, display (reverse), target sum subset, get subsequence**
* **Some good questions -> Tower of Hanoi , combination sum , place tiles (using fibonacci like sequence), optimal binary string, friends pairing problem, staircase , variations of fibonacci ,**
* **BackTracking -> Flood fill, N - Queens, Knights tour, rat in a maze, sudoku solver, m-coloring,**

**Tips:**

* **Always draw the diagram in any question, to make more clarity and DRY RUN is a MUST.**
* **Try to write comments about steps in solving a problem, then implement it.**
* **Always check if the variables are long long or int, if in doubt , use long long .**
* **For characters, try to use single quotes, instead of double quotes.**
* **Use a two-pointer approach to reduce complexity from O(n2) to O(n).**
* **In a problem about palindromes, try reversing .**
* **We can separate assignment statements using , and write them in a single line .**
* **Try to use a two-pointer approach in questions, where D.S. (mainly array) is sorted.**
* **Comparator function should be public, otherwise for a user-defined class, you should define your own comparator.**
* **sort( ) function comparator asks should the first element come before the second ??**
* **Always make the comparator function static.**
* **Variable’s names should be intuitive for better understanding -> Pascal case – variables , Camel case – functions .**
* **Give space after statement and then semicolon ; Use space wherever possible .**
* **Integer reverse method is important.**
* **cout << setprecision(x) << fixed << number ; --> Sets the number of digits after decimal as x ( only setprecision(x) sets the number of digits as x (including the part before the decimal ) ) .**
* **Other manipulators -> dec , hex , oct**
* **for each loop fails on stack , queue and priority\_queue as they don’t allow access to all elements without changing the container .**
* **Learn the basic structure of pattern printing questions.**
* **A processor can do 108 operations in one second, analyse time complexity accordingly .**
* **Use mathematical equations in your code after testing them for C++. Handle division ( / ) very carefully.**
* **Syntax of typedef : typedef** <**existing\_name**> <**alias\_name**> .

**5. Trees**

* **Questions : DFS( preorder, inorder, \*\*postorder), BFS(level order) both iterative and recursive , zigzag traversal, build tree from traversals , symmetric tree , mirror tree , views of a tree , diameter , height balanced , sum replacement , LCA , shortest distance b/w 2 nodes, flatten a tree, max. path sum ,**
* **Questions (BST) : build (preorder, array, sorted array) , search and delete, \*\*isBST() , catalan no. zigzag traversal**
* **Catalan number – 2nCn / (n+1) .**
* **In binary trees and BST, we generally have two recursive calls, one for left and other for right . Try to solve problem in terms of these recursive calls. It increases the prob. Of solving if problem has a recursive solution .**
* **Check 3 base cases -> a node with i) 1 child ii) 2 child iii) 0 child (leaf node) .**
* **Try to think that this problem will be solved in which traversal (DFS-3, BFS-1).**
* **The inorder traversal of BST gives us values in sorted order .**

**6. Binary Search**

* **If our search space is either montonically increasing or montonically decreasing, then think of applying binary search .**
* **Two types of Binary Search , a) we know what element to find b) We don’t know what element to find – we traverse until the while loop breaks .**
* **Find the search space , find low and high and find our criteria .**
* **Use mid = low + (high-low)/2 ; instead of mid = (low+high)/2 ; to avoid overflow .**
* **binary\_search(it1,it2,key) is an inbuilt STL function works in a sorted sequence, returns a bool.**
* **Try to solve problems in terms of existing problems .**
* **One pattern – book allocation , aggressive cows (must do), painter’s partition .**

**STRINGS: -**

* **Contiguous storage -> just like array , but that storage can shift according to the need of elements .**
* **Internallly, string uses a dynamically allocated array (character) .**
* **We can use it = it + c as the memory is contiguous .**
* **Basics -> s.push\_back(x) , s.pop\_back() , s.append() s.size() , s.empty() , s[index] , s.at(index) , s.front() , s.back() .**
* **s.substr(pos, len) returns a string starting from index pos and having length len.**
* **Use s.push\_back(‘a’) , and not s.push\_back(“a”).**
* **We can append a single character using append() , += operator and push\_back().**
* **To add a full string at the end, use append() or += .**
* **Syntax of append : append(string\_name, starting\_index, ending\_index) .**
* **More functions -> s.erase(index, length) , s.insert(index, string) ,**
* **Use** **strtok (str, ” ”)** **to** **extract words from str string word by word (delimiter – “ “) . It returns a char pointer.**
* **str1.compare(str2) returns 0 if both str1(C++ string) and str2 (C++/c-style string) are equal.**
* **stoi(s) -> converts string s (C, C++) to integer.**
* **atoi(x) -> converts x integer to C string.**
* **to\_string(x) -> converts x (integer) to C++ string.**
* **In C++ strings the objects are compared in dictionary (alphabetical / lexical / ASCII value) order.**

**PAIR :-**

* **pair <T1,T2> p ; -> Initialisation . Ex : pair < string , int > pr { “Akku” , 23 } ;**
* **p.first , p.second are the elements of this pair .**
* **make\_pair(10,20) ; -> returns a pair of first as 10 and second as 20 .**

**Dynamic Programming :**

* **Basic problems -> Fibonacci , climbing stairs (with 2 jumps / variable jumps / minimum moves) ,minimum cost traversal maze , gold-mine, LCS,**
* **Intermediate -> Target Sum subsets, Coin change (permut. / comb.) , 0/1 Knapsack , subset sum, equal sum subset partition, count subsets, minimise the difference of sum of partition, partition with a given difference, target sum, unbounded knapsack , LCS variations, count encoding**
* **Challenging -> MCM, egg dropping, scrambled string,**

**VECTORS:**

* **Contiguous storage -> just like array , but that storage can shift according to the need of elements .**
* **Internally, vectors use a dynamically allocated array .**
* **We can use it = it + c as the memory is contiguous .**
* **Basics -> v.push\_back(x) , v.pop\_back() , v.size() , v.empty() , v[index] , v.at(index) .**
* **More functions -> v.insert(it,element) , v.insert(it, it1, it2) , v.erase(it) , v.erase(it1, it2) , v.clear() , v.front() , v.back() .**
* **2-D vector initialisation -> vector <vector <int>> vec (n, vector <int> (m,0)) ;**

**LINKED LISTS (STL):**

* **list <T> L ; -> Initialisation .**
* **Not contiguous , we cannot use it = it + c , but it++ is valid .**
* **We cannot directly access element like in array , vectors and strings .**
* **Implemented internally as Doubly Linked List.**
* **Insertion -> L.push\_back (x) , L.pop\_back() , L.push\_front(x) , L.pop\_front() , L.insert(it,x) , Complexity : O(1)**
* **Erase -> L.erase(it) or L.erase(it1 , it2) . Complexity : O(1) or O(n) .**
* **Remove -> L.remove(x) removes all occurences of x from the list L . Complexity: O(n)**
* **Remove if -> L.remove\_if(func) ; func is a boolean function which return true or false .**
* **To iterate over list -> 1. for loop 2. range based for loop 3. while loop .**
* **Other functions -> L.size( ) , L.empty( ) , L.clear( ) , L.front( ) , L.back( ) -> return references to 1st and last element .**
* **list <T> L -> doubly linked list , forward\_list <T> L -> singly linked list .**

**CONTAINER ADAPTORS 🡪 Use other containers as adpators (interface) for their implementation .**

**STACK :-**

* **Stack doesn’t allow access to all elements , we can only access top element , for each loop fails on stack .**
* **LIFO -> Last In First Out .**
* **If no container class is specified , then stack is implemented using dequeue.**
* **stack <T> st ; -> Initialiasation .**
* **Insertion -> st.push(x) – O(1) . Deletion -> st.pop( ) – O(1) ,**
* **st.top( ) – returns reference to the top element .**
* **Other functions -> st. size( ) , st.empty( ) .**
* **We can use stack in those questions where the brute-force approach gives O(n2) solution , and the inside loop variable(j) depends on outer loop variable .**
* **When using stack, try to think in terms of which direction to traverse and what elements to pop and push in it .**
* **Problems -> NGR, NGL, NSR, NSL, Rainwater trapping, Histogram,**

**QUEUE :-**

* **Queue doesn’t allow access to all elements , we can only access front and back element only , for each loop fails on stack .**
* **FIFO -> First In First Out .**
* **If no container class is specified , then stack is implemented using dequeue.**
* **queue <T> qu ; -> Initialiasation .**
* **Insertion -> qu.push(x) – O(1) . Deletion -> qu.pop() – O(1) ( front element is deleted ) .**
* **qu.front (), qu.back() – returns reference to the front and back element of the queue .**
* **Other functions -> qu.size() , qu.empty() .**

**PRIORITY QUEUE :-**

* **priority\_queue <int> maxh ; -> Max Heap initialisation . Maximum value element is at the top .**
* **priority\_queue <int , vector <int> , greater <int> > minh ; -> Min Heap intialisation . Minimum element at top .**
* **Operations -> pq.top() , pq.push() , pq.pop() , pq.size() , pq.empty() .**
* **A comparator is a third argument in the initialisation syntax of a priority queue .**
* **Problems -> smallest/ largest k elements , points having k smallest distances from origin/ given point ,**

**SET (set , unordered\_set , multiset , unordered\_multiset ) 🡪 Store key values**

* **set <key> S ; -> Initialisation .**
* **Not contiguous , we cannot use it = it + c , but it++ is valid .**
* **Implemented internally as a binary search tree .**
* **Unique keys , no duplicates, ordered key ( sorted ) , Insertion – O(logn) , Accessing – O(logn) , Deleting O(logn) .**
* **Insertion -> S.insert(a) ; a is of type T . Complexity : O(logn) .**
* **Find -> auto it = S.find(a) ; It will return S.end( ) if key not found . Complexity : O(logn) .**
* **Erase -> S.erase(key) or S.erase(iterator) . Complexity : O(logn) .**
* **To iterate over set -> 1. for loop 2. range based for loop 3. while loop .**
* **Other functions -> S.size() , S.empty() , S.clear() , S.upper\_bound(key) , S.count(key) .**

**UNORDERED SET ->**

* **unordered\_set -> implemented as Hash Table .**
* **If order of elements doesn’t matter , prefer using unordered\_set .**
* **unordered\_set elements are not sorted (unordered) .**
* **Insertion O(1) , Accessing O(1) , find O(1) .**
* **Don’t use complex datatypes as key in unordered\_set as their hash function may not be defined .**
* **Unordered set is useful when we want to get only unique (distinct) values .**

**MULTISET ->**

* **Almost similar to set .**
* **Difference -> duplicate keys are allowed .**
* **S.erase(it) deletes one element , S.erase(key) deletes all elements same as key , in case of duplicate elements .**
* **S.find(it) returns the iterator of the first element in case of duplicate elements .**

**MAP -> Associative Container , Stores (key, value) pair**

* **map <key , value > M ; 🡪 Initialisation**
* **Not contiguous , we cannot use it = it + c , but it++ is valid .**
* **maps -> Implemented internally as Binary-Search Tree .**
* **Elements of a map -> pair <T1, T2> . T1 -> key , T2 -> value .**
* **Insertion -> 1. M[key] = value , 2. M.insert(make\_pair(a, b)) , 3. M.insert({a, b}) .**
* **Find -> auto it = M.find(key) ; It will return M.end( ) if the key not found .**
* **Erase -> M.erase(key) or M.erase(iterator) . Complexity : O(logn)**
* **We can use iterator to iterate over the map :**

**auto it = M.begin(); it -> first = x; or (\*it).first = x ;**

* **To iterate over map -> 1. for loop 2. range based for loop 3. while loop .**
* **M[key] ; makes a pair with default value’s value as either 0 , 0.0 , “” acc. to int , double or string (as value declared)**
* **Map -> unique keys , no duplicate keys , ordered key , Insertion O(logn) , Accessing ele O(logn) , find O(logn)**
* **Other functions -> M.size() , M.empty() , M.clear() , M.upper\_bound(key) , M.count(key) .**

**UNORDERED MAP ->**

* **unordered\_map -> implemented as Hash Table .**
* **If order of elements doesn’t matter , prefer using unordered\_map .**
* **elements are not sorted (unordered) .**
* **Insertion O(1) , Accessing O(1) , find O(1) .**
* **Don’t use complex data types as key in unordered\_map as their hash function may not be defined .**

**MULTIMAP ->**

* **Almost similar to a map .**
* **Difference -> duplicate keys are allowed .**

**Bitwise Manipulation ->**

* **\_\_builtin\_popcount(x) : Used to count the number of set bits (1’s) in x.**

**Algorithms ->**

**\*min\_element(forwardIteratorFirst, forwardIteratorLast) -> returns the iterator to smallest element in the range [first,last).**

**CODEFORCES ->**

* **Div 2 < 1900 , Div 1 > 1900 .**

**CODECHEF ->**

* **Div 3 <1600 , 2000 > Div 2 > 1600 , Div 1 > 2000 .**

**Quantitative Aptitude : Work done question , H.C.F. – L.C.M. ,**

**Qualitative Aptitude : Blood relation , sitting arrangement ,**

**GCC -> GNU Compiler Collection (previously GNU C Compiler ) , doesn’t link C++ libraries .**

**G++ -> GNU C++ Compiler , link C++ libraries automatically .**