Standard STL Methods

```
string
            length(); clear();
            append(str);
            push_back(char);  insert(Posn, Char);
            find(str, posn, size), find (char, podn)

 queue <int> q;
             push();   pop();   front();   back();   size();   empty();
 stack <int> s;
             push();   pop();   top();   size();  empty();
 vector <int> s;   vector<int> s(100, 0); //100 elements with 0
             s[i];  front();  back();   size();  empty();
             push_back();   erase(it);
             begin();   end();  rbegin();  rend();
  list <int> l
             l[i], empty(); size(); front(); back();
             push_back(); push_front(); pop_back(); pop_front();
             begin(); end(); rbegin(); rend();
            sort(); reverse(); merge(l2); remove(23);
 map<int, bool> lm;
            m[i];  empty(); size();
            find(); count()
        map<int, bool>::iterator it = lm.begin();
        it->first;  it->second;
 bitset<N> b;
             reset();  set();  set(i, bool);  flip(i, bool); flip();
              b[i] ; size();   count() -> number of 1s  ; size() - count() -> number of 0s
              none(); //returns if none of the bits are set
              test(i);
 priority_queue<int> Q;
             push(); pop(); top(); empty();

 ifstream (filename, ifstream::in|out|app|binary)
             get();  peek(); getline(s, 256);
             open(), close();
             good();
             while (infile >> i) //read integers from a file

 struct Comparator<T>
 {
       bool operator(T a, T b) { if (a > b) {return true;} else {return false;} }
 }
```

```
struct MyHash<T>
{

}
```

Algorithm:

```
sort(v.begin(), v.end(), comp);    -> inline bool comp(int a, int b) { if (a> b) true; else fse}
fill ( v.begin(), v.end(), 0) ; // init vector with all zeros
```

Stacks and Queues

- Implement a queue using stacks
- Expression evaluation: 1+2-3/5;
- Implement a stack using a queue
- Implement a stack- you should be able to extract the minimum value in O(1) operation
- Write a list class, given that you have only a stack
- Sort using 2 stacks
- Towers of Hanoi - for N=3, N= any K

Strings

- Count the number of words in a string - split by space
- Replace all spaces by '%20'
- Given s1, s2 - remove all characters which are present in s2 from s1 (in pac
- hexToDecimal(), dectoHex()
- strcpy, strrev (ascii, utf8), strdedup
- atoi, itoa, atof , HARD -> ftoa
- Anagrams of a group of strings
- HARD: Given a large string, find a sub-string (needle in a haystack)
  -> KMP algo ababac
- HARD: String is matching a particular regular expression or not
  -> Partial match possible

Arrays

- Maximal Sub Array
- Sub Array which sums to zero
- Find 2 numbers in an integer array that sum to X
- Print largest sub-array with elements in an increasing order
- Given an array of size N with numbers 0 to N-1, find a l the repeating numbers
- Find 3 digits in an integer array which sum to zero
- Given an array with a1, a2, a3.. an, b1, b2... bn , produce output of the form a1, b1, a2, b2 … an, bn
- Given an array of the form 0s, 1s and 2s - separate all 0s, 1s and 2s in the form

0..01...12..2
- Given 3 months of stock quotes for MSFTin a chronological order, find the buying selling price of the so that you can maximize the profit

Searching
- Square root of an integer/double
- Binary Search for exact number, if you can't find - find the number lower than the key K
- int power(N,M)
- Rotated binary search
- An array has been rotated by k positions - find k
- Search for an interval which is completely enclosed in another interval, given a set of intervals (2,3)
- Nth largest element in an array - N log K
- Find the median of a billion numbers present in N different machines

2 D Arrays
- Print a 2 D array from top left to middle
- Given a matrix, how will you compute the sum of a rectangle given tl, tr, bl, br
- How will you rotate a matrix by 90/180/270 degrees etc
- Given that all the rows are sorted and columns are sorted, how will you print it in a sorted manner
- Given a 2 D matrix with 1s and 0s. Find an i such that, there is an ith row, with all 1s and ith col with all 1s

Linked List
- Reverse a linked list
- Reverse a linked list (recursion)
- Find and delete node with value = k
- Find if 2 lists intersect or not, Find nth element from the tail of the list
- Loop in a linked list - detect the loop

Trees
- In-order traversal (iterative and recursive)
- Print based on BFS
  (a) level by level
  (b) left to right, and right to left
- Connect all the children of the same level of a tree
- Given pre-order, in-order, form the tree
- Check if a given tree T2 is a sub-tree of T1 or not
- If a tree is a mirror of itself / fold able
- Check if a binary tree is a BST or not
- Check if a tree is balanced or not // find minht, max ht. Make sure diff is not > 1
- Determine the height of a tree
- Determine the diameter of a tree

- BST to doubly linked list
- Find the Least Common Ancestor of 2 nodes
- Successor, Predecessor of a given node
- Deletion of a node
- Print all the paths which sums up to a value
- Count the number of trees with N nodes
- Count number of non-leaf Nodes

Graphs

- Write a function to check if a graph is bi-partite or not
- Graph traversal - check if a route exists between 2 nodes
- Shortest path from (i,j) to (m,n)
- Minimal spanning tree
- Euler tour exists or not
- Hamiltonian path exists or not

Other Data Structures(Heap/Suffix tree)

- Longest Common Sub-sequence of 2 large text strings
- Merge N sorted lists
- Find the median of numbers present in N different machines
- Find a given string in a Trie

Math Style

- Given a plane and N points, find the line which passes through the maximum number of points
- Check if 2 rectangles intersect or not

Bit Operation

- Number of bits which are ones in a number
- Swap 2 variables
- Find the missing number in N numbers
- Number of bit required to change one number M to another number N :-> M xor N
- Swap the odd and even bits of a integer
- Find the maximum of 2 numbers
- How do you check if N is a power of 2 or not -> (n & n-1 == 0)

Probability

- Pick K random elements from N elements from an array
- Pick K random elements from a continuous stream of elements - N is not known
- Shuffle an array - make sure no element in its position again

Recursion

- Fibonacci
- Telephone directory look up: Given number, find the strings corresponding to it

- Permutation (recursive and non-recursive)
- Combination of numbers/string (recursive and non-recursive)
- Valid Combination of Parenthesis Closing e.g 2 -> ()(), (())\
- Subset of an array
- Number of paths taken by a Robot - it can move down or right in a maze
- Given a puzzle and a word, check if the word is present in the puzzle or not

Dynamic Programming
- Longest increasing sub sequence
- Knapsack problem
- Minimum Number of Coins which make up C
- Edit Distance Algorithm (Levenshtein Distance)

Sorting
- Merge two sorted Arrays
- Quick sort :
  Partition Function
- Heap sort : Sift-up,   Sift-down

Fun Questions
- Pascals Triangle
- Assume you have a chess program, given a board class, a piece class, how will you figure if it check mate or not
- You are a robber - rob as many houses as possible, maximize the cash you have
- Histogram of n items stacked against each other. Find Max area under rectangle
- Given 2 events, each with a start and end time, implement a boolean check to see if they overlap
- 25 racehorses, 5 tracks, no stop watch - how will you find the top 3 fastest horses in fewest number of races
- Given a score s, and points p1, p2, ..pn, - what are the list of combinations which add up to s

Other
- Multiply 2 large numbers
- Dot product of 2 large vectors

C++/Java Based Questions
- Virtual Function/ Pure Virtual Function
- Inheritance/Interface
- Polymorphism - function overloading
- Function overriding
- V-Table
- Garbage Collection
  (i) Mark and Sweep
  (ii) Reference Counting

(iii) Smart Pointers

- Design Pattern: Singleton, Factory
- Template
- Functors - Anonymous Delegates
- Casting - Static, Dynamic, Reinterpret

Threading

Java Questions - mostly syntax related

- final
  - final variable - can not change the value for primitives, can not change the reference once assigned (though you could change the state of the final object)
  - final method - can not override this method in subclass
  - final class - can not override this class
- finalize
  - this is rarely used, as you don't know the certainty of when this will be executed.
  - You free resource in this code primarily related native execution.
  - When there is no reference for an object, and when garbage collection is running, then the finalize() method for that object will be run. And the next run of garbage collection will free the memory occupied by this object in heap.
  - You don't close db/connection object or IO operations in this, these needs to be in the finally{} block of try.
- Exception Handling
  - Throwable - super class for all errors and exception
  - Error - These are situation you don't handle in your code as these are abnormal conditions.
  - Exception - Two types RuntimeException (aka UnCheckedException), and CheckedException
  - RuntimeException - a sub class of Exception, normally means bug in the program. Program will compile and run even when these exceptions are not handled using try{} catch{}. Eg. NullPointerException, ArrayIndexOutOfBoundException...
  - CheckedExceptions - All exceptions except sub-classes of RuntimeException. These must be handled using try{} catch{}.

  -
    ```
    try{
    //a code might throw AnException
    }catch (AnException e) {
    //Handle exception
    }finally {
    //This piece will be executed whether exception occurs or not
    }
    ```
    try {} must have a following catch{} block or a finally block.

    If an exception is not handled in the current method, this exception is thrown to caller of the current method, till somewhere it is handled or program exits with this exception it is not handled anywhere.

- AccessModifiers
  public - everyone can access
  private - only members of the class can access
  Only inner classes can be made private. Top level class can not be private
  <default> - no modifier means, all classes in the same package (or directory) can access
  protected - Same as default, in addition subclasses in different package can access
  Top level class can not be protected.

- instanceof
  - <Object ref > instanceof <Class Name> returns true if the object is an instance of the specified class or its subclass
- getClass()
  Returns the Class of the object. Normally used in reflection and equals() method

Collections in Java

- Array
  - Always boundary checked, throws ArrayIndexOutOfBoundException.
- Collection - Super interface of List and Set, Queue
- List - Stores objects in sequence as they are entered/accessed
  - common operations: add(E e), remove(), get(int index)
  - ArrayList, Vector (synchronized form), LinkedList
- Set - Stores unique objects in a sequence (uniqueness is determined by equals())
  - HashSet, TreeSet
- Queue - FIFO
  - add(E e), remove(), element() - -> throws exception
  - offer(E e), poll(), peek() -> returns special values
  - LinkedList, ArrayDeque
- Deque - FIFO and LIFO - subclass of Queue
  - Use when a stack is needed
- Map - Stores key value pairs.
  - common operations: put(key,value)
  - HashMap, HashTable(synchronized form), TreeMap.
- Hash collections:
  - hashcode() and equals() methods of key should be overridded.
  - objects that are equal should have identical hashcodes
- Tree collections:
  - objects that are added should be comparable. Natural ordering (using Comparable) or Comparator should be provided.
- Collections.synchronized[List,Set]()
  - Wrap Collection object with this method and synchronize your access to this collection in a multi-threaded env.

Answers

```
void ExpressionEvaluation(string s)
{
  stack <int> operand;
  stack <char> operator;

  for (int i=0; i < s.length(); i++)
  {
       if (IsOperator(s[i])
       {
             if (IsLowerPrecedence(a[i], operator.top())
             {
                 char oper = operator.pop();
                 int operand1 = operand.pop();
                 int operand2 = operand.pop();
                 int val = Evaluate(oper, operand1, operand2);
                 operand.push(val);
             }
             else
             {
                 operator.push(a[i]);
             }
       }
       else
       {
             int operand1 = ReadOperand(s, &i);
             operand.push(operand1);
       }
  }

  while (!operator.empty())
  {
        char oper = operator.pop();
        int operand1 = operand.pop();
        int operand2 = operand.pop();
        int val = Evaluate(oper, operan1, operand2);
        operand.push(val);
  }
  return operand.pop();
}
```

```
 int hToD(string s)
           {
                int retVal =0;
                for (int i=0 ; i < s.length-1; i++)
                {
                    int tempVal = hexCharToVale(s[i]);
                    retVal += retVal + tempval * 16;
                }
         }
         100 -> 100%16; 100/16

       string decToHex(int a)
       {
           string s="";
            while ( a > 0)
            {  temp = a % 16;
               char tC = Lookup(temp)l
               a = a / 16;
               s = tC + s;
            }
            return strrev(s);
        }

bool MatchPattern(string Pattern, string regexstring)
{
      if (patttern = = "" && regexstr == "") { return tru; }
      //base conditions
      if (pattern[0]== "." && paten1].==  ".")
     {
           if (MatchPattern(pattern +2, regexstring +1)) retun true;
          if  (MatchPattern(patter, regexstring + 1)) return true;
          retur false;
     }
     else if (pattern[0] == regexstring[0])
    {
         MP(patern+1, regexstring+1);
   }

}


Node * ReverseList(Node *head)
{
    Node *curr = head;
    Node *prev = NULL;
     if (curr == NULL ||cur->next == NULL ) {return curr;}
    Node *fut = curr->next;
    while (curr->next != NULL)
   {
           curr->next = prev;
           prev = curr;
           curr = fut;
            fut = fut->next;
     }
     return prev;
}

Node * ReverseList(Node *head)
{
  if (head == NULL || head->next == NULL) {return head; }
   Node *tmp = head;
   Node *tail = tmp->next;
   Node *nrev = ReverseList(tmp->next);
   tail->next = head;
   head->next = NULL;
  return nrev;
}

InsertHeap(int arr[], int length, int key)
{

       arr[length] = key;
       child = length;
       parant = child/2;
       while (arr[parent] > a[child])
      {
           swap(arr[parent], arr[child]);
           child = parent;
           parent = child/2;
      }
}
```