http://stackoverflow.com/questions/16958390/difference-between-head-and-head-pointers-in-c

sSelection Sort has O(n2) time complexity making it inefficient on large lists

```
public class SelectionSort {

	public static int a[] = {6,4,9,3,1,7};

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		int  i ,j;
		for(i=0;i<a.length-1;i++) {
			int min = i ;
			for ( j = i+1 ; j<a.length; j++){
				if (a[j] < a[min]){
					min = j;
				}
			}
			if (min !=i) {
				int temp = a[i];
				a[i] = a[min];
				a[min]= temp;
			}
		}
		for (i = 0 ; i<a.length;i++){
			System.out.println("a : " + a[i]);
		}

	}
}
```

Find the minimum element in the remainder of the array for this we do swap outside the inner loop

```
public class BubbleSort {

	public static int a[] = {32,26,23,45,40,1};

	public static void main(String[] args) {

		for (int i = 0 ; i <a.length ; i++){
			for (int j =i+1 ; j <a.length ; j++){
				if(a[j] < a[i] ) {
					int temp = a[i];
					a[i] = a[j];
					a[j] = temp;
				}
			}
		}
		for (int i = 0 ; i < a.length ; i++) {
			System.out.println(a[i]);
		}
		// TODO Auto-generated method stub

	}
```

```
}
```

Sum of digits
While(n!=0)
Remainder = n % 10
Sum = sum + remainder
N = n / 10

Balancing Paranthesis

https://www.youtube.com/watch?v=QZOLb0xHB_Q

```
import java.util.Stack;


public class BalanceParanthesis {

	public static  String input[] =  {"{","(",")","}","["};

	public  boolean arePair(String open, String close) {
		if (open == "{" && close == "}") return true;
		if (open == "(" && close == ")") return true;
		if (open == "[" && close == "]") return true;
		return false;
	}
	public boolean checkForBalancedParanthesis(String[] input) {
		Stack<String> stack = new Stack<String>();
		for (int i=0 ; i<=input.length-1;i++) {
			if(input[i] == "{" || input[i] == "(" || input[i] == "[") {
				stack.push(input[i]);
			}
			else if(input[i] == "}" || input[i] == ")" || input[i] ==
"]" ) {
				if ( arePair(stack.lastElement(),input[i]) == true ) {
					stack.pop();
				}
				else
					return false;
			}
		}

		return stack.empty() ? true : false ;

	}


public static void main(String[] args) {
	// TODO Auto-generated method stub
	BalanceParanthesis bp = new BalanceParanthesis();
	boolean val = bp.checkForBalancedParanthesis(input);
	System.out.println(val);

}

}
```

Associativity happens when both the operators of same precedence are there we evaluate from left to right

Stacks can be implemented using arrays and linked lists
For empty stack top is set as -1
Array size is immutable it cannot be changed and it cannot be deleted
Array implementation of stacks

```
	public static int top = -1;


	public   void push(String x) {
		top = top+1;
		arr[top] = x;

	}
	public void pop() {
		top = top-1;
	}
```

To return the top element arr[top]
To check if stack is empty if(top == -1) empty
Cannot pop if top ==-1

Bubble Sort

```
for (int i = 0 ; i <a.length-1 ; i++){
			for (int j =i+1 ; j <a.length ; j++){
				if(a[j] < a[i] ) {
					int temp = a[i];
					a[i] = a[j];
					a[j] = temp;
				}
			}
		}
```

Selection Sort

```
for(i=0;i<a.length-1;i++) {
			int min = i ;
			for ( j = i+1 ; j<a.length; j++){
				if (a[j] < a[min]){
					min = j;
				}
			}
			if (min !=i) {
				int temp = a[i];
				a[i] = a[min];
				a[min]= temp;
			}
		}
```

Stack Implementation using Linked Lists :-
Insertion can be done at the head or tail O(n)
Deletion also can be done at the head or tail O(1)

Time complexity of String reveerse using stack is 0(n)
Space Complexity = O(n)

Dynamic Memory Allocation means an executing program requests for a block of main memory .It happens in heap
This memory can be used to create new nodes or memory for a new object
Stack Overflow – Memory cannot grow during run time .

Dynamic Memory Allocation in C :-
Malloc
Calloc
Realloc
Free
Malloc returns a pointer to the address location on heap
Int *p;
P = (int *)malloc(sizeof(int))
*p = 10

Malloc creats void pointer so we need to do type casting

Deferencing of pointer
*p returns the value at the address it is pointing to .
Int a = 5
Int *p
P = &a
Print *p // Value at address of a = 5
P → address
*p = value at address

Int a = 10
Int *p = &a; // p = address
*p = value at address returns 10

P+1 increments p by 4 bytes since its int

Int **q → is pointer to pointer

Call by value

```
void increment(int x) {
   x = x+1;
}
int main() {

   int x = 20;
   increment(x);
   printf("value of a %d\n",x);
}
```

1) First main is called in to stack with its variables . Here x = 20
2) Then increment is called and x = 21
3) Again main is called and previous function from stack is removed and x = 20

Call by reference

```
void increment(int  *x) { // here *x = &x from main method
   *x = *x + 1; // Here *x is value at address +1 . so x is incremented to 11
}
int main() {

   int x = 20;
   increment(&x); // Passing address of x
   printf("value of a %d\n",x);
}
```

```
public class RecFib {


	public int fib(int n) {
		if (n <= 1) return n;
		else
			return fib(n-1)+fib(n-2);
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		RecFib rs = new RecFib();
		for(int i=1 ; i<=5; i++) {
		System.out.println(rs.fib(i));
		}


	}
}
```

```
Sum of digits
While(n!=0)
Remainder = remainder % 10
Sum = sum + remainder
N = n / 10
```

Associativity happens when both the operators of same precedence are there we evaluate from left to right

Stacks can be implemented using arrays and linked lists
For empty stack top is set as -1
Array size is immutable it cannot be changed and it cannot be deleted
Array implementation of stacks

```
	public static int top = -1;


	public   void push(String x) {
		top = top+1;
		arr[top] = x;

	}
	public void pop() {
		top = top-1;
	}
```

To return the top element arr[top]
To check if stack is empty if(top == -1) empty
Cannot pop if top ==-1

Bubble Sort

```
for (int i = 0 ; i <a.length-1 ; i++){
			for (int j =i+1 ; j <a.length ; j++){
				if(a[j] < a[i] ) {
					int temp = a[i];
					a[i] = a[j];
					a[j] = temp;
				}
			}
		}
```

Selection Sort

```
for(i=0;i<a.length-1;i++) {
			int min = i ;
			for ( j = i+1 ; j<a.length; j++){
				if (a[j] < a[min]){
					min = j;
				}
			}
			if (min !=i) {
				int temp = a[i];
				a[i] = a[min];
				a[min]= temp;
			}
		}
```

Stack Implementation using Linked Lists

Counting Change

```
public class CountingChange {

    private static int amount = 132;
    public static void main(String[] args) {
    // With 25,10,5,1
    System.out.println("Q : " + computeCoin(25) );
```

```
    System.out.println("D : " + computeCoin(10) );
    System.out.println("N : " + computeCoin(5) );
    System.out.println("P : " + computeCoin(1) );

    }

    public static  int computeCoin(int coinValue) {

        int val = amount / coinValue;
        amount = amount%coinValue;
       // amount -= val * coinValue;

        return val;
    }
}
```

Longest Palindrome Subsequence

```
// Everay single character is a palindrom of length 1
L(i, i) = 1 for all indexes i in given sequence

// IF first and last characters are not same
If (X[i] != X[j])  L(i, j) =  max{L(i + 1, j),L(i, j - 1)}

// If there are only 2 characters and both are same
Else if (j == i + 1) L(i, j) = 2

// If there are more than two characters, and first and last
// characters are same
Else L(i, j) =  L(i + 1, j - 1) + 2
```

http://www.geeksforgeeks.org/dynamic-programming-set-12-longest-palindromic-subsequence/

Overlapping subproblems

Time Complexity of DP – O(n^2)

Dynamic Programming – http://ajeetsingh.org/2013/11/12/find-longest-palindrome-sub-sequence-in-a-string/

```
public class LongestPalindromeSubstring {

	public int max(int x,int y) {
		return x>y ? x : y;

	}

	public int lps(char[] a , int i, int j) {
		if (i ==j) return 1;
		if ((a[i] == a[j]) && (i+1==j)) return 2;
		if (a[i] == a[j]) return lps(a, i+1, j-1) +2;
		return max(lps(a,i+1,j),lps(a,i,j-1));
```

```
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub

		String s = "forgeeksskeegfor";
		char[]  c = s.toCharArray();
		LongestPalindromeSubstring lps = new LongestPalindromeSubstring();
		System.out.println(lps.lps(c,0 ,c.length-1));

	}

}
```

Dynamic Programming

```
public static int getLongestPalindromicSubSequenceSize(String source){
    int n = source.length();
    int[][] LP = new int[n][n];

    //All sub strings with single character will be a palindrome of size 1
    for(int i=0; i < n; i++){
        LP[i][i] = 1;
    }
    //Here gap represents gap between i and j.
    for(int gap=1;gap<n;gap++){   //-------> 1
        for(int i=0;i<n-gap;i++ ){// --------> 2
                    int j=i+gap; // ---------> 3
                    if(source.charAt(i)==source.charAt(j) && gap==1)
                        LP[i][j]=2;
                    else if(source.charAt(i)==source.charAt(j))
                        LP[i][j]=LP[i+1][j-1]+2;
                    else
                        LP[i][j]= Math.max(LP[i][j-1], LP[i+1][j]);
         }
    }
    return LP[0][n-1];
}
```

Subsequence is formed by deleting some characters in the string and reordering the remaining characters

The worst-case running time of any correct algorithm must be at least $O(N)$, where N is the length of the string. This is because the algorithm cannot be correct unless it can verify whether the entire string is a palindrome (because in that case it is itself the longest palindromic substring), and it is not possible to conclude that a string is palindromic unless every character is examined.

Longest Common Subsequence in PERL

```
use Algorithm::Diff qw/ LCS /;

my @a = split //, 'thisisatest';
my @b = split //, 'testing123testing';

print LCS( \@a, \@b );
```

http://stevekrenzel.com/articles/longest-palnidrome
For odd length palindrome fix a center at the middle .
For even length palindrome fix two centers low and high and expand in both directions
.
Suffix tree is compressed trie of all the suffixes as keys and their positions as values .
http://www.geeksforgeeks.org/pattern-searching-set-8-suffix-tree-introduction/
A generalized suffix tree is a suffix tree made for a set of words instead of only for a single word. It represents all suffixes from this set of words. Each word must be terminated by a different termination symbol or word.

Diff utility in unix makes use of longest common subsequence

Longest Common Subsequence

```
 for (i=0; i<=m; i++)
   {
     for (j=0; j<=n; j++)
     {
       if (i == 0 || j == 0)
         L[i][j] = 0;

       else if (X[i-1] == Y[j-1])
         L[i][j] = L[i-1][j-1] + 1;

       else
         L[i][j] = max(L[i-1][j], L[i][j-1]);
     }
   }

   /* L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1] */
   return L[m][n];
}
Print Longest common subsequence
int index = L[m][n];

   // Create a character array to store the lcs string
   char lcs[index+1];
   lcs[index] = '\0'; // Set the terminating character

   // Start from the right-most-bottom-most corner and
   // one by one store characters in lcs[]
   int i = m, j = n;
```

```
   while (i > 0 && j > 0)
   {
      // If current character in X[] and Y are same, then
      // current character is part of LCS
      if (X[i-1] == Y[j-1])
      {
          lcs[index-1] = X[i-1]; // Put current character in result
          i--; j--; index--;     // reduce values of i, j and index
      }

      // If not same, then find the larger of two and
      // go in the direction of larger value
      else if (L[i-1][j] > L[i][j-1])
         i--;
      else
         j--;
   }

   // Print the lcs
   cout << "LCS of " << X << " and " << Y << " is " << lcs;
```

$$LCS\left(X_i, Y_j\right) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ LCS\left(X_{i-1}, Y_{j-1}\right) + 1 & \text{if } x_i = y_j \\ \text{longest}\left(LCS\left(X_i, Y_{j-1}\right), LCS\left(X_{i-1}, Y_j\right)\right) & \text{if } x_i \neq y_j \end{cases}$$

https://www.youtube.com/watch?v=P-mMvhfJhu8

Method 2 (Use Dynamic Programming)
We can avoid the repeated work done is the method 1 by storing the Fibonacci numbers calculated so far.

```
#include<stdio.h>

int fib(int n)
{
  /* Declare an array to store fibonacci numbers. */
  int f[n+1];
  int i;

  /* 0th and 1st number of the series are 0 and 1*/
  f[0] = 0;
  f[1] = 1;

  for (i = 2; i <= n; i++)
  {
      /* Add the previous 2 numbers in the series
         and store it */
      f[i] = f[i-1] + f[i-2];
  }

  return f[n];
}

int main ()
```

```
{
  int n = 9;
  printf("%d", fib(n));
  getchar();
  return 0;
}
```

Time Complexity: **O(n)**
Extra Space: **O(n)**

Space complexity for series without using third variable to swap is O(1)
http://www.geeksforgeeks.org/program-for-nth-fibonacci-number/

http://web.engr.illinois.edu/~jeffe/teaching/algorithms/notes/05-dynprog.pdf

recursive fib = T(n-1)+T(n-2)

towers of Hanoi
   1) Move n-1 disks from Origin to intermediate

   2) Move the nth disk from origin to destination

   3) Move the n-1 disks from intermediate to destination (recursive)

2(n-1)+1 time complexity

```
public void moveHanoi(int n , char origin, char dest , char inter) {
		if (n == 1) {
			System.out.println( origin + " -> " +dest);
		} else {
			moveHanoi(n-1, origin ,  dest , inter);
			System.out.println(origin + "->" + dest );
			moveHanoi(n-1,inter,origin,dest);
		}
	}
```

Permutations

```
public class Permuatitions {

	static char a[] = {'1','2','3','4'};

	public int perm(char a[], int k  , int n) {
		if(k==n) {
				System.out.println(a);

		} else {
			for(int j = k;j<n;j++) { // Fixing the first spot
				swap(k,j);
				perm(a,k+1,n);
```

```
                        swap(k,j); // swap undo so that the elements remain
same for next iteration
                    }
                }
                return n;

        }
        public void swap(int k ,int n) {
                char temp = a[k];
                a[k]=a[n];
                a[n]=temp;
        }

        public static void main(String[] args) {
                // TODO Auto-generated method stub

                Permuatitions p = new Permuatitions();
                p.perm(a,0,a.length);

        }

}

permutation(prefix + str.charAt(i),  str.substring(0, i) + str.substring(i+1));
```

http://introcs.cs.princeton.edu/java/23recursion/Permutations.java.html
http://learnprogramming.machinesentience.com/java_permutations_recursion/

Queue

1) Push or Enqueue
2) Pop or dequeue
3) Is empty
4) Full
5) Peek or Front

Insertion and deletion should happen at opposite ends . Insert and rear and remove at front
Time complexity is O(1)

Implementation of Queue using Array

```
public class ArrayQueue {

        static int a[] = new int[5];
        static int rear = -1;
        static int front = -1;

        public boolean isEmpty() {
                if(rear == -1 && front == -1 ) {
                        return true;
                } else return false;
        }
        public boolean isFull() {
                if (rear ==a.length-1) return true;
                else return false;

        }
```

```
	public void enQueue(int x) {
		if (isFull()==true ) return ;
		else if (isEmpty() == true) {
			rear = front = 0;
		} else {
			rear = rear+1;
		}
		a[rear] =x;
public void deQueue() {
		if (isEmpty() == true) return;
		else if (front == rear ) {
			rear = front = -1;
		} else {
			front = front+1;
		}
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		ArrayQueue q = new ArrayQueue();
		q.enQueue(5);

		q.enQueue(6);
		q.enQueue(7);
		q.enQueue(8);
		q.deQueue();
		System.out.println("rear " + rear + front);
		q.enQueue(10);
		for(int i=0;i<5;i++ )
		System.out.println(a[i]);
	}

}
```

In circular array we do (rear+1)%N so that we don't waste the empty index
Dequeue - > front = (front+1)%N

Linked List http://www.cprogramming.com/tutorial/c/lesson15.html

Binary tree is a tree that is has zero nodes or at the most every node has at the most two children
The maximum number of nodes in a binary tree of depth k is 2^k-1

Binary search tree is a binary tree . It may be empty . When it is not empty , it has to satisfy three conditions :-

1) Every node should have a unique key
2) The keys for left nodes should be less than the root
3) The keys for right nodes should be more than the root
4) The left and right subtrees are also binary trees

Stack stores function calls and local variables . Stack size is fixed .
Heap is not fixed . It can be requested . Example malloc .

To insert an element at nth position go to n-1th position and insert

Tmp2 = malloc
Tmp2->next = tmp->next
Tmp->next = tmp2

To delete an element at nth position in an linked list :-

```
struct node *head;
struct node *temp1;

if (n==1) {
head = temp1->next;
free (temp1);

else {
for (int i = 0; i<n-2;i++){
temp1 = temp1->next;
}
Struct node *temp2 = malloc
temp2 = temp1->next;
temp1->next = temp2->next;
free(temp2);
}
```

Delete nth element in Double Linked list :-

```
if(temp->next==NULL) {

temp->prev->next=NULL;

} else {

temp->next->prev=temp->prev;

temp->prev->next=temp->next;

} free(temp);
```

Insertion in a linked list using stack :-

```
Top ;
Temp = malloc
Temp->data = 5
Temp->next = top
Top = temp // Update top
```

Delete Last Node in a Linked List (De Queue)

```
while(temp->n->n !=NULL) // Iterate over two nodes at once
            {
                temp=temp->n;
            }
            temp2=temp->n;
            temp->n=NULL;
            free(temp2);
```

http://www.mytechinterviews.com/singly-linked-list-delete-node

You are given a pointer to a node (not the tail node) in a singly linked list. Delete that node from the linked list

1. First make the node to be deleted to point the next node.
2. Copy the content of the next node in to the node that has to be deleted.
3. Assign the next pointer of the newly copied node to the next pointer of the node from which the content has been copied.
4. Delete the node from which the data has copied.

```
void deleteNode( Node * node )
{
    Node * temp = node->next;
    node->data = node->next->data;
    node->next = temp->next;
    free(temp);
}
O(1) time Complexity
```

Doubly Linked List can be used for reverse look up

Insert in a DLL :-

```
While(tmp!=null) {
      tmp = tmp->next
}
Tmp2 = malloc
Tmp2->data = data
Tmp2->next =  tmp->next
Tmp2->prev =  tmp
Tmp->next->prev =  tmp2
Tmp->next  =  tmp2
```

Circular Linked List : http://codepad.org/XkdvFimF

Here :-

```
while(tmp->next!=head) {

      tmp = tmp->next;
}
Last = tmp->next;
Last = malloc;
Last->data = data
Last->next = head // If insert at end
head = last  // Update head if insert in the beginning
tmp->next = head // Update address of last node to head – create link between last and head
```

If head is null create a new node and point it to itself

```
(*head)->next = *head
```

Delete a node based on key

```
if (head->data == key) {
       struct node* temp = head;
       head= head->next;
       free(temp);

    }
    else {

         struct node* temp = head;
         while (temp->next->data !=key)
             temp = temp->next;

         struct node* temp2 = temp->next;
         temp->next = temp2->next;
         free(temp2);

    }
```

Insertion Sort Algorithm

Using marker . Shift elements from unsorted to sorted list

```java
public class Insertionsort {


	public static int n[] = {7,2,9,3,5,1};
	public static void main(String[] args) {
		// TODO Auto-generated method stub

		for(int i=1; i <=n.length-1 ; i++  ) {
			int temp = n[i];
			int hole = i;
			while( hole > 0 && temp < n[hole-1] ) {


				n[hole] = n[hole-1];
				hole = hole-1;

			}
			n[hole] = temp;
		}
		for(int i=0 ; i<=5 ; i++)
		System.out.println(n[i]);
	}

}
```

Mergre Sort :-

```java
public class MergeSort {

	public static int b[] = {6,2,3,1,9};

	public void MergeSort(int a[],int left[], int right[]) {

		int i = 0,j = 0,k = 0;

		int l = left.length;
		int r = right.length;
		int v = a.length;

		while ( i < l && j < r) {
			if( left[i] < right[j]) {
				a[k] = left[i] ;
				i++;
			} else {
				a[k] = right [j];
				j++;
			}
			k++;
		}
		while ( i < l ) {
			a[k] = left[i];
			k++;
			i++;

		}
```

```
		while ( j < r ) {
			a[k] = right[j];
			k++;
			j++;

		}

	}


public void Merge(int a[], int length) {

		int n = length;
		if(n < 2) return;
		int mid = n/2;

		int left[] = new int[mid];
		int right[] = new int[n-mid];
		for (int i = 0 ; i <mid ; i++) {
			left[i] = a[i];
		}
		for (int i = mid ; i <n ; i++) {
			right[i-mid] = a[i];
		}

		Merge(right,n-mid);
		Merge(left,mid);
		MergeSort(a, left, right);

	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub


		MergeSort ms = new MergeSort();
		ms.Merge(b,b.length);
		for(int i=0 ; i < b.length ; i++)
		System.out.println(b[i]);

	}

}
```

https://gist.github.com/mycodeschool/9678029 - Insertion Sort

Quick Sort O(nlogn) average case
O(n^2) – worst case
IN Place sorting algorithm

O(n logn) → Time complexity of Merge Sort → Worst Case . Space Complexity O(n) {Not IN Place algorithm}

Insertion sort best case is completely sorted array TC = O(n)
Insertion sort worst and average case TC = O(n^2) = Bubble Sort = Selection Sort
Insertion sort is In-place but Merge Sort in not In-Place

- In-place; i.e., only requires a constant amount O(1) of additional memory space

Quick Sort

http://www.algolist.net/Al gorithms/Sorting/Quicksort

```
int partition(int arr[], int left, int right)
{
      int i = left, j = right;
      int tmp;
      int pivot = arr[(left + right) / 2];

      while (i <= j) {
            while (arr[i] < pivot)
                  i++;
            while (arr[j] > pivot)
                  j--;
            if (i <= j) {
                  tmp = arr[i];
                  arr[i] = arr[j];
                  arr[j] = tmp;
                  i++;
                  j--;
            }
      };

      return i;
}

void quickSort(int arr[], int left, int right) {
if(left > = right ) return ;
      int index = partition(arr, left, right);
     // if (left < index - 1)
            quickSort(arr, left, index - 1);
     // if (index < right)
            quickSort(arr, index, right);
}
```

Heap is a nearly complete binary tree . Nearly complete binary tree is at the lowest level not completely filled

Parent = floor[i/2]
Left = 2i ; right = 2i+1

Max heap → If A[Parent] >= A[i] . Root will have max value in Max Heap
Min Heap → If A[Parent] <= A[i] . min element is ROOT in min head

Heapify:-
http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/heapSort.htm

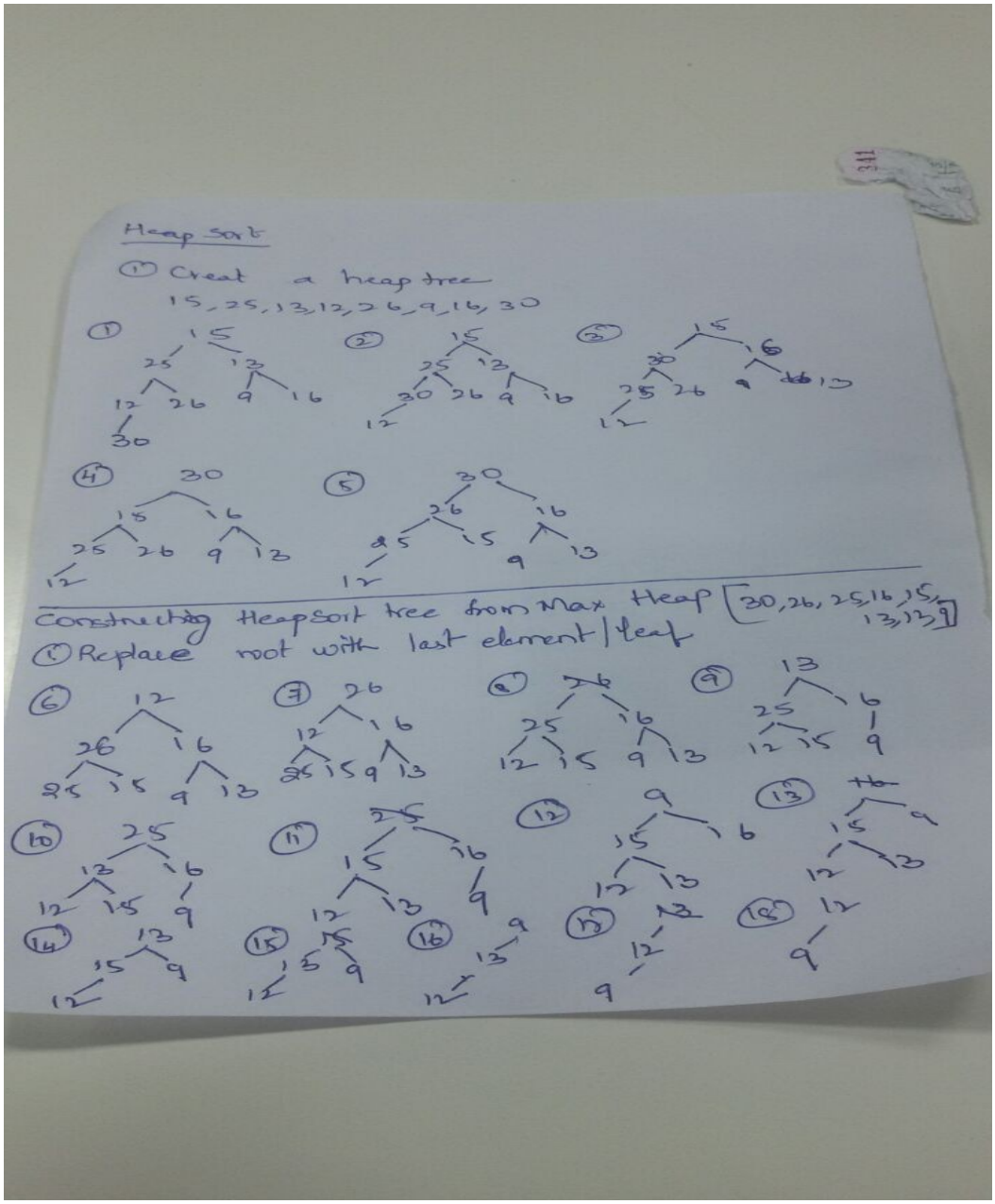
Heap Sort :-
1) Create a max heap tree
2) Replace the root node with the leaf element and add the root element to the array

http://www.sanfoundry.com/java-program-implement-heap-sort/

```
public static void maxheap(int arr[], int i)
    {
        int left = 2*i ;
        int right = 2*i + 1;
        int max = i;
        if (left <= N && arr[left] > arr[i])
            max = left;
        if (right <= N && arr[right] > arr[max])
            max = right;

        if (max != i)
        {
            swap(arr, i, max);
            maxheap(arr, max);
        }
}
public static void heapify(int arr[])
    {
        N = arr.length-1;
        for (int i = N/2; i >= 0; i--)
            maxheap(arr, i);
      }
```

Build Heap

```
For(int i=n;i>1;i++)
Swap(a[1],a[n])
N=n-1;
maxheap(arr,1)
```



https://www.youtube.com/watch?v=18_91wFdcW8

Insert a node in a sorted list
http://www.geeksforgeeks.org/given-a-linked-list-which-is-sorted-how-will-you-insert-in-sorted-way/

```
void sortedInsert(struct node** head_ref, struct node* new_node)
{
    struct node* current;
    /* Special case for the head end */
    if (*head_ref == NULL || (*head_ref)->data >= new_node->data)
    {
        new_node->next = *head_ref;
        *head_ref = new_node;
    }
```

```
    else
    {
        /* Locate the node before the point of insertion */
        current = *head_ref;
        while (current->next!=NULL &&
               current->next->data < new_node->data)
        {
            current = current->next;
        }
        new_node->next = current->next;
        current->next = new_node;
```

Reverse a Linked List :- Iteration Time Complexity - O(n) Space O(1)

```
struct Node
{
    int data;
    struct Node* next;
};
struct Node* head;
void Reverse()
{
  struct Node *current,*prev,*next;
  current = head;
  prev = NULL;
  while(current != NULL)
  {
     next = current->next;
     current->next = prev;
     prev = current;
     current = next;
  }
  head = prev;
}
```

Update the value of prev and current .

Reverse a string using Stack

```
for i -> n
stack.push(a[i]
for i->n
a[i] = stack.top
stack.pop
```

Reverse recursively

```
if(node->next == null) {
 return head = node;
}
 reverse(node->next);
curr = node->next;
curr->next = node;
node->next = null;
```

Reverse a Linked List Using stack

```
void Reverse() {
    if(head == NULL) return;
    stack<struct Node*> S;
    Node* temp = head;
    while(temp != NULL) {
        S.push(temp);
        temp= temp->next;
    }
    temp = S.top(); head = temp;
    S.pop();
    while(!S.empty()){
        temp->next = S.top();
        S.pop();
        temp = temp->next;
    }
    temp->next = NULL;
}
```

mycodeschool.com

Reverse a Double Linked List :-

```
  while(curr != NULL)
       {

           Node *next = curr->next;
           Node *prev = curr->prev;
           curr->prev = curr->next;
           curr->next = prev;
           curr = next;
       }
       head = curr

   }
```

Return the middle element of a linked list in one pass :-

Traverse linked list using two pointers. Move one pointer by one and other pointer by two. When the fast pointer reaches end slow pointer will reach middle of the linked list.

```
if (head!=NULL)
   {
       while (fast_ptr != NULL && fast_ptr->next != NULL)
       {
           fast_ptr = fast_ptr->next->next;
           slow_ptr = slow_ptr->next;
       }
       printf("The middle element is [%d]\n\n", slow_ptr->data);
```

```
    }
}
```

Find Loop in a Linked List

```
A   B
=   =
1   2
2   4
3   6
4   8
5   4
6   6
```

http://stackoverflow.com/questions/10275587/finding-loop-in-a-singly-linked-list

	Worst-Case	Average-Case	Best-Case
Insertion Sort	O(n^2)	O(n^2)	O(n)
Quick Sort	O(n^2)	O(n logn)	O(nlogn)

Shell Sort is same as Insertion sort but go back (i.e hole--) by h-sort

Convex Hull of N points is a smallest perimeter that enclosed the fence .
Smallest convex polygon that encloses all the points

Binary Search : Time Complexity is O(log n)

```
while( ( $low <= $high ) && !$found_key ) {
  $mid = ( $low + $high ) / 2;
  if( $key == $array[$mid] ) {
    $found_key = 1;
    $index = int( $mid );
  } elsif( $key < $array[$mid] ) {
    $high = $mid - 1;
  } else {
    $low = $mid + 1;// BinarySearch(string,mid+1,high,key)
```

Huffman Coding :
It uses Lossless Data Compression

Variable Length Encoding – Frequently appearing characters are given shorter code and less frequent are given larger code
Prefix free coding

Each ASCII character takes 8 bits to represent so to reduce that size we use Huffman Encoding

1) Select the words with lowest frequency first and form a binary tree
2) Mark the elements on the left side as 0 and right as 1
3) No code should be prefix of other code

Brute Force try all possible subsets

Priority Queue is a regular queue or stack but with priority assigned to it . Element with higher priority is served first than element with lower priority. If they are same priority then served based on the order in the queue

Comparable Interface uses Natural Order
Comparator Interface uses alternate Order

1.1.1 Problem :-

Get the maximum value without exceeding the total weight W

KnapSack(v, w, n, W)
{
  for $(w = 0$ to $W)$ $V[0, w] = 0$;
  for $(i = 1$ to $n)$
    for $(w = 0$ to $W)$
      if $(w[i] \leq w)$
        $V[i, w] = \max\{V[i-1, w], v[i] + V[i-1, w - w[i]]\}$;
      else
        $V[i, w] = V[i-1, w]$;
  return $V[n, W]$;
}

Time complexity: Clearly, $O(nW)$.

```
for (i = 0; i <= n; i++)
   {
       for (w = 0; w <= W; w++)
       {
           if (i==0 || w==0)
               K[i][w] = 0;
           else if (wt[i-1] <= w)
                 K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w]);
           else
                 K[i][w] = K[i-1][w];
       }
   }

   return K[n][W];
```

https://www.youtube.com/watch?v=1fDAOvgK11s
http://www.geeksforgeeks.org/dynamic-programming-set-10-0-1-knapsack-problem/
https://dzone.com/articles/knapsack-problem

N Queens

```
if (q[i] == q[n])            return false;   // same column
if ((q[i] - q[n]) == (n - i)) return false;   // same major diagonal
if ((q[n] - q[i]) == (n - i)) return false;   // same minor diagonal
```

http://introcs.cs.princeton.edu/java/23recursion/Queens.java

```
if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))            //
Backtracking
```

```
 if(a[i]==j)
    printf("Q\t");
   else
    printf("*\t");
```

http://www.c-program-example.com/2011/10/c-program-to-solve-n-queens-problem.html

Time Complexity of Search in Binary Tree is O(logn) and also its same for insertion and deletion

	Array (unsorted)	Linked List	Array (sorted)	BST (balanced)
Search(x)	O(n)	O(n)	O(log n)	O(log n)
Insert(x)	O(1)	O(1)	O(n)	O(log n)
Remove(x)	O(n)	O(n)	O(n)	O(log n)

n- n/2 – n/4 …. 1 search goes on till it reaches 1 node

Create a Binary Search Tree :-

https://gist.github.com/mycodeschool/44e1a9183ab0e931f729

Insert in a Binary Search Tree

```
if(root = NULL)
 root-> data = data
 root-> left = root->right = NULL
} else if(data <= root->data) {
 Tmp = malloc;
 root->left = tmp
 tmp->left=tmp->right= NULL
} else (data > root->data) {
 Tmp = malloc;
 root->right = tmp;
 tmp->left = tmp->right = data
```

Finding min or max in a binary search tree

```
While(root->left!=Null){ // Because minimum value will be to the left side

  root = root->left
}
```

```
return root->data

//For max root->right!=NULL
```

Find height of a binary tree

```
int FindHeight(struct Node *root) {
    if(root == NULL)
        return -1;
    return max(FindHeight(root->left),FindHeight(root->right)) +1;
}
```

The process of visiting the node exactly once in any order is called Tree Traversal

Breadth First Traversal is called Level Order Traversals.

Root->Left-> Right – Pre order
Left->Root->Right – In Order
Left->Right->Root – Post Order

Level Order Traversal :-

```
 Q.push(root)
while(!q.notEmpty()) {
  node *current = Q.front();
  Q.pop;
  if(current->left!=NULL) Q.push(current->left)
  if(current->right!=NULL)Q.push(current->right)
}
```

Level Order Traversal at a Level:-

```
void printlevel(struct treenode *node, int curlevel, int level)
{
if(curlevel==level)
{
printf("%d\n",node->data);
}
else
{
printlevel(node->left,curlevel+1,,level);
printlevel(node->right,curlevel+1,level);
}
}
```

http://algorithmsandme.blogspot.in/2014/02/tree-traversal-without-recursion-using.html#.VD5Q5vmSwTA

```
void printPostorder(struct node* node)
{
     if (node == NULL)
        return;

     // first recur on left subtree
```

```
     printPostorder(node->left);

     // then recur on right subtree
     printPostorder(node->right);

     // now deal with the node
     printf("%d ", node->data);
}

/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct node* node)
{

     if (node == NULL)
          return;
     /* first recur on left child */
     printInorder(node->left);

     /* then print the data of node */
     printf("%d ", node->data);

     /* now recur on right child */
     printInorder(node->right);
}
/* Given a binary tree, print its nodes in inorder*/
void printPreorder(struct node* node)
{
     if (node == NULL)
          return;
     /* first print data of node */
     printf("%d ", node->data);
     /* then recur on left sutree */
     printPreorder(node->left);
     /* now recur on right subtree */
     printPreorder(node->right);
}
```

Iterative PreOrder

```
 if (root == NULL)
       return;

    // Create an empty stack and push root to it
    stack<node *> nodeStack;
    nodeStack.push(root);

/* Note that right child is pushed first so that left is processed first */
    while (nodeStack.empty() == false)
    {
        // Pop the top item from stack and print it
        struct node *node = nodeStack.top();
        printf ("%d ", node->data);
        nodeStack.pop();

        // Push right and left children of the popped node to stack
        if (node->right)
            nodeStack.push(node->right);
        if (node->left)
            nodeStack.push(node->left);
    }
```

```
}
```

Iterative PostOrder using two stacks

```
Stack.push(root)
while (!isEmpty(s1))
    {
        // Pop an item from s1 and push it to s2
        node = stack.top();
         stack.pop();
        stack2.push( node);

        // Push left and right children of removed item to s1
        if (node->left)
            push(s1, node->left);
        if (node->right)
            push(s1, node->right);
    }

    // Print all elements of second stack
    while (!isEmpty(s2))
    {
        node = pop(s2);
        printf("%d ", node->data);
    }
}
```

```
InOrder Stack
void inorder_stack(Node * root){
        stack ms;
        ms.top = -1;
        Node * temp  = root;
        while(!is_empty(ms) || temp){
                if(temp!=NULL){
                        stack.push( temp);
                        temp = temp->left;
                }
                else {
                        temp = stack.top();
                       stack.pop();
                        printf("%d  ", temp->value);
                        temp = temp->right;
                }
        }
}
```

Merge Two Linked Lists Using Recursion

```
Node MergeLists(Node list1, Node list2) {
  if (list1 == null) return list2;
  if (list2 == null) return list1;

  if (list1.data < list2.data) {
    list1.next = MergeLists(list1.next, list2);
    return list1;
  } else {
    list2.next = MergeLists(list2.next, list1);
    return list2;
  }
}
```

http://algorithmsandme.blogspot.in/2013/10/linked-list-merge-two-sorted-linked.html#.VDJgtfmSwTA

```
public class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

        ListNode p1 = l1;
        ListNode p2 = l2;

        ListNode fakeHead = new ListNode(0);
        ListNode p = fakeHead;

        while(p1 != null && p2 != null){
          if(p1.val <= p2.val){
              p.next = p1;
              p1 = p1.next;
          }else{
              p.next = p2;
              p2 = p2.next;
          }

          p = p.next;
        }

        if(p1 != null)
            p.next = p1;
        if(p2 != null)
            p.next = p2;

        return fakeHead.next;
    }
}
```

To find k^{th} largest element in a linked list

1) Use bubble sort to run upto k times
2) Print the last k elements : (Onk) time complexity

http://www.crazyforcode.com/kth-largest-smallest-element-array/

Append the last n nodes of a linked list to the beginning of the list

eg: 1->2->3->4->5->6

if n=2

5->6->1->2->3->4

```
node *prepend(node * root, int k)
{
        node *prev, *curr;
        curr = root;
        for (int i = 0; i < k; i++) {
                curr = curr->next;
                if (curr == NULL)

                        return NULL;
        }
        prev = root;
        while (curr->next != NULL) {
                curr = curr->next;
                prev = prev->next;
        }
        curr->next = root;
        root = prev->next;
        prev->next = NULL;
        return root;
}
```

Print Leave Nodes of a Binary Tree – Leaf

```
 if(root == NULL)
        return;
       if(root->left == NULL && root->right==NULL)
          System.out.println(root->data);
       printLeafNodes(root->left);
       printLeafNodes(root->right);
}
```

Boundary Traversal
http://www.geeksforgeeks.org/boundary-traversal-of-binary-tree/

1) Left nodes top to bottom
2) Leaf nodes left to right
3) Right nodes bottom to to

of Binary Tree IN Place

```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;
    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```

Mirror Of Binary Tree by creating a new tree

http://www.crazyforcode.com/mirror-image-binary-tree/

```
  Temp = malloc
  temp->data  = root->data
  temp->left  = mirror(root->right)
  temp->right = mirror(root->left)
  return temp
```

Foldable Binary Tree

```
bool isFoldable(struct node *root)
{
  bool res;

  /* base case */
  if(root == NULL)
    return true;

  /* convert left subtree to its mirror */
  mirror(root->left);

  /* Compare the structures of the right subtree and mirrored
    left subtree */
  res = isStructSame(root->left, root->right);

  /* Get the originial tree back */
  mirror(root->left);

  return res;
}


bool isStructSame(struct node *a, struct node *b)
{
  if (a == NULL && b == NULL)
  {  return true; }
  if ( a != NULL && b != NULL && a.val == b.val &&
       isStructSame(a->left, b->left) &&
       isStructSame(a->right, b->right)
     )
  {  return true; }

  return false;
}
```

http://www.geeksforgeeks.org/foldable-binary-trees/

```
/*
 Given two trees, return true if they are
 structurally identical.
*/
int sameTree(struct node* a, struct node* b) {
  // 1. both empty -> true
  if (a==NULL && b==NULL) return(true);


  // 2. both non-empty -> compare them
  else if (a!=NULL && b!=NULL) {
    return(
```

```
      a->data == b->data &&
      sameTree(a->left, b->left) &&
      sameTree(a->right, b->right)
    );
  }
  // 3. one empty, one not -> false else return(false);
```

Check if Binary Tree is Binary Search Tree or Not

_____

```
int isBST2(struct node* node) {
  return(isBSTUtil(node, INT_MIN, INT_MAX));
}

/*
 Returns true if the given tree is a BST and its
 values are >= min and <= max.
*/
int isBSTUtil(struct node* node, int min, int max) {
  if (node==NULL) return(true);

  // false if this node violates the min/max constraint
  if (node->data<min || node->data>max) return(false);

returns  isBSTUtil(node->left, min, node->data) &&
           isBSTUtil(node->right, node->data, max)
         );
       }

TC - O(n)

if(node->data > min && node->data < max && isbst(node->left,min,node->data) &&
isBST(node->right,node->data,max))

 return true
```

Write a function to find 5th element from a singly linked List from the end(not from the head) in one pass.

Take 2 pointers- move the first one past 5 nodes from the head and then start the second one at the head. Keep advancing both the pointers one step at a time until the first pointer reaches the last node when which the second will be pointing to the 5th node from the last.

Code :- Same as Append

```
node *prepend(node * root)
{
        node *prev, *curr;
        curr = root;
        for (int i = 0; i < 5; i++) {
                curr = curr->next;// Pointer will reach 5th element
                if (curr == NULL)
                        return NULL;
        }
        prev = root;
        while (curr->next != NULL) {
                curr = curr->next;
                prev = prev->next;
        }
         prev->data // data at 5th element
        return root;
}
```

Consider sorted singly linked list having following nodes

10->30->50->70->NULL

You are given pointer to node 50 and a new node having value 40. Can you inserte node 40 correctly in the list maintaining the ascending order?

```
pNew->next = p->next;
p->next = pNew;
p->val = 40; // pnew->val
pNew->val = 50; //p->val
```

Remove Duplicates in Linked List

Sorted:-

http://www.geeksforgeeks.org/remove-duplicates-from-a-sorted-linked-list/

```
while(curr->next!=NULL){
       if(curr->next->data == curr->data) {
              temp = curr->next->next;
              curr->next = temp;
              free(temp)
       } else {
              curr = curr->next
       }
}
```

Tc :o(n)

Reverse Every K elements in a Linked List

Reverse a linked list for first k terms

http://www.geeksforgeeks.org/reverse-a-list-in-groups-of-given-size/

```
struct node *reverse (struct node *head, int k)
{
    struct node* current = head;
    struct node* next = NULL;
    struct node* prev = NULL;
    int count = 0;

    /*reverse first k nodes of the linked list */
    while (current != NULL && count < k) // for(int i=0; i<k; i++)
    {
       next  = current->next;
       current->next = prev;
       prev = current;
       current = next;
       count++;
    }

    /* next is now a pointer to (k+1)th node
       Recursively call for the list starting from current.
       And make rest of the list as next of first node */
    if(next !=  NULL)
    {  head->next = reverse(next, k); }

    /* prev is new head of the input list */
    return prev;

}
```

Reverse every alternate K nodes in Linked List

```
  if(head != NULL)
      head->next = current;

    /* 3) We do not want to reverse next k nodes. So move the current
        pointer to skip next k nodes */
    count = 0;
    while(count < k-1 && current != NULL )
    {
      current = current->next;
      count++;
    }
if(current !=  NULL)
       current->next = kAltReverse(current->next, k);
```

http://www.dsalgo.com/2013/02/print-nodes-of-given-level.html

```
private static void printLevelofDepth(Node a, int depth)
 {
  if (a == null)
   return;
```

```
  if (depth == 1)
  {
   System.out.println(a.value);
   return;
  }
  printLevelofDepth(a.left, depth - 1);
  printLevelofDepth(a.right, depth - 1);
 }
```

Spiral Order Print of Binary Tree

```
void printSpiral(struct node *root)

{
    if (root == NULL)  return;   // NULL check

    // Create two stacks to store alternate levels
    stack<struct node*> s1;  // For levels to be printed from right to left
    stack<struct node*> s2;  // For levels to be printed from left to right

    // Push first level to first stack 's1'
    s1.push(root);

    // Keep ptinting while any of the stacks has some nodes
    while (!s1.empty() || !s2.empty())
    {
        // Print nodes of current level from s1 and push nodes of
        // next level to s2
        while (!s1.empty())
        {
            struct node *temp = s1.top();
            s1.pop();
            cout << temp->data << " ";

            // Note that is right is pushed before left
            if (temp->right)
                s2.push(temp->right);
            if (temp->left)
                s2.push(temp->left);
        }
// Print nodes of current level from s2 and push nodes of
        // next level to s1
        while (!s2.empty() {
            struct node *temp = s2.top();
            s2.pop();
            cout << temp->data << " ";
            // Note that is left is pushed before right
            if (temp->left)
                s1.push(temp->left);
            if (temp->right)
                s1.push(temp->right);
        }
    }
```

Merge Two Linked Lists ALternatively

```
void merge(struct node *p, struct node **q)
{
      struct node *p_curr = p, *q_curr = *q;
```

```
     struct node *p_next, *q_next;

     // While therre are avialable positions in p
     while (p_curr != NULL && q_curr != NULL)
     {
         // Save next pointers
         p_next = p_curr->next;
         q_next = q_curr->next;

         // Make q_curr as next of p_curr
         q_curr->next = p_next;  // Change next pointer of q_curr
         p_curr->next = q_curr;  // Change next pointer of p_curr

         // Update current pointers for next iteration
         p_curr = p_next;
         q_curr = q_next;
    }

    *q = q_curr; // Update head pointer of second list
}
```

Flatten a Linked List

```
void flattenList(struct node *head)
{
    /*Base case*/
    if (head == NULL)
       return;

    struct node *tmp;

    /* Find tail node of first level linked list */
    struct node *tail = head;
    while (tail->next != NULL)
        tail = tail->next;

    // One by one traverse through all nodes of first level
    // linked list till we reach the tail node
    struct node *cur = head;
    while (cur != tail)
    {
        // If current node has a child
        If (cur->child)
        {
            // then append the child at the end of current list
            tail->next = cur->child;

            // and update the tail to new last node
            tmp = cur->child;
```

```
            while (tmp->next != null)
                tmp = tmp->next;
            tail = tmp;
        }

        // Change current node
        cur = cur->next;
    }
```

The pseudo-code for the **merge sort** is really simple.

```
# C = output [length = N]
# A 1st sorted half [N/2]
# B 2nd sorted half [N/2]
i = j = 1
for k = 1 to n
    if A[i] < B[j]
        C[k] = A[i]
        i++
    else
        C[k] = B[j]
        j++
```

It is easy to see that in every loop you will have 4 operations: **k++**, **i++** or **j++**, the **if statement** and the **attribution C = A|B**. So you will have less or equal to 4N + 2 operations giving a *O(N)* complexity. For the sake of the proof 4N + 2 will be treated as 6N, since is true for N = 1 (**4N +2 <= 6N**).

So assume you have an input with **N** elements and assume **N** is a power of 2. At every level you have two times more subproblems with an input with half elements from the previous input. This means that at the the level **j** = 0, 1, 2, ..., lgN there will be **2^j** subproblems with an input of length **N / 2^j**. The number of operations at each level **j** will be less or equal to

> 2^j * 6(N / 2^j) = 6N

Observe that it doens't matter the level you will always have less or equal 6N operations.

Since there are lgN + 1 levels, the complexity will be

> O(6N * (lgN + 1)) = O(6N*lgN + 6N) = **O(n lgN)**

3-way-Merge Sort : Suppose that instead of dividing in half at each step of Merge Sort, you divide into thirds, sort each third, and finally combine all of them using a three-way merge subroutine. What is the overall asymptotic running time of this algorithm? (Hint: Note that the merge step can still be implemented in $O(n)$ time.)

Your Answer		Score	Explanation
◉ $n\log(n)$	✔	1.00	That's correct! There is still a logarithmic number of levels, and the overall amount of work at each level is still linear.
Total		1.00 / 1.00	

http://www.cs.princeton.edu/courses/archive/spr03/cs226/lectures/analysis.4up.pdf

Mergesort analysis: Comparison count

Def. T(N) ≡ number of comparisons to mergesort an input of size N

= T(N/2) + T(N/2) + N
(left half, right half, merge)

Mergesort recurrence T(N) = 2 T(N/2) + N
for N > 1, with T(1) = 0

- not quite right for odd N
- same recurrence holds for many algorithms
- same number of comparisons for any input of size N.

lg N ≡ $\log_2$ N

Solution of Mergesort recurrence T(N) ~ N lg N

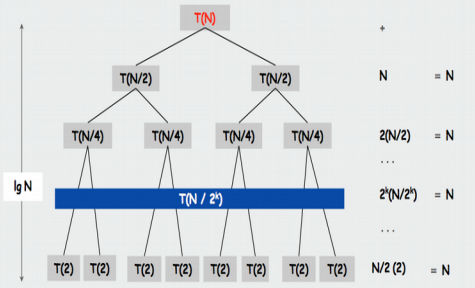
- true for all N, as long as integer approx of N/2 is within a constant
- easy to prove when N is a power of 2.

can then use induction for general N (see COS 341)

9

Mergesort recurrence: Proof 1 (by recursion tree)

T(N) = 2 T(N/2) + N
for N > 1, with T(1) = 0
(assume that N is a power of 2)



T(N) = N lg N

10

Mergesort recurrence: Proof 2 (by telescoping)

T(N) = 2 T(N/2) + N
for N > 1, with T(1) = 0
(assume that N is a power of 2)

Pf.

T(N) = 2 T(N/2) + N		given
T(N)/N = 2 T(N/2)/N + 1		divide both sides by N
= T(N/2)/(N/2) + 1		algebra
= T(N/4)/(N/4) + 1 + 1		telescope (apply to first term)
= T(N/8)/(N/8) + 1 + 1 + 1		telescope again
...		
= T(N/N)/(N/N) + 1 + 1 + ... + 1		stop telescoping, T(1) = 0
= lg N		

T(N) = N lg N

11

Mergesort recurrence: Proof 3 (by induction)

T(N) = 2 T(N/2) + N
for N > 1, with T(1) = 0
(assume that N is a power of 2)

Claim. If T(N) satisfies this recurrence, then T(N) = N lg N.

Pf. [by induction on N]

- Base case: N = 1.
- Inductive hypothesis: T(N) = N lg N
- Goal: show that T(2N) = 2N lg (2N).

T(2N) = 2 T(N) + 2N		given
= 2 N lg N + 2 N		inductive hypothesis
= 2 N (lg (2N) - 1) + 2N		algebra
= 2 N lg (2N)		QED

Ex. (for COS 341). Extend to show that T(N) ~ N lg N for general N

12

http://www.careercup.com/question?id=22355666

x

The Master Theorem

Let $a \geq 1, b > 1, d \geq 0$, and $T(n)$ be a monotonically increasing function of the form:

$$T(n) = aT(n/b) + \Theta(n^d)$$

Then:

$$\begin{array}{ll} T(n) \text{ is } \Theta(n^d) & \text{if } a < b^d \\ T(n) \text{ is } \Theta(n^d \log n) & \text{if } a = b^d \\ T(n) \text{ is } \Theta(n^{\log_b a}) & \text{if } a > b^d \end{array}$$

- a is the number of subproblems
- n/b is the size of each subproblem (if n/b is a fraction, b will be a whole number)
- n^d is the work done to prepare the subproblems and assemble the sub-results

Applying the Master Theorem

$$T(n) = aT(n/b) + \Theta(n^d)$$

$$\begin{array}{ll} T(n) \text{ is } \Theta(n^d) & \text{if } a < b^d \\ T(n) \text{ is } \Theta(n^d \log n) & \text{if } a = b^d \\ T(n) \text{ is } \Theta(n^{\log_b a}) & \text{if } a > b^d \end{array}$$

- Merge sort's recurrence relation is T(n) = 2T(n/2) + O(n)
  - $a = 2, b = 2, d = 1 \rightarrow a = b^d$
  - T(n) is $\Theta(n^d \log n) = \Theta(n^1 \log n) = \Theta(n \log n)$

https://www.cs.umd.edu/class/fall2009/cmsc451/lectures/Lec08-inversions.pdf

Time complexity of n matrix multiplication is O(n^3) .

```
C[i][j] = c[i][j]+a[i][k]*b[k][j]
```

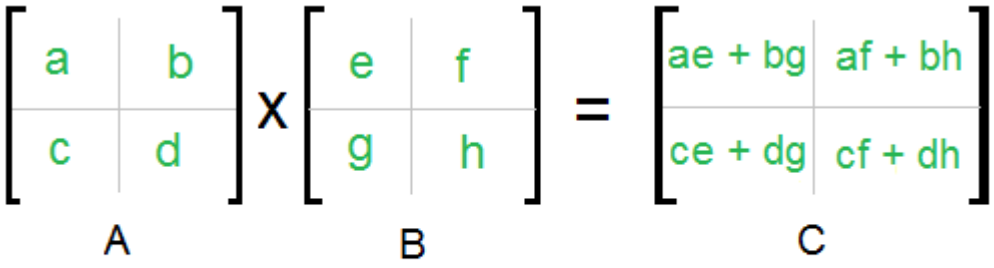
http://www.geeksforgeeks.org/strassens-matrix-multiplication/

Divide and Conquer

Following is simple Divide and Conquer method to multiply two square matrices.

1) Divide matrices A and B in 4 sub-matrices of size N/2 x N/2 as shown in the below diagram.

2) Calculate following values recursively. ae + bg, af + bh, ce + dg and cf + dh.



A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2

In the above method, we do 8 multiplications for matrices of size N/2 x N/2 and 4 additions. Addition of two matrices takes $O(N^2)$ time. So the time complexity can be written as

$T(N) = 8T(N/2) + O(N^2)$ => a > b^d = O(n^logb a)

From Master's Theorem, time complexity of above method is $O(N^3)$ which is unfortunately same as the above naive method.

Strassen's multiplication has only 7 multiplications and 4 additions.
7T(N/2)+)(n^2)

http://prismoskills.appspot.com/lessons/Dynamic_Programming/Chapter_11_-_Egg_dropping_puzzle.jsp

So, we can find max droppings on each floor and find minimum among them to arrive programatically at our answer.

http://datagenetics.com/blog/july22012/index.html

Print BST in given range

```
/* The functions prints all the keys which in the given range
```

```
[k1..k2].
    The function assumes than k1 < k2 */
void Print(struct node *root, int k1, int k2)
{
   /* base case */
   if ( NULL == root )
      return;

   /* Since the desired o/p is sorted, recurse for left subtree
first
      If root->data is greater than k1, then only we can get o/p
keys
      in left subtree */
   if ( k1 < root->data )
     Print(root->left, k1, k2);

   /* if root's data lies in range, then prints root's data */
   if ( k1 <= root->data && k2 >= root->data )
     printf("%d ", root->data );

  /* If root->data is smaller than k2, then only we can get o/p keys
      in right subtree */
   if ( k2 > root->data )
     Print(root->right, k1, k2);
}
```

given an array with elements check if just by exchanging two elements of the array we get a sorted array. time restriction: O(NlogN) space restriction: 2N

1) Copy the original array

2) Sort the copy

3) loop through the arrays and check how many corresponding elements don't match

O(NlogN) - space 2N

Assuming you have a binary tree which stores integers, count the number of nodes whose value is lower than the value of its upper nodes.

In a binary tree there can be any value on any node, so we need to keep track of the Min. value found so far for that depth branch and propagate it until you reach the leafs, be aware not to count duplicates in the way back, a HashSet would help to avoid that, and the final HashSet will hold all Nodes which are lesser than all their possible parents.

```
public static int countNumLesserNodes(Node<Integer> root) {
      if(root == null) {
            return 0;
      }
      Set<Node<Integer>> set = new HashSet<>();
      countNumLesserNodes(root.left, root.data, set);
      countNumLesserNodes(root.right, root.data, set);
      return set.size();
}

public static void countNumLesserNodes(Node<Integer> root, int pVal,
Set<Node<Integer>> set) {
      if(root == null) {
            return;
      }
      if(root.data < pVal) {
            set.add(root);
            pVal = root.data;
      }
      countNumLesserNodes(root.left, pVal, set);
      countNumLesserNodes(root.right, pVal, set);
}
```

Find the longest running positive sequence in an array

```
int[] a = { 1, 2, 3, 4, 0, 19, 1, 1, 2, 2, 3, 3, 2 };
int count = 1, max = 1;
for (int i = 1; i < a.length; i++) {
    if (a[i] >= a[i - 1]) {
        count++;
    } else {
        if (count > max) {
            max = count;
        }
        count = 1;
    }
```

```
}
System.out.println(max);
```

Here, `count` is the number of contiguous and sorted elements. We increment it while we're on a continuous/increasing streak. Once this streak breaks (`else` clause), we check if `count` is greater than `max`, our variable for storing the maximum count: if it is, we set `max` to `count`. We then set `count` back to 1, since in the `else` clause we know our streak has ended and we'll need to start over.

Find nth most frequent number in array

```
public static int getNthMostFrequent(int[] a) {
             HashMap<Integer, Integer> map = new HashMap<Integer,
Integer>();
             int maxCount=0, freqValue = -1;
             for(int i=0; i < a.length; i++) {
                if(map.get(a[i]) == null) { //Insert new.
                   map.put(a[i], 1);
                }else { //Already exists. Just increment the count.
                   int count = map.get(a[i]);
                   count++;
                   map.put(a[i], count);
                  if(count > maxCount) {
                      maxCount = count;
                      freqValue = a[i];
                   }

                }

             }
                   //incase all numbers are unique.
              if(freqValue == -1 && a.length>;0)
                 return a[0];
              return freqValue;
            }
```

1. add to map with number as the key and its frequency as the value O(n)
2. get the values from the map as list and do kth select algorithm O(log n);

Given an unsorted array of integers find a minimum number which is not present in array. e.g -1 ,4, 5, -23,24 is array then answer should be -22.

```
    HashSet<Integer> set = new HashSet<Integer>();
    int minVal = Integer.MAX_VALUE;
    for(int i : arr){
        if(i < minVal){
            minVal = i;
        }
        set.add(i);
    }
    //search the HashSet for the first value counting up from the min that is NOT
in the set
```

```
    while(set.contains(minVal)){
        minVal++;
    }
    return minVal;
```

O(n) memory and O(n) runtime

http://www.careercup.com/question?id=5758677862580224

Given an array of random integers and a sum value, find two numbers from the array that sum up to the given sum.

eg. array = {2,5,3,7,9,8}; sum = 11 output -> 2,9

```
for(i = 0; i < arr_size; i++)
  {
    temp = sum - arr[i];
    if(temp >= 0 && binMap[temp] == 1)
    {
      printf("Pair with given sum %d is (%d, %d) \n", sum, arr[i], temp);
    }
    binMap[arr[i]] = 1;
  }
}
```

You are given an array, you have to replace each element of the array with product of the rest element.

Example: {1,2,3}==> {6,3,2}

```
      for(int i = 0; i < arr.length; i++)
            {
                  if(arr[i] == 0)
                        continue;
                  product *= arr[i];
            }
            for (int i = 0; i < arr.length; i++)
            {
                  if(arr[i] == 0)
                        continue;
                  arr[i] = product / arr[i]; // O(n*n) TC

O(n) complexity:-

int a[N] // This is the input
int products_below[N];
p=1;
for(int i=0;i<N;++i) {
  products_below[i]=p;
  p*=a[i];
}

int products_above[N];
p=1;
for(int i=N-1;i>=0;--i) {
  products_above[i]=p;
  p*=a[i];
}

int products[N]; // This is the result
for(int i=0;i<N;++i) {
```

```
  products[i]=products_below[i]*products_above[i];
}
int a[N] // This is the input
int products[N];

// Get the products below the current index
temp=1;
for(int i=0;i<N;++i) {
  products[i]=temp;
  temp*=a[i];
}
**-
// Get the products above the curent index
temp=1;
for(int i=N-1;i>=0;--i) {
  products[i]*=temp;
  temp*=a[i];
}
```

Algorithm:

1) Construct a temporary array left[] such that left[i] contains product of all elements on left of arr[i] excluding arr[i].

2) Construct another temporary array right[] such that right[i] contains product of all elements on on right of arr[i] excluding arr[i].

3) To get prod[], multiply left[] and right[].

Children-Sum Property

http://www.geeksforgeeks.org/check-for-children-sum-property-in-a-binary-tree/

```
int isSumProperty(struct node* node)
{
  /* left_data is left child data and right_data is for right child data*/
  int left_data = 0,  right_data = 0;

  /* If node is NULL or it's a leaf node then
     return true */
  if(node == NULL ||
     (node->left == NULL && node->right == NULL))
    return 1;
  else
  {
    /* If left child is not present then 0 is used
       as data of left child */
    if(node->left != NULL)
      left_data = node->left->data;

    /* If right child is not present then 0 is used
      as data of right child */
    if(node->right != NULL)
      right_data = node->right->data;
```

```
    /* if the node and both of its children satisfy the
       property return 1 else 0*/
    if((node->data == left_data + right_data)&&
        isSumProperty(node->left) &&
        isSumProperty(node->right))
      return 1;
    else
      return 0;
  }
}
```

Find all palindromes in a given string. Single letters are also Considered as palindromes.

```
	public static void getSubStringPalindrome(String str)	{

		for(int i = 0; i < str.length() ; i++) {

			for(int j = i ; j < str.length(); j++ ){

				String s = str.substring(i , j);

				if(s.length() <= 1) {
					continue;
				}
				if(isPalindrome(s)) {

					System.out.println(s);
				}
			}
		}

	}

	public static boolean isPalindrome(String s ) {

		//System.out.println("IsPalindrome:" + s + "?");
		StringBuilder sb = new StringBuilder(s);
		String reverse = sb.reverse().toString();
		//System.out.println("IsPalindrome Reverse:" + reverse + "?");
		return reverse.equals(s)
```

O(n^3) complexity

Can also use Manacher's algorithm

Design a system for finding the costliest element always whenever we pick up an element from a box.(concept of Max Heap)

DFS using Stack

```
void dfs(GraphNode node) {
    // sanity check
    if (node == null) {
        return;
    }
```

```
    // use a hash set to mark visited nodes
    Set<GraphNode> set = new HashSet<GraphNode>();

    // use a stack to help depth-first traversal
    Stack<GraphNode> stack = new Stack<GraphNode>();
    stack.push(node);

    while (!stack.isEmpty()) {
        GraphNode curr = stack.pop();

        // current node has not been visited yet
        if (!set.contains(curr)) {
            // visit the node
            // ...

            // mark it as visited
            set.add(curr);
        }

        for (int i = 0; i < curr.neighbors.size(); i++) {
            GraphNode neighbor = curr.neighbors.get(i);

            // this neighbor has not been visited yet
            if (!set.contains(neighbor)) {
                stack.push(neighbor);
            }
        }
    }
}
```

in BFS we use a queue and for every node we add all the adjacent nodes into the visited list .

https://www.youtube.com/watch?v=zLZhSSXAwxI

NonRecursive Method to calculate Height of Tree

```
     Queue.push(root);
     Queue.push(Null);  // null is marker that one level ends
     height=0;
     While( ! Queue.empty() )
      {
       node = Queue.pop();
       if(node== null)
       {
         ++height;
       if(! Queue.empty())
       Queue.push(null);
        }
        else
```

```
     {
      if(node.left)
      Queue.push(node.left);

      if(node.right)
      Queue.push(node.right);
      }
     }
 return height;
```

Implement Queue using Stacks

Use two stacks . s1 and s2. For enqueue directly push into s1. For dequeue , if s1 is not empty and s2 is empty pop elements from s1 into s2 and pop the top element in s2
http://www.geeksforgeeks.org/queue-using-stacks/

Implement Stack using Queues

```
push(s,  x)
  1) Enqueue x to q1 (assuming size of q1 is unlimited).

pop(s)
  1) One by one dequeue everything except the last element
from q1 and enqueue to q2.
       2) Dequeue the last item of q1, the dequeued item is
     result, store it.
  3) Swap the names of q1 and q2
  4) Return the item stored in step 2.
// Swapping of names is done to avoid one more movement of all
elements
// from q2 to q1.
```

Given ten million numbers, each having 11 bits, find the most time efficient way to sort the numbers.

Since we know we're sorting fixed-size integers, radix sort is much more efficient than any comparison-based sort

LSD radix sorts typically use the following sorting order: short keys come before longer keys, and keys of the same length are sorted lexicographically. This coincides with the normal order of integer representations, such as the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. MSD radix sorts use lexicographic order, which is suitable for sorting strings, such as words, or fixed-length integer representations. A sequence such as "b, c, d, e, f, g, h, i, j, ba" would be lexicographically sorted as "b, ba, c, d, e, f, g, h, i, j". If lexicographic ordering is used to sort variable-length integer representations, then the representations of the numbers from 1 to 10 would be output as 1, 10, 2, 3, 4, 5, 6, 7, 8, 9, as if the shorter keys were left-justified and padded on the right with

blank characters to make the shorter keys as long as the longest key for the purpose of determining sorted order.

Find Level which is having maximum sum in Binary tree

Same like BFS using queue.

```
            q.push(root)
            q.push(null) // Null is pushed to mark end of level
            maxSum=currSum=maxLevel=level=0
            while (!q.isEmpty) {
             temp = q.pop();
             if (temp == null) {
                  if(currSum > maxSum) {
                        maxSum=currSum;
                        maxLevel=level;
                  }
                  currSum = 0;
                  if (!q.isEmpty) {
                        q.push(null);
                        level++
                  }
             } else {
                  currSum += temp->data;
                  if(temp->left) q.push(temp->left);
                  if(temp->right) q.push(temp->right);
             }
```

Find max element in a binary tree

Problems on Binary Trees

Problem-1 Give an algorithm for finding maximum element in binary tree.

Solution: One simple way of solving this problem is: find the maximum element in left subtree, find maximum element in right sub tree, compare them with root data and select the one which is giving the maximum value. This approach can be easily implemented with recursion.

```
int FindMax(struct BinaryTreeNode *root) {
        int root_val, left, right, max = INT_MIN;
        if(root !=NULL) {
                root_val = root→data;
                left = FindMax(root→left);
                right = FindMax(root→right);
                // Find the largest of the three values.
                if(left > right)
                        max = left;
                else    max = right;
                if(root_val > max)
                        max = root_val;
        }
        return max;
}
```

Time Complexity: $O(n)$. Space Complexity: $O(n)$.

Problem-2 Give an algorithm for finding maximum element in binary tree without recursion.

Solution: Using level order traversal: just observe the elements data while deleting.

```
int FindMaxUsingLevelOrder(struct BinaryTreeNode *root){
        struct BinaryTreeNode *temp;
        int max = INT_MIN;
        struct Queue *Q = CreateQueue();
        EnQueue(Q,root);
```



```
        while(!IsEmptyQueue(Q)) {
                temp = DeQueue(Q);
                // largest of the three values
                if(max < temp→data)
                        max =  temp→data;
                if(temp→left)
                        EnQueue (Q, temp→left);
                if(temp→right)
                        EnQueue (Q, temp→right);
        }
        DeleteQueue(Q);
        return max;
}
```

Time Complexity: $O(n)$. Space Complexity: $O(n)$.

For searching an element `if (data==temp->data) return 1;`

Number of leaves in binary tree

`if(!root->left && !root->right) count++;`

Number of full nodes
if(root->left && root->right) count++;
Number of half nodes
If(!root->left && root->right || !root->right && root->left) count++
Sum of all nodes in binary tree
Sum += curr->data;
Return(root->data+Sum(root->left)+Sum(root+right));

If two tree traversals are given and one of them is Inorder then a unique binary tree can be constructed.

BFS in reverse order.

Same BFS using queue , just push the element popped from queue into stack

Delete Element in Binary Tree

For that node delete left and right nodes and delete the root

```
Void deleteBinaryTree(node)
        if (root == null)  { return null;}
        deleteroot(root->left);
        deleteroot(root->righ
free(root);
```

Find max sum from root-> leaf path

```
int maxSumPath (struct node *root)
{
    if (root == NULL)
        return 0;
    if(root->left==NULL && root->right==NULL)
      return root->data;
return max(root->data+maxSumPath(root->left),root->data+maxSumPath(root->right));
```

Max Sum from Root to leaf

```
    public void maxSum(Node root, int sum){
        if(root!=null){
            sum=sum+root.data;
            if(sum>maxSum && root.left==null && root.right==null){
                maxLeaf = root;
                maxSum = sum;

            }
        //  System.out.println("Sum " + sum);
            maxSum(root.left,sum);
            maxSum(root.right,sum);
       }
```

Find existence of a path with a given sum :O(n) TC and sc

Problem-21 Give an algorithm for checking the existence of path with given sum. That means, given a sum check whether there exists a path from root to any of the nodes.

Solution: For this problem, the strategy is: subtract the node value from the sum before calling its children recursively, and check to see if the sum is 0 when we run out of tree.

```
int HasPathSum(struct BinaryTreeNode * root, int sum) {
        // return true if we run out of tree and sum==0
        if(root == NULL) return(sum == 0);
         else {      // otherwise check both subtrees
                  int remainingSum = sum - root→data;
                  if((root→left && root→right)||(!root→left && !root→right))
                        return(HasPathSum(root→left, remainingSum) || HasPathSum(root→right, remainingSum));
                  else if(root→left)
                           return HasPathSum(root→left, remainingSum);
                  else     return HasPathSum(root→right, remainingSum);
         }
```

```
int hasPathSum(struct node* node, int sum) {
  // return true if we run out of tree and sum==0
  if (node == NULL) {
    return(sum == 0);
  }
  else {
  // otherwise check both subtrees
    int subSum = sum - node->data;
    return(hasPathSum(node->left, subSum) ||
           hasPathSum(node->right, subSum));
  }
}
```

For n nodes there are 2^n-n binary trees

Print all paths form root-leaf

```
void printPathsRecur(struct node* node, int path[], int pathLen) {
  if (node==NULL) return;
  // append this node to the path array
  path[pathLen] = node->data;
  pathLen++;
  // it's a leaf, so print the path that led to here
  if (node->left==NULL && node->right==NULL) {

    printArray(path, pathLen);
  }
  else {
  // otherwise try both subtrees
    printPathsRecur(node->left, path, pathLen);
    printPathsRecur(node->right, path, pathLen);
Start with the root. Push root node to stack.
    while(stack not empty)   //i'll take top=top_of_stack
        mark top as visited
        if(top.left_node exists AND not_visited)
            push(top.left_node)
        else if(top.right_node exists AND not_visited)
```

```
            push(top.right_node)
        else      //Leaf node
            print stack
            pop
```

http://cslibrary.stanford.edu/110/BinaryTrees.html

Sort an almost sorted array where only two elements are swapped – O(n)

```
Input:  arr[] = {10, 20, 60, 40, 50, 30}
// 30 and 60 are swapped
Output: arr[] = {10, 20, 30, 40, 50, 60}
```

```
void sortByOneSwap(int arr[], int n)
{
    // Travers the given array from rightmost side
    for (int i = n-1; i > 0; i--)
    {
        // Check if arr[i] is not in order
        if (arr[i] < arr[i-1])
        {
            // Find the other element to be
            // swapped with arr[i]
            int j = i-1;
            while (j>=0 && arr[i] < arr[j])
                j--;

            // Swap the pair
            swap(arr[i], arr[j+1]);
            break;
        }
    }
}
```

You are given a long list of integers, so long that you can not fit the entire list into memory, implement an algorithm to get the max 100 elements in the list.

Use a min heap of size 100. Insert the first 100 elements of the list into the heap. From the 101st element, check if the current element in the list is greater than the min element in the heap. If yes, delete the min element and insert the current element into the min heap. Repeat this until the list is exhausted and in the end, the top 100 elements will be present in the heap.

Implement stack with findMiddle and deleteMiddle() in constant time. Use double linked list

Inorder traversal of bst gives a sorted array . Increment

the count when count== K print the element (kth largest element)
For smallest element , inorder traversal and direcrtly access the element
Using quickselect .
if (k==pivot) return pivot;
If(k < pivot) partition(array, 1, pivot-1)
If(k > pivot) partition(array, pivot+1, n)

Boundary Traversal of Binary Tree

```
void printLeaves(struct node* root)
{
    if ( root )
    {
        printLeaves(root->left);

        // Print it if it is a leaf node
        if ( !(root->left)  &&  !(root->right) )
            printf("%d ", root->data);

        printLeaves(root->right);
    }
}

// A function to print all left boundry nodes, except a leaf node.

// Print the nodes in TOP DOWN manner
void printBoundaryLeft(struct node* root)
{
    if (root)
    {
        if (root->left)
        {
            // to ensure top down order, print the node
            // before calling itself for left subtree
            printf("%d ", root->data);
            printBoundaryLeft(root->left);
        }
        else if( root->right )
        {
            printf("%d ", root->data);
            printBoundaryLeft(root->right);
        }
        // do nothing if it is a leaf node, this way we avoid
        // duplicates in output
    }

// A function to print all right boundry nodes, except a leaf node
// Print the nodes in BOTTOM UP manner
void printBoundaryRight(struct node* root)
{
    if (root)
    {
```

```
        if ( root->right )
        {
            // to ensure bottom up order, first call for right
            //  subtree, then print this node
            printBoundaryRight(root->right);
            printf("%d ", root->data);
        }
        else if ( root->left )
        {
            printBoundaryRight(root->left);
            printf("%d ", root->data);
        }
       // do nothing if it is a leaf node, this way we avoid
       // duplicates in output
    }
}
// A function to do boundary traversal of a given binary tree
void printBoundary (struct node* root)
{
    if (root)
    {
        printf("%d ",root->data);

        // Print the left boundary in top-down manner.
        printBoundaryLeft(root->left);

        // Print all leaf nodes
        printLeaves(root->left);
        printLeaves(root->right);

        // Print the right boundary in bottom-up manner
        printBoundaryRight(root->right);
    }
}
```

Delete elements in BST
deleteBST(node)
if(node == null)return
deleteBST(node->left); deleteBST(node->right)
if(node->left == null && node->right == null) free node

Design an algorithm to search for Anagrams of a word in a dictionary. The interviewer stressed on not processing all the words in the dictionary.

Pre-process the dictionary: Store the whole dictionary in a trie. This will give search O(1). Now generate all the permutations of the given word and check for the generated word in dictionary.

Total time complexity: O(n!) where n is the length of the word.

National Flag Problem - O(n) Time Complexity

https://www.youtube.com/watch?v=CNVN76UWpBo

Tc is O(n)

Sort the array in the order of 0,1,2
Int i=j=0, k=n-1

```
While(j<=k) {
 if( a[j]==0) {
   Swap(a[i],a[j];
   I++;j++;
} else if (a[j] ==1 ) { j++;}
else ( a[j]==2) {swap(a[j],a[k];k--}
```

Sort array with only 0s and 1s

```
/*Function to put all 0s on left and all 1s on right*/
void segregate0and1(int arr[], int size)
{
    /* Initialize left and right indexes */
    int left = 0, right = size-1;
    while (left < right)
    {
        /* Increment left index while we see 0 at left */
        while (arr[left] == 0 && left < right)
            left++;
        /* Decrement right index while we see 1 at right */
        while (arr[right] == 1 && left < right)
            right--;
        /* If left is smaller than right then there is a 1 at left
          and a 0 at right.  Exchange arr[left] and arr[right]*/
        if (left < right)
        {
            arr[left] = 0;
            arr[right] = 1;
            left++;
            right--;
        }
    }
}
```

Threads are reference to the predecessors and successors of the node according to an inorder traversal.

InOrder Successor of a Node

If node has right subtree-> get the leftmost element
IF node has no right subtree ->get the parent where the node is in its left subtree
https://gist.github.com/mycodeschool/6515e1ec66482faf9d79

```
struct Node* Getsuccessor(struct Node* root,int data) {
       // Search the Node - O(h)
       struct Node* current = Find(root,data);
       if(current == NULL) return NULL;
       if(current->right != NULL) {  //Case 1: Node has right subtree
              return FindMin(current->right); // O(h)
       }
       else {   //Case 2: No right subtree  - O(h)
              struct Node* successor = NULL;
              struct Node* ancestor      = root;
              while(ancestor != current) {
                     if(current->data < ancestor->data) {
                            successor = ancestor; // so far this is the deepest node for which current node is in left
                            ancestor = ancestor->left;
                     }
```

```
                    Else-
                          ancestor = ancestor->right;
              }
              return successor;
       }
}
```

InOrder Predessor in Opposite of Successor.Just look for max element on right and

```
if(curr->data > root->data) {
 Successor = ancestor;
 Ancestor = ancestor->right
} else {
 ans = ans->left;
}
```

Palindrome in Single Linked List

Go to mid node
Reverse the right half
Compare it with the left half

Preorder/Inorder to Binary Tree : TC n SC = O(n)

```
int search(char arr[], int strt, int end, char value);
struct node* newNode(char data);

/* Recursive function to construct binary of size len from
   Inorder traversal in[] and Preorder traversal pre[].  Initial values
   of inStrt and inEnd should be 0 and len -1.  The function doesn't
   do any error checking for cases where inorder and preorder
   do not form a tree */
struct node* buildTree(char in[], char pre[], int inStrt, int inEnd)
{
  static int preIndex = 0;

  if(inStrt > inEnd)
     return NULL;

  /* Pick current node from Preorder traversal using preIndex
    and increment preIndex */
  struct node *tNode = newNode(pre[preIndex++]);

  /* If this node has no children then return */
  if(inStrt == inEnd)
    return tNode;

  /* Else find the index of this node in Inorder traversal */
  int inIndex = search(in, inStrt, inEnd, tNode->data);

  /* Using index in Inorder traversal, construct left and
     right subtress */
  tNode->left = buildTree(in, pre, inStrt, inIndex-1);
  tNode->right = buildTree(in, pre, inIndex+1, inEnd);

  return tNode;
}
 buildTree(in,pre,0,n-1);
int search(char arr[], int strt, int end, char value)
{
  int i;
  for(i = strt; i <= end; i++)
```

```
  {
    if(arr[i] == value)
      return i;
  }
}
```

PostORder from INorder and Preorder

```
   void printPostOrderUtil(int pre[], int& preIdx, int in[],
   int low, int high)
   {
      if(low>high)
           return;
      int idx=search(in, pre[preIdx], low, high);
      int curr=preIdx;
      preIdx++;
      printPostOrderUtil(pre, preIdx, in, low, idx-1);
      printPostOrderUtil(pre, preIdx, in, idx+1, high);
      cout<<pre[curr]<<" ";
   }
   // Prints postorder traversal from given inorder and
   preorder traversals
```

Radix Sort first mod by 10 and by 1 amd mod by 100 and by 10 till the max integer value

Node at K distance

```
void printKDistant(node *root , int k)
{
   if(root == NULL)
      return;
   if( k == 0 )
   {
      printf( "%d ", root->data );
      return ;
   }
   else
   {
      printKDistant( root->left, k-1 ) ;
      printKDistant( root->right, k-1 ) ;
   }
```

Given a sorted array with only 0's and 1's.Count the number of 0's. e.g: 0 0 0 0 1 1 Ans: 4.

```
if(high >=low) {
            int mid = (low+high)/2;
            //System.out.println(mid);
            if(mid == 0 && a[mid]==0) {
                  return mid+1;
            }
            if((mid == 0 || a[mid-1]==0)&& a[mid]==1) {
```

```
                return mid;
            }
            if (a[mid]==1) {
                return returnCount(arr,low,mid-1);
            } else {
                return returnCount(arr,mid+1,high);
            }
        }
            return -1;
        }
```

Find the largest pair sum in an unsorted array

```
int findLargestSumPair(int arr[], int n)
{
    // Initialize first and second largest element
    int first, second;
    if (arr[0] > arr[1])
    {
        first = arr[0];
        second = arr[1];
    }
    else
    {
        first = arr[1];
        second = arr[0];
    }

    // Traverse remaining array and find first and second largest
    // elements in overall array
    for (int i = 2; i<n; i ++)
    {
        /* If current element is greater than first then update both
          first and second */
        if (arr[i] > first)
        {
            second = first;
            first = arr[i];
        }

        /* If arr[i] is in between first and second then update second  */
        else if (arr[i] > second && arr[i] != first)
            second = arr[i];
    }
    return (first + second);
}
```

Group multiple occurrence of array elements ordered by first occurrence

```
public class MinVal {
      static int b[] = {5, 3, 5, 1, 3, 3};
      public static void main(String[] args) {
            // TODO Auto-generated method stub
            HashMap<Integer, Integer> map =new HashMap<Integer, Integer>();
            for(int i=0;i<b.length;i++) {
                  if(map.get(b[i]) == null) {
                        map.put(b[i], 1);
```

```
                } else {
                        int count = map.get(b[i]);
                        count++;
                        map.put(b[i], count);
                }
            }
            for(int i=0;i<b.length-1;i++) {
                int count=0;
                if(map.get(b[i]) != null) {
                 count = map.get(b[i]);
                }

                if(count != 0) {
                        for(int j=0;j<count;j++) {
                                System.out.println(b[i] + " ");
                        }
                }
                map.remove(b[i]);
            }
        }
}
```

Given a sorted and rotated array, find if there is a pair with a given sum

```
l=0;
r=length-1
 while (l != r)
    {
         // If we find a pair with sum x, we return true
         if (arr[l] + arr[r] == x)
             l++;
             r--;
              return true;

         // If current pair sum is less, move to the higher sum
         if (arr[l] + arr[r] < x)
              l + +;
         else  // Move to the lower sum side
              r--;
    }
```

```
HashSet<Integer> pairs = new HashSet<>();
int pair=0;
for(int i=0;i<arr.length;i++) {
if(pairs.contains(arr[i]) ) {
pair = sum-arr[i];
System.out.println("Pair: " + arr[i] + " - "+ pair);
pairs.remove(arr[i]);
}
else {
pair = sum-arr[i];
pairs.add(pair);
}
```

In conditional statements u pick the complexity of condition of worst case
If(O(n)) else (O(n2)) then tc = 0 (n^2)

Kadence Algorithm

```
int maxSubArraySum(int a[], int size)
{
   int max_so_far = 0, max_ending_here = 0;
   int i;
   for(i = 0; i < size; i++)
   {
     max_ending_here = max_ending_here + a[i];
     if(max_ending_here < 0)
         max_ending_here = 0;


     /* Do not compare for all elements. Compare only
        when  max_ending_here > 0 */
     else if (max_so_far < max_ending_here)
         max_so_far = max_ending_here;
   }
   return max_so_far;
}
int maxSubArraySum(int a[], int size)
{
   int max_so_far = a[0], i;

  int curr_max = a[0];

   for (i = 1; i < size; i++)
   {
        curr_max = max(a[i], curr_max+a[i]);
        max_so_far = max(max_so_far, curr_max);
   }
   return max_so_far;
}
```

Max Product SubArray

Max_ending_here = min_ending_here = max_so_far =1

```
 for (int i = 0; i < n; i++)
    {
        if (arr[i] > 0)
        {
            max_ending_here = max_ending_here*arr[i];
            min_ending_here = min (min_ending_here * arr[i], 1);
        }

        else if (arr[i] == 0)
        {
            max_ending_here = 1;
            min_ending_here = 1;
        }

        else
        {
            int temp = max_ending_here;
            max_ending_here = max (min_ending_here * arr[i], 1);
            min_ending_here = temp * arr[i];
```

```
        }

        if (max_so_far <  max_ending_here)
          max_so_far  =  max_ending_here;
    }
```

http://www.geeksforgeeks.org/maximum-product-subarray/

Max Product of three Numbers in array

```
public static void product(int a[]) {


                int high = Math.max(a[0], a[1]);
                int low = Math.min(a[0], a[1]);

                int high_prod_2 = a[0]*a[1];
                int low_prod_2 = a[0]*a[1];
                int high_prod_3 = a[0]*a[1]*a[2];

                for (int i = 2; i < a.length ; i++) {

                int curr = a[i];
                int max_3 = Math.max(high_prod_2 *curr ,low_prod_2 * curr);
                        int max_2 = Math.max(high * curr, low * curr);
                        int min_2 = Math.min(high*curr, low*curr);


                        high_prod_3 = Math.max(high_prod_3, max_3 );
                        high_prod_2 = Math.max(high_prod_2, max_2);
                        low_prod_2 = Math.min(low_prod_2, min_2);

                        high = Math.max(high, curr);
                        low = Math.min(low, curr);

                }
                System.out.println(high_prod_3);

        }
```

Find Any element in Sorted Rotated Array

```
                while(low<=high) {
                        int mid = (low+high)/2;
                        if(a[mid] == k) {
                                System.out.println(a[mid]);
                                break;
                        }
                        if(a[low] <= a[mid]) {
                                if(a[low] <=k  && k < a[mid]) {
                                        high =mid-1;
                                } else {
                                        low=mid+1;
                                }
                        }else if(a[mid] < a[high]) {
                                if(a[mid] < k && k <= a[high]) {
                                        low = mid+1;
                                } else {  high = mid-1;
                } else {
                  low++;    // if there are duplicates as we are comparing with low skip low with duplicates
                }
                    }
```

http://articles.leetcode.com/2010/04/searching-element-in-rotated-array.html

O(logn) Time complexity

http://www.geeksforgeeks.org/find-the-element-that-appears-once-in-a-sorted-array/ - Ologn
http://ideone.com/Mos8Kl

```
        while (low < high) {
                        int mid = (low+high)/2;
                        if (mid %2 ==0 ){
                                if(a[mid] == a[mid+1]) {
                                        low = mid+2;
                                } else {
                                        high =mid;
                                }
                        } else {
                                if(a[mid] == a[mid-1]) {
                                        low = mid+1;
                                } else {
                                        high = mid-1;
                                }
                        }
                }
                if (low == high) {
                        System.out.println(a[low]);
                }
```

http://www.geeksforgeeks.org/majority-element/ - O(n)

```
  int maj_index = 0, count = 1;
    int i;
    for(i = 1; i < size; i++)
    {
        if(a[maj_index] == a[i])
            count++;
        else
            count--;
        if(count == 0)
        {
            maj_index = i;
            count = 1;
        }
    }
```

Longest Increasing Subsequence – O(nlogn) /O(n2)

First Non Repeated Character in a String

```
public class NonRepeatedChar {
 public static void main(String[] args) {
  String str = "javapassion";
  int[] count = new int[128];
  char[] charArr = str.toLowerCase().toCharArray();
  for (char c : charArr) {
   count[c]++;
  }
  for (char c : charArr) {
   if (count[c] == 1) {
    System.out.println("First Non repeated character is : " + c);
    break;
   }
  }
```

```
 }
}
```

Pending – Longest Increasing Subsequence

Find the starting of the loop in a circular linked list
http://umairsaeed.com/blog/2011/06/23/finding-the-start-of-a-loop-in-a-circular-linked-list/

```
  while (n2.next != null) { // n2 moves faster, will reach end
before n1
        n1 = n1.next;          // 1 * speed
        n2 = n2.next.next;     // 2 * speed
        if (n1 == n2) break;  // end if reach the meeting point.
    }
    // Error check - there is no meeting point, and therefore no
loop
    if (n2.next == null) return null; // if ends cuz n2 is the end
of the linked list
    // Move n1 to Head. Keep n2 at Meeting Point. Each are k
steps  from the Loop Start. If they move at the same pace, they
must  meet at Loop Start.
    n1 = head;
    while (n1 != n2) {
        n1 = n1.next;          // same speed
        n2 = n2.next;
    }
    // Now n2 points to the start of the loop.
    return n2;
}
```

Remove Duplicates in an unsorted Linked List Using Hashing - O(n)
Non Hashing Method:
https://gist.github.com/Kadenwxz/5876714

```
if(head == null)
       return;
    LinkedListNode previous = head;
    LinkedListNode current = head.next;
    while(current! = null){
       LinkedListNode runner = head;
       while(runner! = current){
          if(runner.data == current.data){
             /* Remove current */
             current = current.next;
             previous.next = current;
```

```
            break;
        }
        runner = runner.next;
      }
      if(runner == current){
        previous = previous.next;
        current = current.next;
```

Given an array of size n-1 whose entries are integers in the range [1, n], find an integer that is missing. Use only O(1) space and treat the input as read-only.

```
Missing Number = n(n+1)/2 – Sum of all the elements in Array
```

Find the element that appears once –O(n)

```
  for( int i=0; i< n; i++ )
    {
        /* The expression "one & arr[i]" gives the bits that are
           there in both 'ones' and new element from arr[].  We
           add these bits to 'twos' using bitwise OR

           Value of 'twos' will be set as 0, 3, 3 and 1 after 1st,
           2nd, 3rd and 4th iterations respectively */
        twos  = twos | (ones & arr[i]);


        /* XOR the new bits with previous 'ones' to get all bits
           appearing odd number of times

           Value of 'ones' will be set as 3, 0, 2 and 3 after 1st,
           2nd, 3rd and 4th iterations respectively */
        ones  = ones ^ arr[i];


        /* The common bits are those bits which appear third time
           So these bits should not be there in both 'ones' and 'twos'.
           common_bit_mask contains all these bits as 0, so that the bits can
           be removed from 'ones' and 'twos'

           Value of 'common_bit_mask' will be set as 00, 00, 01 and 10
           after 1st, 2nd, 3rd and 4th iterations respectively */
        common_bit_mask = ~(ones & twos);


        /* Remove common bits (the bits that appear third time) from 'ones'

           Value of 'ones' will be set as 3, 0, 0 and 2 after 1st,
           2nd, 3rd and 4th iterations respectively */
        ones &= common_bit_mask;


        /* Remove common bits (the bits that appear third time) from 'twos'

           Value of 'twos' will be set as 0, 3, 1 and 0 after 1st,
           2nd, 3rd and 4th itearations respectively */
        twos &= common_bit_mask;

        // uncomment this code to see intermediate values
        //printf (" %d %d \n", ones, twos);
    }

    return ones;
}
```

http://www.geeksforgeeks.org/find-the-element-that-appears-once/

A number is POT

n > 0 && (n & (n - 1) == 0)

Sort a Nearly Sorted Array using Insertion Sort O(nk)

Reverse the words in the String .

1)Use a stack and push each word in the stack and pop the stack . O(n)

- Reverse individual word
- Then reverse whole string

Reverse A string

```
for(int i=0;i<a.length;i++) {
		arr[a.length-i-1] = a[i]; }
```

Another way to reverse is swap first and last element and then increment pointer from starting index and decrement pointer from last index

```
		int i=0;
		int j=n-1;
		while (i<j) {
			String temp = a[i];
			a[i] = a[j];
			a[j] = temp;
			i++;
			j--;
		}
		for(int k=0;k<=a.length-1;k++) {
			System.out.println(a[k]);
		}
```

http://www.geeksforgeeks.org/reverse-words-in-a-given-string/ - O(n)

```
while( *temp )
  {
    temp++;
    if (*temp == '\0')
    {
      reverse(word_begin, temp-1);
    }
    else if(*temp == ' ')
    {
      reverse(word_begin, temp-1);
      word_begin = temp+1;
    }
  }
/*STEP 2 of the above algorithm */
  reverse(s, temp-1)
```

Frequency of a number in array

```
int main()

{

    string str = "aaabcc";

    int count=0;

    int i=0,j;

    int l = str.length();

    while(i<l)
```

```
    {


        cout << str[i];
                count=1;
        j=i;
        while(j+1<l && str[j+1]==str[i])
        {
            count++;
            j++;
        }
        cout << count;
        i=j+1;
    }
    return 0;
}
```

Smallest Number

Remove characters from the first string which are present in the second string

- Count array of second string
- If count[element of first string] == 0 then add the element
- Else skip
- O(m+n)
- http://www.geeksforgeeks.org/remove-characters-from-the-first-string-which-are-present-in-the-second-string/

Nearest Smallest Number

```
 for (int i=0; i<n; i++)
    {
        // Keep removing top element from S while the top
        // element is greater than or equal to arr[i]
        while (!S.empty() && S.top() >= arr[i])
            S.pop();

        // If all elements in S were greater than arr[i]
        if (S.empty())
            cout << "_, ";
        else  //Else print the nearest smaller element
            cout << S.top() << ", ";

        // Push this element
        S.push(arr[i]);
    }
}
```

http://www.intertechtion.com/answers/6-22-2012.html - Sort odd and even elements

Check two strings are anagram

```
bool anagrams(string s,string t)
{
    if (s.length() ! = t.length())
        return false;
    int letters[256] = {0};
    for(int i=0;i<s.length();i++)
    {
        letters[s[i]]++;
        letters[t[i]]--;
    }
    for (int i = 0; i < 256; i++)
    {
        if(letters[i] != 0)
            return false;
    }
    return true;
}
```

http://www.crazyforcode.com/algorithm/

Find Occurrence of a number in array - O(logn)

```
int first(int arr[], int low, int high, int x, int n)
{
  if(high >= low)
  {
    int mid = (low + high)/2;  /*low + (high - low)/2;*/
    if( ( mid == 0 || x > arr[mid-1]) && arr[mid] == x)
      return mid;
    else if(x > arr[mid])
      return first(arr, (mid + 1), high, x, n);
    else
      return first(arr, low, (mid -1), x, n);
  }
  return -1;
}


/* if x is present in arr[] then returns the index of LAST occurrence
   of x in arr[0..n-1], otherwise returns -1 */
int last(int arr[], int low, int high, int x, int n)
{
  if(high >= low)
  {
    int mid = (low + high)/2;  /*low + (high - low)/2;*/
    if( ( mid == n-1 || x < arr[mid+1]) && arr[mid] == x )
      return mid;
    else if(x < arr[mid])
      return last(arr, low, (mid -1), x, n);
```

```
    else
      return last(arr, (mid + 1), high, x, n);
  }
  return -1;
}  return j-i+1;
```

http://www.geeksforgeeks.org/count-number-of-occurrences-in-a-sorted-array/

Move 0's to end of array

```
for(int i=0;i<a.length;i++) {
                        if(a[i] != 0) {
                                a[k] =a[i];
                                k++;
                        }
                }
                System.out.println(k);
                for(int j=k;j<a.length;j++) {
                        a[j] = 0;
                }
```

http://qa.geeksforgeeks.org/882/find-the-palindromes-in-array-of-strings?show=909#a909

http://www.geeksforgeeks.org/find-union-and-intersection-of-two-unsorted-arrays/

http://www.geeksforgeeks.org/find-a-fixed-point-in-a-given-array/

Find Prime Number or not

```
int j=2;
boolean flag = false;
while (j <= sum/2) {
if(sum % j == 0) {
flag = true;
}
j++;
```

Count Trailing Zeroes in Factorial

http://www.geeksforgeeks.org/count-trailing-zeroes-factorial-number/

```
int count=0;
int n=100;
for(int i=5;i<=n;i=i*5) {
        count += n/i;
}
```

Height of a complete Binary Tree – (LogN)
http://geeks.i-pupil.com/index.php?option=com_content&view=article&id=201&catid=95

Maximum Number of Nodes in a Complete Binary Tree – (2^h+1) where
h = height
No. of nodes at each level – (No.of nodes / 2^h+1)
Build Heap Tc is O(n)

The leaves in a tree start from (n/2+1 to n)

The total distance form root to leaf in heap sort in O(logn) . So heap sort is O(nlogn) O(n) Is from Build_Max_heap

Return Max Element from Heap Sort

Max = a[1]
a[1] = a[length]
length=length-1
max_heapify(array,1)

Increase Key

A[i]=key
While(I > 1 && a[i/2] < a[i]) // Parent is less than he node
Swap(a[i],a[i/2])
I=i/2
TC =O(logn)

MAX-HEAP-INSERT(A, key)

heap-size[A] ← heap-size[A] + 1
A[heap-size[A]]←−∞
HEAP-INCREASE-KEY(A, heap-size[A], key)

DecreaeKey – Replace key and call max_heapify

Find min -> All the mimimum elements will be in the leaf nodes .
SO leaf nodes are from n/2+1 to n . So iterate and find min O(n) or create a min heapify with those leaf nodes and get the root node O(1)

http://clrs.skanev.com/index.html → Solutions for Cormen

n! permutations are possible and there are no more than 2^h leaves
n! <= l <= 2^h => log(n1) – Sterling Approximation nlogn

The length of the longest path from the root of a decision tree to any of its reachable leaves represents the worst-case number of comparisons that the corresponding sorting algorithm performs. Consequently, the worst-case number of comparisons for a given comparison sort algorithm equals the height of its decision tree

http://www.gatecse.org/best-video-lectures-for-gate-in-cse/
http://www.bowdoin.edu/~ltoma/teaching/cs231/fall07/Lectures/sortLB.pdf

http://www.dsalgo.com/2013/02/BinaryTreeSumChildNodes.php.html

```
int toSumTree(struct node *node)
{
    // Base case
    if(node == NULL)
      return 0;

    // Store the old value
    int old_val = node->data;

    // Recursively call for left and right subtrees and store the sum as
    // new value of this node
    node->data = toSumTree(node->left) + toSumTree(node->right);

    // Return the sum of values of nodes in left and right subtrees and
    // old_value of this node
    return node->data + old_val;
}
```

http://qa.geeksforgeeks.org/978/questions-linked-geeksforgeeks-almost-everything-linked

http://qa.geeksforgeeks.org/1536/questions-binary-geeksforgeeks-almost-everything-binary

```
Merge Two Sorted Arrays – O(n)
int length = my_array.length + alices_array.length;

		int[] new_array = new int[length];
		int i = 0;
		int j = 0;
		int count = 0;
		while (count < length) {
			if(i < my_array.length && my_array[i] < alices_array[j] )
{
				new_array[count] = my_array[i];
					i++ ;

			} else {
				new_array[count] = alices_array[j];
				j++;
			}

			count++;
		}
		for(int k=0;k<length;k++) {
			System.out.println(new_array[k]);
```

http://gatecse.in/wiki/Best_video_lectures_for_CSE

http://www.geeksforgeeks.org/how-to-check-if-a-given-array-represents-a-binary-heap/

http://articles.leetcode.com/2010/11/stack-that-supports-push-pop-and-getmin.html

Use two stacks .

- Push an element into s1 and into s2
- If the next value being pushed in s1 < s1.top , push it into s2 also
- To retrieve min, s2.top
- While pop, if s1.top == s2.top pop from both stacks

  http://prismoskills.appspot.com/lessons/Arrays/Next_largest_element_for_each_element.jsp

Remove a loop in linked list

1) Detect Loop using Floyd's Cycle detection algo and get the pointer to a loop node.

2) Count the number of nodes in loop. Let the count be k.

3) Fix one pointer to the head and another to kth node from head.

4) Move both pointers at the same pace, they will meet at loop starting node.

5) Get pointer to the last node of loop and make next of it as NULL.

After finding the starting point

```
While(p1 -> next ! = p2) {
        p1 = p1->next;
}
 p1->next = null;
```

Another method is

1) Detect loop
2) Set p1 = head , p2 = same point
3) While(p2->next != p1) {
   P1=p1->next
   P2=p2->next
   }
   P2->next = null;

Where in a max-heap might the smallest element reside, assuming that all elements are distinct? Any leaf node from n/2+1 to n

What are the minimum and maximum numbers of elements in a heap of height h? Since a heap is an almost-complete binary tree (complete at all levels except possibly the lowest), it has at most 1+2+22 +23 +…+2h =2h+1-1 elements (if it is complete) and at least 2h -1+1=2h elements

Height of a tree

The number of edges on a simple downward path from a root to a leaf. Note that the height of a tree with n node is $\lfloor \lg n \rfloor$ which is $\Theta(\lg n)$. This implies that an n-element heap has height $\lfloor \lg n \rfloor$

In order to show this let the height of the n-element heap be h. From the bounds obtained on maximum and minimum number of elements in a heap, we get

$$-2^h \le n \le 2^{h+1}-1$$

Where n is the number of elements in a heap.

$$2^h \le n \le 2^{h+1}$$

Taking logarithms to the base 2

$$h \le \lg n \le h+1$$

It follows that $h = \lfloor \lg n \rfloor$.

removal from a heap which is O(log(n)). You do it n-1 times so it is O(nlog(n)). The last steps are cheaper but for the reverse reason from the building of the heap, most are log(n) so it is O(nlog(n)) overall for this part. The build part was O(n) so it does not dominate. For the whole heap sort you get O(nlog(n)).

Q2: Exercise 7.2-2 (10 points)

What is the running time of `Quicksort` when all elements of array A have the same value?

- If all elements are the same, the quick sort partition return index $q = r$. This means, the problem with size n is reduced to one subproblem with size $n-1$. So the recurrence is $T(n) = T(n-1) + n$. By iteration method, $T(n) = \Theta(n^2)$.

Q3: Exercise 7.2-3 (10 points)

Show that the running time of `QuickSort` is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

- In each partition, the pivot element is always the smallest element. Therefore, each partition will produce two subarrays. The first subarray contains only one element (the smallerest element) and this element is in its correct position. The second subarray contains the remaining elements.
  In this case, the running time recurrence will be $T(n) = T(n-1) + n$. Thus, $T(n) = \Theta(n^2)$.

http://codercareer.blogspot.com/2012/02/no-32-remove-numbers-in-array.html
Remove Numbers in an Array;
Two pointers are defined to solve this problem: The first pointer (denoted as `p1`) moves forward until it reaches a number equal to the target value, which is initialized to the beginning of array. The other (denoted as `p2`) moves backward until it reaches a number not equal to the target value, which is initialized to the end of array. Two numbers pointed by `p1` and `p2` are swapped. We repeat the moving and swapping operations until all target numbers are scanned.

Intersection of sorted array (mlogn) - Take element from one array and search in another array

Rotate a array by N. N can be smaller of greater than the array length.
Rotate 0 to n-1
Rotate 0 to k
Rotate k to n-1

Replace spaces with '%20'

- Count the total space
- s in a string in one iteration, say the count is spaceCount
- Calculate the new length of a string by newLength = length + 2*spaceCount; (we need two more places for each space since %20 has 3 characters, one character will occupy the blank space and for rest two we need extra space)
- Do another iteration in reverse direction
- If you encounter the space, for next 3 spaces put %,2,0.
- If you encounter the character, copy it

```
public void replace(String s1, int length) {
                char[] chars = s1.toCharArray();
                int spaceCount = 0;
                for (int i = 0; i < length; i++) {
```

```
                        if (chars[i] == ' ') {
                                spaceCount++;
                        }
                }
                int newLength = length + 2 * spaceCount;
                char [] charsNew = new char [newLength];
                for (int i = length - 1; i >= 0; i--) {
                        if (chars[i] == ' ') {
                                charsNew[newLength - 1] = '0';
                                charsNew[newLength - 2] = '2';
                                charsNew[newLength - 3] = '%';
                                newLength = newLength - 3;
                        } else {
//                              System.out.println(chars[i]);
                                charsNew[newLength - 1] = chars[i];
                                newLength = newLength - 1;
                        }
                }
                System.out.println("Output String : " +
String.valueOf(charsNew));
        }
```

Implement an algorithm to determine if a string has all unique characters. What if
you can not use additional data structures?

```
public static boolean isUniqueChars2(String str) {
boolean[] char_set = new boolean[256];
for (int i = 0; i < str.length(); i++) {
     int val = str.charAt(i);
     if (char_set[val]) return false;
     char_set[val] = true;
}
8 return true;
```

Even numbers at even indexed and odd numbers at odd indexes in an array
For this same as 0 and 1s . But i=0 and j=1 . I incremenet by 2 and j increment by 2

```
arrangeArray (int a[], int size)
{
    oddInd = 1;
    evenInd = 0;
    while (true)
    {
        while (evenInd < size && a[evenInd]%2 == 0)
            evenInd += 2;
        while (oddInd < size && a[oddInd]%2 == 1)
            oddInd += 2;
        if (evenInd < size && oddInd < size)
            swap (a[evenInd], a[oddInd])
        else
            break;
    }
```

Post Order using one Stack

```
void postOrderIterative(struct Node* root)
{
    // Check for empty tree
    if (root == NULL)
        return;

    struct Stack* stack = createStack(MAX_SIZE);
    do
    {
        // Move to leftmost node
        while (root)
        {
            // Push root's right child and then root to stack.
            if (root->right)
                push(stack, root->right);
            push(stack, root);

            // Set root as root's left child
            root = root->left;
        }

        // Pop an item from stack and set it as root
        root = pop(stack);

        // If the popped item has a right child and the right child is not
        // processed yet, then make sure right child is processed before root
        if (root->right && peek(stack) == root->right)
        {
            pop(stack);  // remove right child from stack
            push(stack, root);  // push root back to stack
            root = root->right; // change root so that the right
                                // child is processed next
        }
        else  // Else print root's data and set root as NULL
        {
            printf("%d ", root->data);
            root = NULL;
        }
    } while (!isEmpty(stack));
}
```

Minimum depth of a binary tree.

Same like Find height of binary tree using queue but when you encounter a leaf node return depth

```
Find Equilibrium Point == Left Sum = Right Sum
int equilibrium(int arr[], int n)
{
   int sum = 0;      // initialize sum of whole array
   int leftsum = 0; // initialize leftsum
   int i;

   /* Find sum of the whole array */
   for (i = 0; i < n; ++i)
        sum += arr[i];
   for( i = 0; i < n; ++i)
   {
      sum -= arr[i]; // sum is now right sum for index i
      if(leftsum == sum)
        return i;
      leftsum += arr[i]
```

1. **A binary tree is balanced** if for each node it holds that the number of inner nodes in the left subtree and the number of inner nodes in the right subtree differ by at most 1.

2. A binary tree is balanced if for any two leaves the difference of the depth is at most 1.

Convert a Given Tree to Its Sum Tree

```
        int toSumTree(struct node *node)
{
    // Base case
    if(node == NULL)
      return 0;
    // Store the old value
    int old_val = node->data;
    // Recursively call for left and right subtrees and store the sum as
    // new value of this node
    node->data = toSumTree(node->left) + toSumTree(node->right);
    // Return the sum of values of nodes in left and right subtrees and
    // old_value of this node
    return node->data + old_val;
}
```

Lowest Common Ancestor
Inorder – Elements between the nodes
Post Order – Last element from the elements between in post order

Check Anagrams are Palindromes or not

```
public static boolean checkPalindrome(String input)
{
         int [] count = new int[26];
         for( int i = 0; i < input.length(); i++ )
         {
             char ch = input.charAt(i);
             count[ch-'a']++;
         }
         int oddOccur = 0;
         for( int cnt:count )
         {
             if( oddOccur > 1) // more than 1 char should have odd frequency
                 return false;
             if( cnt%2 == 1 )
                 oddOccur++;
         }
         return true;
}
```

Length of Longest Substring without repeating characters

```
For(I -> n)
       Count[a[i]]++
       If(count[a[i]] > 1 )
              Max = count;
              Count = 0;
       Else count++;
Return max
```

Count Pairs with a Difference

```
        public static void main (String[] args {
          Integer a[] = new Integer[] {7,6,23,19,10,11,9,3,15};
          int count = 0;
          int k = 4;
          Map map= new HashMap();
```

```
for (int i=0;i<a.length;i++) {
  if (!map.containsKey(a[i]))
  map.put(a[i],1);
}for (int i=0;i<a.length;i++) {
   if (map.containsKey(a[i]+k) || map.containsKey(a[i]-k)
   map.remove(a[i])}
```

http://www.crazyforcode.com/passing-car-east-west-codility/

```
for(int i = 0; i < N; i++)
        if(A[i]==0)
        {
            totalOfZeros ++;
        }
        else if (A[i]==1)
        {
            count = count + totalOfZeros;
        }
        if(count > 1000000000) return -1;
    }
    return count;
```

http://www.crazyforcode.com/find-minimum-distance-numbers/

```
        int minDistance(int *input, int n1, int n2, int length)
{
        int pos1 = INT_MAX;
        int pos2 = INT_MAX;
        int distance = length+1;
        int newDistance;
        pos1 = pos2 = distance = length;

        for (int i = 0; i < length; i++)
        {
                if (input[i] == n1)
                        pos1 = i;
                else if (input[i] == n2)
                        pos2 = i;

                if (pos1 < length && pos2 < length)
                {
                        newDistance = abs(pos1 - pos2);
                        if (distance > newDistance)
                                distance = newDistance;
                }
        }

        return distance == length+1 ? -1 : distance;
}
```

Find Leader in Array – Where all elements on right are smaller than the number

```
 Max = a[length-1]
For(I = length -2 -> 0)
        If(a[i] > max )
                Print a[i]
                Max = a[i]
```

http://www.crazyforcode.com/leaders-array/

Find Merge Point in Sorted Array

O(m+n) approach below

```
vector<int> findIntersection(vector<int> A, vector<int> B) {
  vector<int> intersection;
  int n1 = A.size();
  int n2 = B.size();
  int i = 0, j = 0;
  while (i < n1 && j < n2) {
    if (A[i] > B[j]) {
      j++;
    } else if (B[j] > A[i]) {
      i++;
    } else {
      I*ntersection.push_back(A[i]);
      i++;
j++;
```

This can also be done by taking element in one array and searching in 2nd array using binary search

O(m*logn) approach

Shift Zeros to Right
http://techieme.in/shift-zeroes-to-right/

Merge Intervals
http://www.programcreek.com/2012/12/leetcode-merge-intervals/

```
/* A function that constructs Balanced Binary Search Tree from a sorted array */
Sorted array to bst
```

```
struct TNode* sortedArrayToBST(int arr[], int start, int end)
{
    /* Base Case */
    if (start > end)
      return NULL;

    /* Get the middle element and make it root */
    int mid = (start + end)/2;
    struct TNode *root = newNode(arr[mid]);

    /* Recursively construct the left subtree and make it
       left child of root */
    root->left =  sortedArrayToBST(arr, start, mid-1);

    /* Recursively construct the right subtree and make it
       right child of root */
    root->right = sortedArrayToBST(arr, mid+1, end);

    return root;
}
```

https://github.com/mission-peace/interview/blob/master/src/com/interview/tree/FenwickTree.java

Get Parent .
2's complement - AND and Subtract

Get Next-
2's complement - AND and Add

Get sum upto n elements in O(logn) TC
https://www.hackerearth.com/notes/BLANKRK/binary-indexed-treebit/
http://theoryofprogramming.com/2014/12/24/binary-indexed-tree-or-fenwick-tree/
https://www.hackerearth.com/notes/binary-indexed-tree-made-easy-2/

http://www.programcreek.com/2014/05/leetcode-implement-trie-prefix-tree-java/

Find consecutive elements sum equal to target

```
public static boolean solution(int[] nums, int target) {
        if(nums.length == 0) {
            return target == 0;
        }
        int start = 0;
        int end = 0;
        int sum = nums[0];
        if (sum == target) {
            return true;
        }
        while(end < nums.length) {
            if(sum > target) {
                sum -= nums[start];
                start++;
            } else {
                end++;
                if(end < nums.length) {
                    sum += nums[end];
```

```
            }
        }
        if (sum == target) {
            return true;
        }
```

http://www.geeksforgeeks.org/longest-prefix-matching-a-trie-based-solution-in-java/

Binary Search Without length of the array

```
1.  lower = 1, upper = 1;

2.  while (A[upper] < element) upper *= 2;

3.  Normal binary search on (lower, upper)
```

N Steps Problem –

Nth step can be reached from either n-1th step or n-2th step
Y(n) = y(n-1)+y(n-2)
Y(n)=F(n+1)
http://ms.appliedprobability.org/data/files/Articles%2033/33-1-5.pdf
https://web.stevens.edu/ssm/nov13.pdf

We have a 10 step staircase. One can climb up 1,2, or 3 stairs at a time. How many ways are there to climb up the stairs? Let's let Nk denote the number of ways to climb k stairs in the manner described. (So we're looking for N10.) Notice that for $k \geq 4$ there are 3 "moves" one can do for your first step: you can climb 1,2, or 3 stairs. If you climb 1 stair then there are N_{k-1} ways to finish; if you climb 2 stairs there are N_{k-2} ways to finish; and if you climb 3 stairs there are N_{k-3} ways to finish. Thus: $N_k = N_{k-1} + N_{k-2} + N_{k-3}$

Well, it's pretty easy to see that for the $k < 4$ we have $N_1 = 1$, $N_2 = 2$ and $N_3 = 4$, so using the above we can calculate N_{10} using brute force.
$N_4 = N_3 + N_2 + N_1 = 4 + 2 + 1 = 7$
$N_5 = N_4 + N_3 + N_2 = 7 + 4 + 2 = 13$
$N_6 = N_5 + N_4 + N_3 = 13 + 7 + 4 = 24$
$N_7 = N_6 + N_5 + N_4 = 24 + 13 + 7 = 44$
$N_8 = N_7 + N_6 + N_5 = 44 + 24 + 13 = 81$
$N_9 = N_8 + N_7 + N_6 = 81 + 44 + 24 = 149$
$N_{10} = N_9 + N_8 + N_7 = 149 + 81 + 44 = 274$
There are 274 ways to climb the stairs

Palindrome Number

```
n = num;
 rev = 0;
 while (num > 0)
 {
      dig = num % 10;
      rev = rev * 10 + dig;
      num = num / 10;
 }
```

If `n == rev` then `num` is a palindrome:

O(nlogn)= log(n!)

3Sums problem - O(n^2)

For(i=o;n)
---Sum of two digits equal to a number problem---

https://www.nayuki.io/page/next-lexicographical-permutation-algorithm

Find next highest number from a number

- Find the highest index i such that s[i] < s[i+1]. If no such index exists, the permutation is the last permutation.
- Find the highest index j > i such that s[j] > s[i]. Such a j must exist, since i+1 is such an index.
- Swap s[i] with s[j].
- Reverse all the order of all of the elements after index i

http://www.ardendertat.com/2012/01/02/programming-interview-questions-24-find-next-higher-number-with-same-digits/

```
 i=n-2
while( a[i] > a[i+1]) i--
j=n-1
while( a[i] > a[j] ) j--
swap(a[i],a[j] )
reverse(i+1,n)
```

```
/* Function to calculate x raised to the power y in O(logn)*/
int power(int x, unsigned int y)
{
    int temp;
    if( y == 0)
        return 1;
    temp = power(x, y/2);
    if (y%2 == 0)
        return temp*temp;
    else
        return x*temp*temp;
}
```

Time Complexity of optimized solution: O(logn)

Let us extend the pow function to work for negative y and float x.

```
/* Extended version of power function that can work
 for float x and negative y*/
#include<stdio.h>

float power(float x, int y)
{
    float temp;
    if( y == 0)
       return 1;
    temp = power(x, y/2);
    if (y%2 == 0)
        return temp*temp;
    else
    {
        if(y > 0)
            return x*temp*temp;
        else
            return (temp*temp)/x;
    }
}
```

The space complexity of recursive tree traversal is O(h) h is height of the tree

Pigeonhole Principle

IF there are n boxes and k holes then atleast one box will have n/k
http://www.careercup.com/question?id=14990323

Given An Array with N integer with values ranging from 1 to N. there is only one duplicate in the Array.
Find out Duplicate value.

i.e.
A = { 10,6,3,4,7,5,2,4,9,1}

Finding the starting of loop in linked list solution

http://www.geeksforgeeks.org/find-the-maximum-element-in-an-array-which-is-first-increasing-and-then-decreasing/

Maximum Element

```
while(l <= h) {
                        int mid = (l+h)/2;
                        if(a[mid] > a[mid-1] && a[mid] > a[mid+1]) {
                                System.out.println(a[mid]);
                        }
                        if( a[mid] < a[mid+1] && a[mid] > a[mid-1]) {
                                System.out.println("entered here");
                                l = mid+1;
                        } else { // a[mid] > next && a[mid] < prev
                                h=mid-1;
                        }
                }
```

Egg Dropping
http://www.geeksforgeeks.org/dynamic-programming-set-11-egg-dropping-puzzle/

https://leetcode.com/discuss/56350/straight-forward-c-solution-with-explaination

When there is an array first think about length . What happens when it has odd length or even length or null

A subarray which has sum equal to 0. SUBARRAY sum = 0

```
public void findSumZero(int [] arr){

        int sumArray [] = new int[arr.length];

        sumArray[0] =  arr[0];

        for(int i=1 ; i<arr.length ; i++){
            sumArray[i] = sumArray[i-1] + arr[i];
        }

        for(int i=0;i<sumArray.length;i++){
            for(int j = i+1; j<sumArray.length;j++){
                if(sumArray[i] == sumArray[j] || sumArray[j] == 0){
                   print(i,j,arr);
                }
            }
        }
    }
```

This algorithm will find all subarrays with sum 0 and it can be easily modified to find the minimal one or to keep track of the start and end indexes. This algorithm is O(n).

Given an int[] input array, you can create an int[] tmp array where tmp[i] = tmp[i - 1] + input[i]; Each element of tmp will store the sum of the input up to that element.

Now if you check tmp, you'll notice that there might be values that are equal to each other. Let's say that this values are at indexes j an k with j < k, then the sum of the input till j is equal to the sum till k and this means that the sum of the portion of the array between j and k is 0! Specifically the 0 sum subarray will be from index j + 1 to k.

Algo:-
let the array be arr
1.Create An array sum as below
sum[i]=arr[0]+arr[1]+...arr[i];

2.Now see for duplicate values in sum array.
for eg:-
the array is 1,2,3,-2,-1,4,-5,1
sum array 1,3,6, 4, 3,7,2,3
so after finding duplicate values you will see the all the elements between their indices will sum upto zero and you can print it.

Number of edges in directed graph = n(n-1)
Undirected graphs = n(n-1)/2

Cycle – Start and end edges are same and no other vertices or edges are repeated
A graph with no cycle is Acyclic graph
A tree is undirected acyclic graph
Space Complexity of graphs = O(no.of edges+no.of vertex)

Topological sort = DFS of graph and represent as linked list
We can perform a topological sort in time *O(V + E)*, since depth-first search
takes *O(V + E)* time and it takes *O(1)* time to insert each of the |*V*| vertices onto
the front of the linked list.

A **Binomial Tree** of order 0 has 1 node. A Binomial Tree of order k can be constructed by taking two binomial trees of order k-1, and making one as leftmost child of other.
A Binomial Tree of order k has following properties.
a) It has exactly 2^k nodes.
b) It has depth as k.
c) There are exactly kCi nodes at depth i for i = 0, 1, . . . , k.
d) The root has degree k and children of root are themselves Binomial Trees with order k-1, k-2,.. 0 from left to right.

For a given Fibonacci heap H, we
indicate by t (H) the number of trees in the root list of H and by m(H) the number
of marked nodes in H. The potential of Fibonacci heap H is then defined by
(H) = t (H) + 2m(H) .

Kruskals
MST-KRUSKAL*(G,w)*
1 *A* ← ∅
2 **for** each vertex *v* ∈ *V*[*G*]
3 **do** MAKE-SET*(v)*
4 sort the edges of *E* into nondecreasing order by weight *w*
5 **for** each edge *(u, v)* ∈ *E*, taken in nondecreasing order by weight
6 **do if** FIND-SET*(u)* != FIND-SET*(v)*
7 **then** *A* ← *A* ∪ {*(u, v)*}
8 UNION*(u, v)*
9 **return** *A*

SC = O(E+V)
TC = O(ELogE)

https://github.com/mission-peace/interview/blob/master/src/com/interview/graph/KruskalMST.java

Prims Algorithm is Greedy because tree is augmented at each step with an edge that contributes the minimum amount possible to the tree's weight.

MST-PRIM*(G,w, r)*
1 **for** each $u \in V[G]$
2 **do** $key[u] \leftarrow \infty$
3 $\pi[u] \leftarrow$ NIL -> Parent
4 $key[r] \leftarrow 0$
5 $Q \leftarrow V[G]$
6 **while** Q != ∅
7 **do** $u \leftarrow$ EXTRACT-MIN*(Q)*
8 **for** each $v \in Adj[u]$
9 **do if** $v \in Q$ and $w(u, v) < key[v]$
10 **then** $\pi[v] \leftarrow u$
11 $key[v] \leftarrow w(u, v)$

In Prim's, you always keep a connected component, starting with a single vertex. You look at all edges from the current component to other vertices and find the smallest among them. You then add the neighbouring vertex to the component, increasing its size by 1. In N-1 steps, every vertex would be merged to the current one if we have a connected graph.
In Kruskal's, you do not keep one connected component but a forest. At each stage, you look at the globally smallest edge that does not create a cycle in the current forest. Such an edge has to necessarily merge two trees in the current forest into one. Since you start with N single-vertex trees, in N-1 steps, they would all have merged into one if the graph was connected.

Djikstras
No negative edge lengths allowed in Djikstras algorithm. Directed weighted graph

```
for each vertex v ∈ V[G]
       do d[v]←∞
       π[v]← NIL
d[s] ← 0
S ← ∅
Q ← V[G]
while Q != ∅
       do u ← EXTRACT-MIN(Q)
       S ← S ∪ {u}
       for each vertex v ∈ Adj[u]
              if d[v] > d[u] + w(u, v)
                     then d[v] ← d[u] + w(u, v)
                     π[v]← u
```

O(ELogV)=OE+vlogv

Naive String Matching Algorithm

```
    /* A loop to slide pat[] one by one */
    for (int i = 0; i <= N - M; i++)
    {
        int j;

        /* For current index i, check for pattern match */
        for (j = 0; j < M; j++)
```

```
            if (txt[i+j] != pat[j])
                break;

        if (j == M)  // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
           printf("Pattern found at index %d \n", i);
```

Inversion array –
http://www.geeksforgeeks.org/counting-inversions/

if a[i] > a[j] then all values from a[i+1] to a[mid] are greater than a[j] so count = count+mid-i

https://github.com/mission-peace/interview/blob/master/src/com/interview/dynamic/LongestIncreasingSubsequence.java
https://github.com/mission-peace/interview/blob/master/src/com/interview/array/LongestIncreasingSubSequenceOlogNMethod.java

If longer then insert , if smaller binary search to insert

Travelling Salesman problem in NP complete .
Brute Force - O(n!)
Dynamic programming algorithms: $O(n^2 2^n)$

Task Scheduling Job

Schedule tasks based on earliest finish time. Remove overlapping sets. – O(nlogn)
https://www.youtube.com/watch?v=6VGXp6RGGJc

Nuts and Bolts Problem
http://www.geeksforgeeks.org/nuts-bolts-problem-lock-key-problem/

Remove Adjacent Pairs
http://www.crazyforcode.com/eliminate-pairs-two-chars-adjacent-string/
Use stack

Minimum Number of Steps Taken to Reach end of the array

```
for(int i=1; i < arr.length; i++){
        for(int j=0; j < i; j++){
            if(arr[j] + j >= i){
                if(jump[i] > jump[j] + 1){
                    result[i] = j;
                    jump[i] = jump[j] + 1;
return jump[n-1]
```

https://github.com/mission-peace/interview/blob/master/src/com/interview/dynamic/MinJumpToReachEnd.java

Anagram of a given string is palindrome or not
http://comproguide.blogspot.com/2013/11/checking-if-any-anagram-of-given-string.html

Articulation Points in Undirected Graph
https://kartikkukreja.wordpress.com/2013/11/09/articulation-points-or-cut-vertices-in-a-graph/

Floyd-Warshall - All pairs shortest path – O(V^3)

```
 n ← rows[W]
 D(0) ← W
 for k ← 1 to n
          do for i ← 1 to n
                   do for j ← 1 to n
```

Dij = min(Dij,Dik+Dkj)

Partition Array into two halves such that sum of differenes in mimimum-

Greedy- O(n^2)

1. Find the neartest neighbor with mimumum distance for I . Mark both as visited
2. If sum of I < sum of J then partition[l++] = max (a[i],a[j]) partition[r++] = min . else vice versa

***Coin Change* Bottom Up –**

Table from video .

For(coins 1-n) T[coin][0] =1
For(coins 1)(sum 1-n) T[1][Sum] = 0

For(coins 1-n)
For(sum 1-n)
If(coins > sum)
T[c][s] = T[c-1][s]
Else T[c][s] = T[c-1][s]+T[c][Sum-Coin]
TC = O(Coins * Sum)

```
for (int i = 1; i <= v.length; i++) {
                    for (int j = 1; j <= amount; j++) {
                        // check if the coin value is less than the amount needed
                        if (v[i - 1] <= j) {
                            // reduce the amount by coin value and
                            // use the subproblem solution (amount-v[i]) and
                            // add the solution from the top to it
                            solution[i][j] = solution[i - 1][j]
                                    + solution[i][j - v[i - 1]];
                        } else {
                            // just copy the value from the top
                            solution[i][j] = solution[i - 1][j];
                        }
                    }
                }
                return solution[v.length][amount];
        }
```

Partitioning Problem is A set can be divided into two sets such that sum of S1 = sum of S2 .
O(nN) N= total sum of array
http://algorithmsandme.in/2014/04/balanced-partition-problem/

```
int T[10240];
int partition(int C[], int size){
    //computethe total sum
    int N= 0;
    for(int i= 0;i<n;i++) N+=C[i];
   //initialize the table
   T[0]=true;
   for(int i=1;i<=N;i++)T[i]= false;


  //process the numbers one by one
  for (int i =0; i<n; i++){
    for(int j = N-C[i]; j>=0; j--){
        if(T[j]) T[j+C[i]]= true;  //  Create a new subset that adds up to N from old subset
    }
  }
  return T[N/2];
}
```

Balanced Partition - two subsets such that you minimize |S1 – S2|,

```
P[i][j] = 1 -> P[i-1][j]=1 or P[i-1][j-Ai]=1
P[i][j] = max (P[i-1][j],P[i-1][j-Ai])
```

Longest Palindrome Substring DP – O(n^2) and O(n^2) space

```
  int maxLength = 1;
    for (int i = 0; i < n; ++i)
        table[i][i] = true;
```

```
    // check for sub-string of length 2.
    int start = 0;
    for (int i = 0; i < n-1; ++i)
    {
        if (str[i] == str[i+1])
        {
            table[i][i+1] = true;
            start = i;
            maxLength = 2;
        }
    }

    // Check for lengths greater than 2. k is length
    // of substring
    for (int k = 3; k <= n; ++k)
    {
        // Fix the starting index
        for (int i = 0; i < n-k+1 ; ++i)
        {
            // Get the ending index of substring from
            // starting index i and length k
            int j = i + k - 1;

            // checking for sub-string from ith index to
            // jth index iff str[i+1] to str[j-1] is a
            // palindrome
            if (table[i+1][j-1] && str[i] == str[j])
            {
                table[i][j] = true;

                if (k > maxLength)
                {
                    start = i;
                    maxLength = k;
                }
            }
        }
    }

    printf("Longest palindrome substring is: ");
    printSubStr( str, start, start + maxLength - 1 );

    return maxLength; // return length of LPS
}
```

Number of times S appears in T

```
S(i) ,T(j)
L(I,j) = Max {L(i-1,j-1)+L(I,j-1) , if S[i]=S[j],
                L(i-1,j) if S[i]!=S[j]
        0 if T=0;
                1 if s=0
```

Cutting Rod

```
Max= min_int
For(i=0->n) // Cuts
For(j=0->i) // Value for cuts
        Max = Math.max(max,valueForTheCut[j]+solution[i-j]) // Value at that cut + value of the cuts
before it get the max)
        Solution[i] = max;
Return solution[n];
```

Weight Scheduling Algorithm

```
For(i=0->n)
        For(j=0->i)
                If(profit[i] < profit[j]+profit[i])
                T[i] = profit[i]
```

Merge Intervals LeetCode

http://www.programcreek.com/2012/12/leetcode-merge-intervals/

LCA

```
struct node* LCA(struct node* root, struct node* p, struct node *q)
{
          struct node *l,*r,*temp;
```

```
        if (root == NULL)  return NULL;
        if (root == p || root == q) return root;
        L = LCA(root->left, p, q);
        R = LCA(root->right, p, q);
        if (L && R) //if both nodes on either side of tree. return root;
        return L ? L : R;  // either both nodes are on same side or not in left and right
subtree.
```

Shortest Word Distances

```
public class Solution {
    public int shortestWordDistance(String[] words, String word1, String
word2) {
        int posA = -1;
        int posB = -1;
        int minDistance = Integer.MAX_VALUE;

        for (int i = 0; i < words.length; i++) {
            String word = words[i];

            if (word.equals(word1)) {
                posA = i;
            } else if (word.equals(word2)) {
                posB = i;
            }

            if (posA != -1 && posB != -1 && posA != posB) {
                minDistance = Math.min(minDistance, Math.abs(posA - posB));
            }

            if (word1.equals(word2)) {
                posB = posA;
            }
        }

        return minDistance;
    }
}
```

Given a sorted and rotated array, find if there is a pair with a given sum

```
l=0;
r=length-1
 while (l != r)
    {
         // If we find a pair with sum x, we return true
         if (arr[l] + arr[r] == x)
             l++;
             r--;
              return true;

         // If current pair sum is less, move to the higher sum
         if (arr[l] + arr[r] < x)
              l + +;
         else  // Move to the lower sum side
              r--;
    }
```

Find Pair of Elements whose difference is k

You can do it in O(n) with a hash table. Put all numbers in the hash for O(n), then go through them all again looking for number[i]+k. Hash table returns "Yes" or "No" in O(1), and you need to go through all numbers, so the total is O(n

http://stackoverflow.com/questions/10450462/find-pair-of-numbers-whose-difference-is-an-input-value-k-in-an-unsorted-array

http://www.geeksforgeeks.org/find-a-pair-with-the-given-difference/
http://codercareer.blogspot.com/2013/02/no-42-three-increasing-elements-in-array.html

```
Subset Sum – Running Time O(sum * N)

I = array index, t = 1 to sum
For(I = 1 – t) s[o][i] = 0
```

If a[i] < t,
S(i,t) = S(i - 1,t - a[i]) OR S(i - 1, t)
Else: S(i, t) = S(i - 1, t)
To print
If(S[i-1][j] == true) print
Else (s[i-1][j-A[i]) if its true print this number

We can check by excluding the ith element in the subset and we check if we can get the sum by j by including ith by checking if the sum j-A[i] exists without the ith element

Combination Sum O(n+k)!

```
public void combinationSum(int[] candidates, int target, int j,
ArrayList<Integer> curr, ArrayList<ArrayList<Integer>> result){
   if(target == 0){
        ArrayList<Integer> temp = new ArrayList<Integer>(curr);
        result.add(temp);
        return;
   }

   for(int i=j; i<candidates.length; i++){
        if(target < candidates[i])
             return;

        curr.add(candidates[i]);
        combinationSum(candidates, target - candidates[i], i, curr, result);
        curr.remove(curr.size()-1);
   }
}
```

Permutation of String using recursion – O(n*n!)

Integer as sum of two non-consecutive Fibonacci Numbers

```
int nearestSmallerEqFib(int n)
{
        // Corner cases
        if (n == 0 || n==1)
        return n;

        // Find the greatest Fibonacci Number smaller
        // than n.
        int f1 = 0, f2 = 1, f3 = 1;
        while (f3 <= n)
        {
                f1 = f2;
                f2 = f3;
                f3 = f1+f2;
        }
        return f2;
}
// Prints Fibonacci Representation of n using
// greedy algorithm
void printFibRepresntation(int n)
```

```
{
        while (n>0)
        {
                // Find the greates Fibonacci Number smaller
                // than or equal to n
                int f = nearestSmallerEqFib(n);
                // Print the found fibonacci number
                cout << f << " ";
                // Reduce n
                n = n-f;
        }
}
```

Buy and Sell Stocks iii. At most two transactions

```
public int maxProfit(int[] prices) {
        if (prices == null || prices.length < 2) {
                return 0;
        }

        //highest profit in 0 ... i
        int[] left = new int[prices.length];
        int[] right = new int[prices.length];
        // DP from left to right
        left[0] = 0;
        int min = prices[0];
        for (int i = 1; i < prices.length; i++) {
                min = Math.min(min, prices[i]);
                left[i] = Math.max(left[i - 1], prices[i] - min);
        }
        // DP from right to left
        right[prices.length - 1] = 0;
        int max = prices[prices.length - 1];
        for (int i = prices.length - 2; i >= 0; i--) {
                max = Math.max(max, prices[i]);
                right[i] = Math.max(right[i + 1], max - prices[i]);
        }
        int profit = 0;
        for (int i = 0; i < prices.length; i++) {
                profit = Math.max(profit, left[i] + right[i]);
        }

        return profit;} https://gist.github.com/Jeffwan/9908858
Best Time to Buy Stocks with cooldown-
https://leetcode.com/discuss/71354/share-my-thinking-process
public int maxProfit(int[] prices) {
    int sell = 0, prev_sell = 0, buy = Integer.MIN_VALUE, prev_buy;
    for (int price : prices) {
        prev_buy = buy;
        buy = Math.max(prev_sell - price, prev_buy);
        prev_sell = sell;
        sell = Math.max(prev_buy + price, prev_sell);
    }
    return sell;
}
```

Wiggle Sort

A[0] <= A[1] >= A[2] <= A[3] >= A[4] <= A[5]
So we could actually observe that there is pattern that

A[even] <= A[odd],
A[odd] >= A[even].

```
public class Solution {
    public void wiggleSort(int[] nums) {
        if (nums == null || nums.length <= 1) {
            return;
        }
        for (int i = 0; i < nums.length - 1; i++) {
            if (i % 2 == 0) {
                if (nums[i] > nums[i + 1]) {
                    swap(nums, i, i + 1);
                }
            } else {
                if (nums[i] < nums[i + 1]) {
                    swap(nums, i, i + 1);
```

http://algorithms.tutorialhorizon.com/construct-a-binary-tree-from-given-inorder-and-postorder-traversal/

BST to DLL

```
static node* prev = NULL;
void BinaryTree2DoubleLinkedList(node *root, node **head)
{
    // Base case
    if (root == NULL) return;
 // Recursively convert left subtree
    BinaryTree2DoubleLinkedList(root->left, head);

    // Now convert this node
    if (prev == NULL)
        *head = root;
    else
    {
        root->left = prev;
        prev->right = root;
    }
    prev = root;

    // Finally convert right subtree
    BinaryTree2DoubleLinkedList(root->right, head);
}
```

O(n) Time Complexity

Longest Common Substring

```
   public int longestCommonSubstring(char str1[], char str2[])
 {
         int T[][] = new int[str1.length+1][str2.length+1];
         int max = 0;
         for(int i=1; i <= str1.length; i++){
             for(int j=1; j <= str2.length; j++){
                 if(str1[i-1] == str2[j-1]){
                     T[i][j] = T[i-1][j-1] +1;
                     if(max < T[i][j]){
                         max = T[i][j];
                     }
                 }
             }

         }
```

http://introcs.cs.princeton.edu/java/42sort/LRS.java.html
Here's a simple algorithm for building a suffix array:
● Construct all the suffixes of the string in time Θ(m2).
● Sort those suffixes using heapsort or mergesort. – Makes O(m log m) comparisons, but each comparison takes O(m) time. – Time required: O(m2 log m).
● Total time: O(m^2 log m)
http://www.geeksforgeeks.org/suffix-array-set-1-introduction/

Median from Stream

```
Step 1: Add next item to one of the heaps

   if next item is smaller than maxHeap root add it to maxHeap,
   else add it to minHeap

Step 2: Balance the heaps (after this step heaps will be either balanced or
   one of them will contain 1 more item)

   if number of elements in one of the heaps is greater than the other by
   more than 1, remove the root element from the one containing more elements and
   add to the other one
```

Then at any given time you can calculate median like this:

```
   If the heaps contain equal elements;
     median = (root of maxHeap + root of minHeap)/2
   Else
     median = root of the heap with more elements
```

Binary Without Consecutive Ones

```
Public class CountNumberOfBinaryWithoutConsecutive1s {

    public int count(int n){
        int a[] = new int[n];
        int b[] = new int[n];

        a[0] = 1;
        b[0] = 1;

        for(int i=1; i < n; i++){
            a[i] = a[i-1] + b[i-1];
            b[i] = a[i-1];
        }

        return a[n-1] + b[n-1];
    }
```

Find Number of Islands

```
public class Solution {
    public int numIslands(char[][] grid) {
        if (grid == null || grid.length == 0) {
            return 0;
        }
        int result = 0;
        boolean[][] visited = new boolean[grid.length][grid[0].length];
        for (int i = 0; i < grid.length; i++) {
            for (int j = 0; j < grid[0].length; j++) {
                if (!visited[i][j] && grid[i][j] == '1') {
```

```
                    dfs(grid, visited, i, j);
                    result++;
                }
            }
         return result
    private void dfs(char[][] grid, boolean[][] visited, int i, int j) {
        if (i < 0 || i >= grid.length || j < 0 || j >= grid[0].length || visited[i][j] || grid[i][j]
== '0') {
            return;
        }
        visited[i][j] = true;
        dfs(grid, visited, i - 1, j);
        dfs(grid, visited, i + 1, j);
        dfs(grid, visited, i, j - 1);
        dfs(grid, visited, i, j + 1);
    }
}
```

Add two numbers without +

```
int add_no_arithm(int a, int b) {
 if (b == 0) return a;
 int sum = a ^ b; // add without carrying
 int carry = (a & b) << 1; // carry, but don't add
 return add_no_arithm(sum, carry); // recurse
}
```

Edit Distance

```
public int dynamicEditDistance(char[] str1, char[] str2){
        int temp[][] = new int[str1.length+1][str2.length+1];

        for(int i=0; i < temp[0].length; i++){
            temp[0][i] = i;
        }

        for(int i=0; i < temp.length; i++){
            temp[i][0] = i;
        }

        for(int i=1;i <=str1.length; i++){
            for(int j=1; j <= str2.length; j++){
                if(str1[i-1] == str2[j-1]){
                    temp[i][j] = temp[i-1][j-1];
                }else{
                    temp[i][j] = min(temp[i-1][j-1] + 1,temp[i-1][j] +
EDIT_COST,temp[i][j-1] + EDIT_COST);
                }
            }
        }
        return temp[str1.length][str2.length];

    }
```

http://www.geeksforgeeks.org/dynamic-programming-set-32-word-break-problem/

Find Minimum Window in S with all elements in T

```
bool minWindow(const char* S, const char *T,
               int &minWindowBegin, int &minWindowEnd) {
  int sLen = strlen(S);
  int tLen = strlen(T);

int needToFind[256] = {0};

  for (int i = 0; i < tLen; i++)
    needToFind[T[i]]++;

  int hasFound[256] = {0};
  int minWindowLen = INT_MAX;
  int count = 0;
  for (int begin = 0, end = 0; end < sLen; end++) {
    // skip characters not in T
    if (needToFind[S[end]] == 0) continue;
    hasFound[S[end]]++;
    if (hasFound[S[end]] <= needToFind[S[end]])
      count++;

    // if window constraint is satisfied
    if (count == tLen) {
      // advance begin index as far right as possible,
      // stop when advancing breaks window constraint.
      while (needToFind[S[begin]] == 0 ||
             hasFound[S[begin]] > needToFind[S[begin]]) {
        if (hasFound[S[begin]] > needToFind[S[begin]])
          hasFound[S[begin]]--;
        begin++;
      }

      // update minWindow if a minimum length is met
      int windowLen = end - begin + 1;
      if (windowLen < minWindowLen) {
        minWindowBegin = begin;
        minWindowEnd = end;
        minWindowLen = windowLen;
      } // end if
    } // end if
  } // end for

  return (count == tLen) ? true : false;
}
```

Maximum Length of Substring without Repetition

```
int lengthOfLongestSubstring(string s) {
  int n = s.length();
  int i = 0, j = 0;
  int maxLen = 0;
  bool exist[256] = { false };
  while (j < n) {
    if (exist[s[j]]) {
```

```
    maxLen = max(maxLen, j-i);
    while (s[i] != s[j]) {
      exist[s[i]] = false;
      i++;
    }
    i++;
    j++;
   } else {
    exist[s[j]] = true;
    j++;
   }
 }
 maxLen = max(maxLen, n-i);
 return maxLen;
}
```

Find Next Palindrome

```
void generateNextPalindromeUtil (int num[], int n )
{
    // find the index of mid digit
    int mid = n/2;

    // A bool variable to check if copy of left side to right is sufficient or not
    bool leftsmaller = false;

    // end of left side is always 'mid -1'
    int i = mid - 1;

    // Begining of right side depends if n is odd or even
    int j = (n % 2)? mid + 1 : mid;

   // Initially, ignore the middle same digits
    while (i >= 0 && num[i] == num[j])
        i--,j++;
    // Find if the middle digit(s) need to be incremented or not (or copying left
    // side is not sufficient)
    if ( i < 0 || num[i] < num[j])
        leftsmaller = true;
    // Copy the mirror of left to tight
    while (i >= 0)
    {
        num[j] = num[i];
        j++;
        i--;
    }
    // Handle the case where middle digit(s) must be incremented.
    // This part of code is for CASE 1 and CASE 2.2
    if (leftsmaller == true){
        int carry = 1;
        i = mid - 1;
        // If there are odd digits, then increment
        // the middle digit and store the carry
        if (n%2 == 1)
        {
            num[mid] += carry;
            carry = num[mid] / 10;
            num[mid] %= 10;
            j = mid + 1;
        }
        else
            j = mid;
```

```
        // Add 1 to the rightmost digit of the left side, propagate the carry
        // towards MSB digit and simultaneously copying mirror of the left side
        // to the right side.
        while (i >= 0)
        {
            num[i] += carry;
            carry = num[i] / 10;
            num[i] %= 10;
            num[j++] = num[i--]; // copy mirror to right
        }
    }
```

Longest Palindrome Substring Manachers Algorithm
http://articles.leetcode.com/longest-palindromic-substring-part-i/

```
string preProcess(string s) {
  int n = s.length();
  if (n == 0) return "^$";
  string ret = "^";
  for (int i = 0; i < n; i++)
    ret += "#" + s.substr(i, 1);

  ret += "#$";
  return ret;
}

string longestPalindrome(string s) {
  string T = preProcess(s);
  int n = T.length();-
  int *P = new int[n];
  int C = 0, R = 0;
  for (int i = 1; i < n-1; i++) {
    int i_mirror = 2*C-i; // equals to i' = C - (i-C)

    P[i] = (R > i) ? min(R-i, P[i_mirror]) : 0;

    // Attempt to expand palindrome centered at i
    while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
      P[i]++;

    // If palindrome centered at i expand past R,
    // adjust center based on expanded palindrome.
    if (i + P[i] > R) {
      C = i;
      R = i + P[i];
    }
  }

  // Find the maximum element in P.
  int maxLen = 0;
  int centerIndex = 0;
  for (int i = 1; i < n-1; i++) {
    if (P[i] > maxLen) {
      maxLen = P[i];
      centerIndex = i;
    }
```

```
  }
  delete[] P;

  return s.substr((centerIndex - 1 - maxLen)/2, maxLen);
}
```

Palindrome Partition

```
public static List<String> palindromePartitioning(String s) {
          List<String> result = new ArrayList<String>();
          if (s == null)
                    return result;
          if (s.length() <= 1) {
                    result.add(s);
                    return result;
          }
          int length = s.length();
          int[][] table = new int[length][length];
          // l is length, i is index of left boundary, j is index of right boundary
          for (int l = 1; l <= length; l++) {
                    for (int i = 0; i <= length - l; i++) {
                              int j = i + l - 1;
                              if (s.charAt(i) == s.charAt(j)) {
                                        if (l == 1 || l == 2) {
                                                  table[i][j] = 1;
                                        } else {
                                                  table[i][j] = table[i + 1][j - 1];
                                        }
                                        if (table[i][j] == 1) {
                                                  result.add(s.substring(i, j + 1));
                                        }
                              } else {
                                        table[i][j] = 0;
                              }}}

          return result;
}
```

–

Topological Sort O(V+E)-
https://github.com/IDeserve/learn/blob/master/topologicalSort.java

Shortest Path in DAG – First topological sort and then Djikstras algorithm

Time Complexity of Permutation of a String using Recursion O(n*n!)
Rearrgange String without repeating elements next to each other

Aaabc – abaca

1. Create character count
2. If count of an element is greater than n/2+1 then return
3. Iterate over each element and take its count
4. J=0 – If j> str.length j=1
5. output[j]=string[i] j=j+2;

http://js-interview.tutorialhorizon.com/2015/10/19/rearrange-characters-in-a-string-so-that-no-character-repeats-twice/

```
function createFrequencyAnalysis(str) {
    var charFrequency = {};

    Array.prototype.forEach.call(str, function(ch) {
        charFrequency[ch] ? charFrequency[ch]++ : charFrequency[ch] = 1;
    });
    return charFrequency;
}

function reArrange (str) {

  var freqMap = createFrequencyAnalysis(str);

  var strLength = str.length;
```

```
  var characters = Object.keys(freqMap);
  var output = new Array(strLength);

  var i = 0;
  var j = 0;
  var k = 0;
  var charFreq = 0;
  var numberOfChar = characters.length;

  for (i = 0; i < numberOfChar; i++) {
    charFreq = freqMap[characters[i]];

    if (charFreq >= ((strLength / 2 ) + 1)) {
      console.log('No valid output');
      return;
    }

    for (k = 0; k < charFreq; k++) {
      // If reached at the end of an array, wrap the array and start from the begining with odd (1, 3,
5 ...) indexes
      if (j > strLength) {
        j = 1;
      }

      output[j] = characters[i];
      j += 2;
    }
  }
  return output.join('');
}
```

Find Number of 1s in a Binary Representation of a Number

```
  int  rst = 0 ;
  while (n != 0) {
       n = n & (n - 1);
       rst++;}
```

Longest Substring with K unique elements

https://ideone.com/JRpJqc

Max Sum from non adjacent elements

```
public int maxsumNonContiguous(int[] array){
            /* Empty array */
            if(array == null)
               return 0;

            int arrayLength = array.length;
            /* Current maximum sum including the current value */
            int sum1 = array[0];
            /* Current maximum sum excluding the current value */
            int sum2 = 0;
            for(int i = 1; i  <  arrayLength; i++){

                /* Current maximum sum excluding the current index value */
               int sum3 = Math.max(sum1,sum2);
                /* Current maximum sum including the current index value */
                sum1 = sum2 + array[i];
              /* Move the value of sum3 into sum2 */
                sum2 = sum3;
            }

            /* Return the maximum of sum1 and sum2 */
            return Math.max(sum1, sum2);
        }
```

O(n)-Tc and SC- O(1)

Combination Sum – O(n+k)!

```
public void getCombination(int[] num, int start, int target, ArrayList<Integer> temp,
ArrayList<ArrayList<Integer>> result){
    if(target == 0){
        ArrayList<Integer> t = new ArrayList<Integer>(temp);
        result.add(t);
        return;
    }
    for(int i=start; i<num.length; i++){
        if(target < num[i])
            continue;

        temp.add(num[i]);
        getCombination(num, i+1, target-num[i], temp, result);
        temp.remove(temp.size()-1);
    }
}
```

Matrix in Spiral Order

```
 while (k < m && l < n)
    {
        /* Print the first row from the remaining rows */
        for (i = l; i < n; ++i)
        {
            printf("%d ", a[k][i]);
        }
        k++;

        /* Print the last column from the remaining columns */
        for (i = k; i < m; ++i)
        {
            printf("%d ", a[i][n-1]);
        }
        n--;

        /* Print the last row from the remaining rows */
        if ( k < m)
        {
            for (i = n-1; i >= l; --i)
            {
                printf("%d ", a[m-1][i]);
            }
            m--;
        }

        /* Print the first column from the remaining columns */
        if (l < n)
        {
            for (i = m-1; i >= k; --i)
            {
                printf("%d ", a[i][l]);
            }
            l++;
        }

}
```

Given an array of size n. It contains numbers in the range 1 to N. Each number is present at least once except for 2 numbers. Find the missing numbers.

Write a program to remove duplicate elements from array

```
for(int i=0;i<b.length;i++) {
                        if(b[Math.abs(b[i])] >= 0) {
                                b[Math.abs(b[i])] = -b[Math.abs(b[i])];
                        } else {
```

```
                            System.out.println(Math.abs(b[i]));
                        }
                }
```

Find the Occurrence in O(n) tc and O(1) space complexity

```
// Function to find counts of all elements present in
// arr[0..n-1]. The array elements must be range from
// 1 to n
void printfrequency(int arr[],int n)
{
    // Subtract 1 from every element so that the elements
    // become in range from 0 to n-1
    for (int j =0; j<n; j++)
        arr[j] = arr[j]-1;
     // Use every element arr[i] as index and add 'n' to
    // element present at arr[i]%n to keep track of count of
    // occurrences of arr[i]
    for (int i=0; i<n; i++)
        arr[arr[i]%n] = arr[arr[i]%n] + n;
    // To print counts, simply print the number of times n
    // was added at index corresponding to every element
    for (int i =0; i<n; i++)
        cout << i + 1 << " ->  " << arr[i]/n << endl;
}
```

Max Repeating element

```
int maxRepeating(int* arr, int n, int k)
{
    // Iterate though input array, for every element
    // arr[i], increment arr[arr[i]%k] by k
    for (int i = 0; i< n; i++)
        arr[arr[i]%k] += k;
    // Find index of the maximum repeating element
    int max = arr[0], result = 0;
    for (int i = 1; i < n; i++)
    {
        if (arr[i] > max)
        {
            max = arr[i];
            result = i;
        }
    }
    /* Uncomment this code to get the original array back
       for (int i = 0; i< n; i++)
          arr[i] = arr[i]%k; */
    // Return index of the maximum element
    return result;
}
```

Quick Sort SC = O(logn) Because stack frame is created by reducing the search space based on index

BST from Preorder

```
   for ( i = 1; i < size; ++i )
    {
        temp = NULL;

        /* Keep on popping while the next value is greater than
           stack's top value. */
        while ( !isEmpty( stack ) && pre[i] > peek( stack )->data )
            temp = pop( stack );
```

```
        // Make this greater value as the right child and push it to the stack
        if ( temp != NULL)
        {
            temp->right = newNode( pre[i] );
            push( stack, temp->right );
        }
        // If the next value is less than the stack's top value, make this value
        // as the left child of the stack's top node. Push the new node to stack
        else   {
            peek( stack )->left = newNode( pre[i] );
            push( stack, peek( stack )->left );
        }
    }
    return root;
```

Kth smallest element in two sorted arrays –O(log m+n)
http://algorithmsandme.in/2014/12/find-kth-smallest-element-in-two-sorted-arrays/

Shortest subarray to be sorted – sub sort

```
public void sort(int a[]) {
    int low = 0;
    int high = a.length-1;
    int start = 0;
    int end = 0;
    while(low < high) {
        if(a[low] > a[low+1]) {
            start = low;
            break;
        }
        low++;
    }
    // if i=n-1 return -1 entire array is sorted
    while(high > 0) {
        if(a[high] < a[high-1]) {
            end = high;
        }
        high --;
    }
    int max = a[start]=min;
    for(int i=start;i<end;i++) {
        if(a[i] > max) max = a[i];
        if(a[i] < min) min = a[i];
    }
    for(int i=0;i<start;i++) {
        if(a[i] > min) start = i;
    }
    for(int i=end ;i<n;i++) {
        if(a[i] < max) end = i;
    }
}
```

Maximum Sliding Window

```
public ArrayList<Integer> maxWindow(int a[],int k) {
    ArrayList<Integer> list = new ArrayList<Integer>();
```

```
    Queue q = new Queue();

    for(int i=0;i<k;i++) {
        while(!q.isEmpty && a[i] > q.back()) {
            q.pop_back();
        }
        q.push_back(i);
    }
    for(int i=k;i <a.length;i++) {
        list.add(a[q.front()]);
        while(!q.isEmpty && a[i] > q.back()) {
            q.pop_back();
        }
        if(!q.isEmpty && q.front() <=i-k) {
            q.pop_front();
        }
        q.push_back(i);
    }
    list.add(a[q.front()]);
    return list;
}
```

Remove Duplicates in Array

```
public void removeDups(int str[]) {
        //4,7,7,1,4,8,5
        int len = str.length;
        if (len < 2 || len == 0) return;
        int tail = 1;
        for (int i = 1; i < len; i++) {
            int j;
            for (j = 0; j < tail; j++) {
                if (str[i] == str[j]) break;
            }
            if (j == tail) {
                str[tail] = str[i];
                tail++;
            }
        }
        for(int i=tail;i<len;i++) {
                str[i] = 0;
        }
    }
```

Three Stacks from an array
Index = stackNum * stackSize + stackPointer[stackNum]
Pg 117

Sort a stack –O(n^2)

```
public static Stack<Integer> sortStack(Stack<Integer> s) {
    Stack<Integer> r = new Stack<Integer>();
    while(!s.isEmpty()) {
        int temp = s.pop();
        while(!r.isEmpty() && r.peek() > temp) {
            s.push(r.pop());
        }
        r.push(temp);
    }
    return r’
```

http://www.geeksforgeeks.org/find-index-0-replaced-1-get-longest-continuous-sequence-1s-binary-array/

```
int maxOnesIndex(bool arr[], int n)
{
```

```
    int max_count = 0;  // for maximum number of 1 around a zero
    int max_index;  // for storing result
    int prev_zero = -1;  // index of previous zero
    int prev_prev_zero = -1; // index of previous to previous zero

    // Traverse the input array
    for (int curr=0; curr<n; ++curr)
    {
        // If current element is 0, then calculate the difference
        // between curr and prev_prev_zero
        if (arr[curr] == 0)
        {
            // Update result if count of 1s around prev_zero is more
            if (curr - prev_prev_zero > max_count)
            {
                max_count = curr - prev_prev_zero;
                max_index = prev_zero;
            }

            // Update for next iteration
            prev_prev_zero = prev_zero;
            prev_zero = curr;
        }
    }

    // Check for the last encountered zero
    if (n-prev_prev_zero > max_count)
       max_index = prev_zero;
    return max_index;
}
```

http://www.geeksforgeeks.org/find-zeroes-to-be-flipped-so-that-number-of-consecutive-1s-is-maximized/

Kth smallest element in two sorted arrays -O(log m+n)
http://algorithmsandme.in/2014/12/find-kth-smallest-element-in-two-sorted-arrays/

Median of two sorted arrays
http://www.geeksforgeeks.org/median-of-two-sorted-arrays/

http://www.geeksforgeeks.org/largest-subarray-with-equal-number-of-0s-and-1s/

InOrder and PreOrder to BT – O(n^2)

```
struct node* buildTree(char in[], char pre[], int inStrt, int inEnd)
{
  static int preIndex = 0;

  if(inStrt > inEnd)
     return NULL;

  /* Pick current node from Preorder traversal using preIndex
    and increment preIndex */
  struct node *tNode = newNode(pre[preIndex++]);

  /* If this node has no children then return */
  if(inStrt == inEnd)
    return tNode;

  /* Else find the index of this node in Inorder traversal */
  int inIndex = search(in, inStrt, inEnd, tNode->data);
   /* Using index in Inorder traversal, construct left and
     right subtress */
  tNode->left = buildTree(in, pre, inStrt, inIndex-1);
  tNode->right = buildTree(in, pre, inIndex+1, inEnd);
```

```
-
  return tNode;
```

Find existence of a path with a given sum :O(n) TC and sc

Problem-21 Give an algorithm for checking the existence of path with given sum. That means, given a sum check whether there exists a path from root to any of the nodes.

Solution: For this problem, the strategy is: subtract the node value from the sum before calling its children recursively, and check to see if the sum is 0 when we run out of tree.

```
int HasPathSum(struct BinaryTreeNode * root, int sum) {
        // return true if we run out of tree and sum==0
        if(root == NULL) return(sum == 0);
         else {      // otherwise check both subtrees
                  int remainingSum = sum - root→data;
                  if((root→left && root→right)||(!root→left && !root→right))
                        return(HasPathSum(root→left, remainingSum) || HasPathSum(root→right, remainingSum));
                  else if(root→left)
                           return HasPathSum(root→left, remainingSum);
                  else     return HasPathSum(root→right, remainingSum);
          }
```

```
int hasPathSum(struct node* node, int sum) {
  // return true if we run out of tree and sum==0
  if (node == NULL) {
    return(sum == 0);
  }
  else {
  // otherwise check both subtrees
    int subSum = sum - node->data;
    return(hasPathSum(node->left, subSum) ||
           hasPathSum(node->right, subSum));
  }
}
```

Odd Even Linked List

```
public ListNode oddEvenList(ListNode head) {
    if(head==null||head.next==null) return head;
    ListNode odd=head,ehead=head.next,even=ehead;
    while(even!=null&&even.next!=null){
        odd.next=even.next;
        odd=odd.next;
        even.next=odd.next;
        even=even.next;
    }
    odd.next=ehead;
    return head;
}
```

The Prime numbers are between 6x+1 and 6x-1

Finding All Combinations of Brackets

```
   public void Brackets(int n)
    {
        for (int i = 1; i <= n; i++)
        {
            Brackets("", 0, 0, i);
        }
    }
    private void Brackets(string output, int open, int close, int pairs)
```

```
    {
        if((open==pairs)&&(close==pairs))
        {
            Console.WriteLine(output);
        }
        else
        {
            if(open<pairs)
                Brackets(output + "(", open+1, close, pairs);
            if(close<open)
                Brackets(output + ")", open, close+1, pairs);
        }
    }
```

Find Common elements in n sorted arrays
http://www.ideserve.co.in/learn/find-common-elements-in-n-sorted-arrays

Find square root of a number
http://www.ideserve.co.in/learn/square-root-of-a-number
Just use binary search

Maximum sum of non-adjacent elements

```
Int inc = a[0];
Int exc = 0;
for (int i = 1; i < a.length; i++) {
  int old_inc = inc;
  inc = exc + a[i];
  exc = Math.max(exc, old_inc);
}
return Math.max(inc, exc);
```

https://www.youtube.com/watch?v=1_Lz_EvA_hs

1<<x Power of 2
x>>1 divide by power of 2

Find number of consecutive 1s in a binary representation
Fins number of non-consecutive 1s and total number of possible binary presentations
1<<x – a[n-1]-b[n-1]
http://www.geeksforgeeks.org/count-strings-with-consecutive-1s/

Single Number III

```
public class Solution {
    public int[] singleNumber(int[] nums) {
        int A = 0;
        int B = 0;
        int AXORB = 0;
        for(int i = 0; i<nums.length; i++){
            AXORB ^= nums[i];
        }

        AXORB = (AXORB & (AXORB - 1)) ^ AXORB; //find the different bit
        for(int i = 0; i<nums.length; i++){
            if((AXORB & nums[i]) == 0)
                A ^= nums[i];
            else
                B ^= nums[i];
        }
        return new int[]{A, B};
    }
}
```

http://traceformula.blogspot.com/2015/09/single-number-iii-leetcode.html

Find the non-repeating number where each number repeats 3 times
http://traceformula.blogspot.com/2015/08/single-number-ii-how-to-come-up-with.html

Deadlock
http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html

Variables that are marked as volatile get only visible to other threads once the constructor of the object has finished its execution completely.

Thread.join() – Wait for the thread to die before continuing execution.
A field may be declared volatile, in which case the Java Memory Model ensures that all threads see a consistent value for the variable

The Java `volatile` keyword is used to mark a Java variable as "being stored in main memory". More precisely that means, that every read of a volatile variable will be read from the computer's main memory, and not from the CPU cache, and that every write to a volatile variable will be written to main memory, and not just to the CPU cache.

Volatile variable is used to notify the latest value of the variable to other threads when it is updated by one thread.

Synchronized – only a single thread can access or execute a block. The object is locked to prevent inconsistent state.

LRU Cache –

http://www.programcreek.com/2013/03/leetcode-lru-cache-java/

IsPowerOfThree

```
public class Solution {
    public boolean isPowerOfThree(int n) {
        if (n < 1) {
            return false;
        }
        while (n % 3 == 0) {
            n /= 3;
        }
       return n == 1;
}
```

http://www.programcreek.com/2014/02/leetcode-longest-common-prefix-java/

Contains Duplicate

Given an array of integers and an integer *k*, find out whether there are two distinct indices *i* and *j* in the array such that nums[i] = nums[j]and the difference between *i* and *j* is at most *k*.

```
public boolean containsNearbyDuplicate(int[] nums, int k) {
        Set<Integer> set = new HashSet<Integer>();
        for(int i = 0; i < nums.length; i++){
            if(i > k) set.remove(nums[i-k-1]);
            if(!set.add(nums[i])) return true;
        }
        return false;
```

Check if number is a Palindrome

```
public boolean isPalindrome(int x) {
   if (x<0 || (x!=0 && x%10==0)) return false;
   int rev = 0;
   while (x>rev){
           rev = rev*10 + x%10;
           x = x/10;
   }
   return (x==rev || x==rev/10);
}
```

Reverse = reverse * 10 + x % 10;

Best Time to Buy and Sell Stocks 1

Keep a track of min and profit for every element and take maxProfit

```
public class Solution {
   public int maxProfit(int[] prices) {
        if (prices.length == 0 ) {
                        return 0;
                }
                int profit = 0;
                int maxProfit = 0;
                int min = prices[0];
                for (int i = 1; i < prices.length; i++) {
                        if (prices[i] < min) {
                                min = prices[i];
                        }
                        if (prices[i] > min) {
                                profit = prices[i] - min;
                        }
                        if (profit > maxProfit) {
                                maxProfit = profit;
                        }
   }
   return maxProfit;
 }
}
```

Best Time to Buy and Sell Stock ii

```
int maxprofit = 0;

      for (int i = 1; i < prices.length; i++) {

         if (prices[i] > prices[i - 1])
```

```
            maxprofit += prices[i] - prices[i - 1];

        }

        return maxprofit;
```

```
/checks whether an int is prime or not.
boolean isPrime(int n) {
    //check if n is a multiple of 2
    if (n%2==0) return false;
    //if not, then just check the odds
    for(int i=3;i*i<=n;i+=2) {
        if(n%i==0)
            return false;
    }
    return true;
}
```

BackTracking

https://discuss.leetcode.com/topic/46161/a-general-approach-to-backtracking-questions-in-java-subsets-permutations-combination-sum-palindrome-partitioning

http://www.geeksforgeeks.org/find-common-elements-three-sorted-arrays/

Elimination Game

```
public int lastRemaining(int n) {
        if (n == 0) {
            return 0;
        }
        if (n == 1) {
            return 1;
        }
        int step = 1;
        int head = 1;
        int remaining = n;
        boolean left = true;
        while (remaining > 1) {
            if (left || remaining % 2 ==1) {
                head = head + step;
            }
            remaining = remaining/2;
            step *= 2;
            left = !left;
         }        return head;
```

Jump Game

```
public boolean canJump(int[] nums) {
    int maxLocation = 0;
    for(int i=0; i<nums.length; i++) {
        if(maxLocation < i) return false; // if previous maxLocation smaller than i, meaning we cannot reach location i, thus return false.
```

```
    maxLocation = (i+nums[i]) > maxLocation ? i+nums[i] : maxLocation; // greedy:
  }
  return true;
}
```

Remove K digits to create a smaller number
Ex- 1432219 → 1219
Use a stack. While a[i] < stack.top() , stack.pop()

http://www.programcreek.com/2013/12/leetcode-solution-of-longest-palindromic-substring-java/

http://www.ideserve.co.in/learn/print-matrix-diagonally

```
       for (int k = 0; k < rowCount; k++) {
            for (row = k, col = 0; row >= 0 && col < columnCount; row--, col++) {
                System.out.print(matrix[row][col] + " ");
            }
            System.out.println();
        }
        for (int k = 1; k < columnCount; k++) {
            for (row = rowCount - 1, col = k; row >= 0 && col < columnCount; row--, col++) {
                System.out.print(matrix[row][col] + " ");
            }
            System.out.println();
        }
```

http://www.ideserve.co.in/learn/how-to-recover-a-binary-search-tree-if-two-nodes-are-swapped
While doing inorder traversal keep a track of prev. Compare prev with curr.
If prev > curr, update firstVariable to prev, lastVariable to curr. Swap after all elements

http://www.ideserve.co.in/learn/leaders-in-an-array - Iterate array from right. If a[i] > currentLeader update currentLeader = a[i]

Minimum Cost Matrix

```
public static int minimumCostPathRec(int[][] costMatrix, int m, int n) {
        if (m < 0 || n < 0)
            return Integer.MAX_VALUE;
        if (m == 0 && n == 0)
            return costMatrix[0][0];
       return costMatrix[m][n]
                + minimum(minimumCostPathRec(costMatrix, m - 1, n - 1),
                          minimumCostPathRec(costMatrix, m - 1, n),
                          minimumCostPathRec(costMatrix, m, n - 1));
    }
    public static int minimumCostPathDP(int[][] costMatrix, int m, int n) {
        int[][] minimumCostPath = new int[m+1][n+1];
        minimumCostPath[0][0] = costMatrix[0][0];
        for (int i = 1; i <= m; i++) {
            minimumCostPath[i][0] = minimumCostPath[i - 1][0] + costMatrix[i][0];
        }
        for (int j = 1; j <= n; j++) {
            minimumCostPath[0][j] = minimumCostPath[0][j - 1] + costMatrix[0][j];
        }
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                minimumCostPath[i][j] = costMatrix[i][j]
                                        + minimum(minimumCostPath[i - 1][j - 1],
                                                  minimumCostPath[i - 1][j],
                                                  minimumCostPath[i][j - 1]);
            }
        }
        return minimumCostPath[m][n];
    }
```

Maximum Value of Index Element Products after Rotations

http://www.ideserve.co.in/learn/maximum-value-of-index-element-product-sum-with-only-rotations

```
 public int findMaxIndexElementProductSum(int[] array)
    {
        // currValue indicates index-element-product sum when no rotation is performed
        int sumElements = 0, currValue = 0;
        for (int i = 0; i < array.length; i++)
        {
            sumElements += array[i];
            currValue   += i*array[i];
        }
        int maxValue = currValue, n = array.length;

        for (int i = 1; i < n; i++)
        {
            currValue += sumElements - n*array[n-i];
            if (currValue > maxValue)
            {
                maxValue = currValue;
            }
        }

        return maxValue;
    }
```

https://gist.github.com/guolinaileen/4670376

```
int i = 1;
for(int j=1; j<length; j++)
      {
        if(A[j]!=A[j-1])
        {
          A[i]=A[j];
          i++;
        }
      }
```

http://www.ideserve.co.in/learn/maximum-average-subarray

maximum sum subarray of size k.

```
   int sum = input[0];
        for (int i = 1; i < k; i++)
            sum += input[i];
        // Initialized to first k elements sum
        int maxSum = sum;
        int maxSumIndex = 0;

        // Sum of remaining sub arrays
        for (int i = k; i < n; i++)
        {
            // Remove first element of the current window and add next element to the window
            sum = sum - input[i-k] + input[i] ;
            if (sum > maxSum)
            {
                maxSum = sum;
                maxSumIndex = i-k;
            }
     }
```

Trapping Rain Water

```
package com.ideserve.saurabh.questions;

public class RainWaterBetweenTowers {

    public static int getMaxRainwaterBetweenTowers(int[] towerHeight) {
        int maxSeenSoFar = 0;

        int[] maxSeenRight = new int[towerHeight.length];
        int maxSeenLeft = 0;

        int rainwater = 0;

        for (int i = towerHeight.length - 1; i >= 0; i--) {
            if (towerHeight[i] > maxSeenSoFar) {
```

```
                maxSeenSoFar = towerHeight[i];
                maxSeenRight[i] = maxSeenSoFar;
            } else {
                maxSeenRight[i] = maxSeenSoFar;
            }
        }

        for (int i = 0; i < towerHeight.length; i++) {
            rainwater = rainwater + Integer.max(Integer.min(maxSeenLeft, maxSeenRight[i]) -
towerHeight[i], 0);
            if (towerHeight[i] > maxSeenLeft) {
                maxSeenLeft = towerHeight[i];
            }
        }

        return rainwater;
    }

    public static void main(String[] args) {
        int[] towerHeight = { 1, 5, 2, 3, 1, 7, 2, 4 };
        System.out.println(getMaxRainwaterBetweenTowers(towerHeight));
    }
}
```

Maze Traversal

http://www.cs.colostate.edu/~asa/courses/cs161/fall09/pmwiki/pmwiki.php/Maze

```
 public boolean traverse (int row, int column)
   {
      boolean done = false;

      if (valid (row, column))
      {
         grid[row][column] = TRIED;  // this cell has been tried

         if (row == grid.length-1 && column == grid[0].length-1)
            done = true;  // the maze is solved
         else
         {
            done = traverse (row+1, column);     // down
            if (!done)
               done = traverse (row, column+1);  // right
            if (!done)
               done = traverse (row-1, column);  // up
            if (!done)
               done = traverse (row, column-1);  // left
         }

         if (done)  // this location is part of the final path
            grid[row][column] = PATH;
      }

      return done;
   }
```

Implement Queue using two Stacks

```
  Stack<Integer> temp = new Stack<Integer>();
    Stack<Integer> value = new Stack<Integer>();

    // Push element x to the back of queue.
    public void push(int x) {
        if(value.isEmpty()){
            value.push(x);
        }else{
            while(!value.isEmpty()){
                temp.push(value.pop());
            }

            value.push(x);

            while(!temp.isEmpty()){
```

```
                value.push(temp.pop());
            }
        }
    }
```

Convert Roman to Decimal

```
int romanToDecimal(string &str)
{
    // Initialize result
    int res = 0;

    // Traverse given input
    for (int i=0; i<str.length(); i++)
    {
        // Getting value of symbol s[i]
        int s1 = value(str[i]);

        if (i+1 < str.length())
        {
            // Getting value of symbol s[i+1]
            int s2 = value(str[i+1]);

            // Comparing both values
            if (s1 >= s2)
            {
                // Value of current symbol is greater
                // or equal to the next symbol
                res = res + s1;
            }
            else
            {
                res = res + s2 - s1;
                i++; // Value of current symbol is
                     // less than the next symbol
            }
        }
        else
        {
            res = res + s1;
        }
    }
    return res;}
```

Subset sum problem

```
private boolean subsetSum(int arr[], int sum) {
  boolean T[][] = new boolean[arr.length][sum];
  for (int i = 0; i < arr.length; i++) {
    T[i][0] = true;
  }

  for (i = 1; i < arr.length; i++) {
    for (j = 1; j <= sum; j++) {
      if (arr[i] < j) {
        T[i][j] = T[i-1][j-arr[i]] || T[i-1][j];
      } else {
        T[i][j] = T[i-1][j];
      }
     }
  }
  return T[arr.length-1][sum];
}
```

Convert BST to DLL using queue

```
Queue<Integer> queue = new LinkedList<Integer>();
last = null;
curr = null;
queue.push(root);
while (!queue.isEmpty()) {
  curr = queue.pop();
  if (temp->left != null) {
    queue.push(temp->left);
  }
```

```
  if (temp->right != null) {
    queue.push(temp->right);
  }
  curr->prev = last;
  curr->next = q.peek();
  last = curr;
}
```

PreOrder to BST

```
Node constructTree(int pre[], int size) {

        // The first element of pre[] is always root
        Node root = new Node(pre[0]);
        Stack<Node> s = new Stack<Node>();
        // Push root
        s.push(root);
        // Iterate through rest of the size-1 items of given preorder array
        for (int i = 1; i < size; ++i) {
            Node temp = null;

            /* Keep on popping while the next value is greater than
             stack's top value. */
            while (!s.isEmpty() && pre[i] > s.peek().data) {
                temp = s.pop();
            }

            // Make this greater value as the right child and push it to the stack
            if (temp != null) {
                temp.right = new Node(pre[i]);
                s.push(temp.right);
            }

            // If the next value is less than the stack's top value, make this value
            // as the left child of the stack's top node. Push the new node to stack
            else {
                temp = s.peek();
                temp.left = new Node(pre[i]);
                s.push(temp.left);
            }
        }

        return root;
    }
```

Strings Order

```
private boolean stringsOrder(String s1, String s2) {
  int j = 0;
  for (int i = 0; i < s1.length(); i++) {
    if (s1.charAt(i) == s2.charAt(j)) {
      j++;
    }
  }
  return j == s2.length() ? true : false;
}
```

#Check if Binary Tree is Balanced

```
public int findHeight(root) {
  if (root == null) {
    return 0;
  }
  return Math.max(findHeight(root->left), findHeight(root->right)) + 1;
}

public boolean isBalanced(root) {
  int lHeight = 0;
  int rHeight = 0;
  if (root == null) {
    return true;
  }
  lHeight = findHeight(root->left);
  rHeight = findHeight(root->right);

  if (Maths.abs(lHeight - rHeight) < =1 && isBalanced(root->left) && isBalanced(root->right)) {
    return true;
```

```
  }
  return false;
}
```

Check if two nodes are cousins

```
 boolean isSibling(Node node, Node a, Node b)
    {
        // Base case
        if (node == null)
            return false;

        return ((node.left == a && node.right == b) ||
                (node.left == b && node.right == a) ||
                isSibling(node.left, a, b) ||
                isSibling(node.right, a, b));
    }

    // Recursive function to find level of Node 'ptr' in
    // a binary tree
    int level(Node node, Node ptr, int lev)
    {
        // base cases
        if (node == null)
            return 0;

        if (node == ptr)
            return lev;

        // Return level if Node is present in left subtree
        int l = level(node.left, ptr, lev + 1);
        if (l != 0)
            return l;

        // Else search in right subtree
        return level(node.right, ptr, lev + 1);
    }

    // Returns 1 if a and b are cousins, otherwise 0
    boolean isCousin(Node node, Node a, Node b)
    {
        // 1. The two Nodes should be on the same level
        //       in the binary tree.
        // 2. The two Nodes should not be siblings (means
        //    that they should not have the same parent
        //    Node).
        return ((level(node, a, 1) == level(node, b, 1)) &&
                (!isSibling(node, a, b)));
    }
```

Maximum Overlapping Integers

```
// Sort arrival and exit arrays
sort(arrl, arrl+n);
sort(exit, exit+n);

// guests_in indicates number of guests at a time
int guests_in = 1, max_guests = 1, time = arrl[0];
int i = 1, j = 0;

// Similar to merge in merge sort to process
// all events in sorted order
while (i < n && j < n)
{
        // If next event in sorted order is arrival,
        // increment count of guests
        if (arrl[i] <= exit[j])
        {
                guests_in++;

                // Update max_guests if needed
                if (guests_in > max_guests)
                {
                        max_guests = guests_in;
                        time = arrl[i];
                }
```

```
                i++; //increment index of arrival array
        }
        else // If event is exit, decrement count
        { // of guests.
                guests_in--;
                j++;
        }
}
```

Quick Select – O(n) worst case- O(n^2)

```
                while (left <= right) {
                        int pivot = parition(arr, left, right);
                        if (k == pivot) {
                                return pivot;
                        }
                        if (k < pivot) {
                                right = pivot-1;
                        } else {
                                left = pivot + 1;
                        }
                }
```

http://algorithms.tutorialhorizon.com/find-the-distance-between-two-nodes-of-a-binary-tree/

n & (n-1) == 0 Power of Two

Find if robot makes a circle

```
static String[] doesCircleExist(String[] commands) {
                String ret[] = new String[commands.length];
                if (commands.length < 1) {
                        return null;
                }
                for (int i = 0; i < commands.length; i++) {
                        int x = 0;
                        int y = 0;
                        int dir = 0;
                        String move = commands[i];
                        if (move.equals("R")) {
                                dir = (dir + 1)%4;
                        } else if (move.equals("L")) {
                                dir = (dir + 3)%4;
                        } else {
                                if (dir == 0) {
                                        y++;
                                } else if (dir == 1) {
                                        x++;
                                } else if (dir == 2) {
                                        y--;
                                } else {
                                        x--;
                                }
                        }
                        if (x == 0 && y == 0) {
                                ret[i] = "YES";
                        } else {
                                ret[i] = "NO";
                        }
                }
                return ret;
        }
```

Find the third maximum in an array

```
 static int ThirdLargest(int[] arr) {
      if (arr.length < 3) {
        return 0;
      }
      int first = arr[0];
      int second = Integer.MIN_VALUE;
      int third = Integer.MIN_VALUE;
      for (int i = 1; i < arr.length; i++) {
        if (first < arr[i]) {
          third = second;
          second = first;
```

```
          first = arr[i];
        } else if (second < arr[i]) {
          third = second;
          second = arr[i];
        } else if (third < arr[i]) {
          third = arr[i];
        }
      }
      return third;
    }
```

Min Diff between Max and Min Element after removing an element
https://www.hackerrank.com/contests/101hack43/challenges/max-min-difference

Mini Max

```
public class Solution {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int[] a = new int[n];
        for(int a_i=0; a_i < n; a_i++){
            a[a_i] = in.nextInt();
        }

        // Sort the array so we know where the maximal and minimal elements will be located
        Arrays.sort(a);
        // Find the difference between the two largest elements
        int maxDifference = a[a.length - 1] - a[a.length - 2];
        // Find the difference between the two smallest elements
        int minDifference = a[1] - a[0];

        // Set index of the new maximal element
        int maxIndex = (maxDifference > minDifference) ? a.length - 2 : a.length - 1;
        // Set index of the new minimal element
        int minIndex = (minDifference > maxDifference) ? 1 : 0;

        // Print the difference
        System.out.println(a[maxIndex] - a[minIndex]);
    }
}
```

https://www.hackerrank.com/contests/university-codesprint/challenges/mini-max-sum
Given five positive integers, find the minimum and maximum values that can be calculated by summing exactly four of the five integers.

1. Iterate over the array
2. Calculate sum and keep track of min and max
3. Sum – min and sum – max are minimum and maximum values

Problem

```
public boolean wordBreak(String s, Set<String> dict) {
        // write your code here
        if(s == null || s.length() == 0) {
            return true;
        }

        boolean valid[] = new boolean[s.length() + 1];
        valid[0] = true;
        for (int i = 1; i <= s.length(); i++) {
            for (int j = 0; j < i; j++) {
                if (valid[j] && dict.contains(s.substring(j, i))) {
                    valid[i] = true;
                    break;
                }
            }
        }
        return valid[s.length()];
    }
}
```

Word Break

```
  public static boolean hasValidWords(String words) {

        // Empty string
        if(words == null || words.length() == 0) {
            return true;
        }

        int n = words.length();
        boolean[] validWords = new boolean[n];
        for (int i = 0; i < n; i++) {
            if (dictionary.contains(words.substring(0, i + 1))) {
                validWords[i] = true;
            }
            if (validWords[i] == true && (i == n - 1))
                return true;
            if (validWords[i] == true) {
                for (int j = i + 1; j < n; j++) {
                    if (dictionary.contains(words.substring(i + 1, j + 1))) {
                        validWords[j] = true;
                    }
                    if (j == n - 1 && validWords[j] == true) {
                        return true;
                    }
                }
            }
        }
        return false;
    }
```

Longest Palindromic Substring

```
public String longestPalindrome(String s) {
          if (s.isEmpty()) {
                    return null;
          }

          if (s.length() == 1) {
                    return s;
          }

          String longest = s.substring(0, 1);
          for (int i = 0; i < s.length(); i++) {
                    // get longest palindrome with center of i
                    String tmp = helper(s, i, i);
                    if (tmp.length() > longest.length()) {
                              longest = tmp;
                    }

                    // get longest palindrome with center of i, i+1
                    tmp = helper(s, i, i + 1);
                    if (tmp.length() > longest.length()) {
                              longest = tmp;
                    }
          }

          return longest;
}

// Given a center, either one letter or two letter,
// Find longest palindrome
public String helper(String s, int begin, int end) {
          while (begin >= 0 && end <= s.length() - 1 && s.charAt(begin) == s.charAt(end)) {
                    begin--;
                    end++;
          }
          return s.substring(begin + 1, end);
}
```

Intersection of Two arrays – O(m+n)

Group Anagrams – O(n * klogk) – K is the size of the largest string from the array of strings.

```
public List<List<String>> groupAnagrams(String[] strs) {
        if (strs.length == 0) {
            return null;
        }
        HashMap<String, List<String>> map = new HashMap<String, List<String>>();
        for (String s : strs) {
            char ch[] = s.toCharArray();
            Arrays.sort(ch);
            String sorted = String.valueOf(ch);
            if (map.containsKey(sorted)) {
                map.get(sorted).add(s);
            } else {
                List<String> list = new ArrayList<String>();
                list.add(s);
                map.put(sorted, list);
            }
        }
         return new ArrayList<List<String>>(map.values());
    }
```

Implement Trie

https://raw.githubusercontent.com/mission-peace/interview/master/src/com/interview/suffixprefix/Trie.java

Insert

1. Check if the first character in the word is a child of the root.
2. If not then create a new node and add this character as key and the new node as value to the map of children of the root
3. Update current = node and at the end of the iteration update endOfWord = true.

TC of Trie = O(nm)

```
  public void insert(String word) – O(average length * no. of words) {
        TrieNode current = root;
        for (int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            TrieNode node = current.children.get(ch);
            if (node == null) {
                node = new TrieNode();// TrieNode(char ch);
                current.children.put(ch, node); // current.children[ch]=
node
            }
            current = node;
        }
        //mark the current nodes endOfWord as true
        current.endOfWord = true;
     }
public boolean search(String word) {
        TrieNode current = root;
        for (int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            TrieNode node = current.children.get(ch);
            //if node does not exist for given char then return false
            if (node == null) {
                return false;
            }
            current = node;
        }
        //return true of current's endOfWord is true else return false.
        return current.endOfWord;
    }
```

http://techieme.in/word-frequency-in-stream/

http://techieme.in/building-an-autocomplete-system-using-trie/

Top K most frequent words in a file
http://www.zrzahid.com/top-k-or-k-most-frequent-words-in-a-document/ - O(n * logK)

Sort Based on values
Sort by values

```
private static HashMap<String, Integer> sortByValue(Map<String, Integer> unsortMap) {

		// 1. Convert Map to List of Map
		List<Map.Entry<String, Integer>> list = new ArrayList<Map.Entry<String,
Integer>>(unsortMap.entrySet());

		// 2. Sort list with Collections.sort(), provide a custom Comparator
		//    Try switch the o1 o2 position for a different order
		Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {
			public int compare(Map.Entry<String, Integer> o1,
					Map.Entry<String, Integer> o2) {
				return (o1.getValue()).compareTo(o2.getValue());
			}
		});

		// 3. Loop the sorted list and put it into a new insertion order Map LinkedHashMap
		HashMap<String, Integer> sortedMap = new LinkedHashMap<String, Integer>();
		for (Map.Entry<String, Integer> entry : list) {
			sortedMap.put(entry.getKey(), entry.getValue());
		}
		return sortedMap;
	}
```

Longest Valid Paranthesis

```
public int longestValidParentheses(String s) {
    Stack<Integer> stack = new Stack<Integer>();
    int max = 0;
    stack.push(-1);
    for (int i = 0; i < s.length(); i++) {
        char ch = s.charAt(i);
        if (ch == ')' && stack.size() > 1 && s.charAt(stack.peek()) == '(') {
            stack.pop();
            max = Math.max(max, i - stack.peek());
        } else {
            stack.push(i);
        }
    }
    return max;
  }
```

Largest Rectangle Area Histogram

```
public static int largestRectangleArea(int[] height) {
		if (height == null || height.length == 0) {
			return 0;
		}

		Stack<Integer> stack = new Stack<Integer>();

		int max = 0;
		int i = 0;

		while (i < height.length) {
			//push index to stack when the current height is larger than the previous one
			if (stack.isEmpty() || height[i] >= height[stack.peek()]) {
				stack.push(i);
				i++;
			} else {
			//calculate max value when the current height is less than the previous one
				int p = stack.pop();
				int h = height[p];
				int w = 0;
				if (stack.isEmpty()) {
					w = i;
```

```
                                } else {
                                        w =  i - stack.peek() - 1;
                                }
                                max = Math.max(h * w, max);
                        }

                }
                while (!stack.isEmpty()) {
                  //area;
                }
```

Max Height of N-ary tree

```
int getHeight(TreeNode root){
            if(root == null || root.isLeaf() ) { return 0; }
            int tallest = 0;

            for(TreeNode n: root.children()){
                 int height = getHeight(n);
                 if(height > tallest ){
                   tallest = height);
                 }
             }

             return tallest + 1;
```

http://www.geeksforgeeks.org/find-water-in-a-glass/
https://careercup.com/question?id=5704645247762432
http://www.geeksforgeeks.org/find-longest-palindrome-formed-by-removing-or-shuffling-chars-from-string/

http://www.geeksforgeeks.org/find-longest-palindrome-formed-by-removing-or-shuffling-chars-from-string/

#Longest Palindrome by deleting or shuffling chars

```
public String longestPalindrome(String str) {
  int count[] = new int[256];
  for (int i = 0; i < str.length(); i++) {
    count[str.charAt(i)]++;
  }
  String start = "";
  String mid = "";
  String end = "";
  for (char ch = 'a'; ch <= 'z'; ch++) {
    if (count[ch] % 2 == 1) {
    //contains only one character and can be overriden by the next char with //odd frequency
      mid = ch;
      count[ch]--;
      ch--;
    } else {
      for (int i = 0; i < count[ch]/2; i++) {
        start += ch;
      }
    }
  }
  end = reverseString(start);
  return start + mid + end;
}
public String reverseString(String str) {
  char a[] = str.toCharArray();
  int i = 0;
  int j = a.length - 1;
  while (i < j) {
    char ch = a[i];
    a[i] = a[j];
    a[j] = ch;
    i++;
    j--;
  }
  return String.valueOf(a);
}
```

Number of 0s in sorted matrix – O(n)

```
private static int countNumZeroes(int[][] matrix) {
    int row = matrix.length - 1, col = 0, numZeroes = 0;
    while (col < matrix[0].length) {
      while (matrix[row][col] != 0) {
        if (--row < 0) {
          return numZeroes;
        }
      }
      // Add one since matrix index is 0 based
      numZeroes += row + 1;
      col++;} return numZeroes;
```

#Longest Zigzag subsequence

```
public int lzs(int a[]) {
  int n = a.length;
  int z[][] = int[n][2];
  for (int i = 0; i < n; i++) {
    z[i][0] = 1;
    z[i][1] = 1;
  }
  int max = 0;
  for (int i = 1; i < n; i++) {
    for (int j = 0; j < i; j++) {
      if (a[j] < a[i]) {
        z[i][0] = Math.max(z[i][0], z[j][1] + 1);
      }
      if (a[j] > a[i]) {
        z[i][1] = Math.max(z[i][1], z[j][0] + 1);
       }
     max = Math.max(max, Math.max(z[i][0], z[i][1]);
    }
  }
  return max;
}
```

KMP

```
public int[] failure(String pattern[]) {
  int n = pattern.length();
  int f[] = new int[n];
  int i = 1;
  int j = 0;
  while (i < n) {
    if (pattern.charAt(i) == pattern.charAt(j)) {
      f[i] = j + 1;
      i++;
      j++;
    } else {
      if (j != 0) {
        j = f[j-1];
      } else {
        f[i] = 0;
        i++;
      }
    }
  }
return f;
}

public boolean kmp(String text, String pattern) {
  int f[] = failure(pattern);
  int i = 0;
  int j = 0;
  while (i < text.length() && j < pattern.length()) {
    if (text.charAt(i) == pattern.charAt(j)) {
     i++;
     j++;
    } else {
      if (j != 0) {
        j = f[j-1];
      } else {
        i++;
      }
    }
```

```
  }
  if (j == pattern.length()) {
    return true;
  }
  return false;
}
```

Word Ladder

```
public int ladderLength(String begin, String endWord, Set<String> wordList) {
        Queue<String> queue = new LinkedList<String>();
                wordList.add(endWord);
                queue.add(begin);
                int dist = 1;
                while (!queue.isEmpty()) {
                  for (int size = queue.size(); size > 0; size--) {
                        String word = queue.remove();
                        if (endWord.equals(word)) {
                                return dist;
                        }
                        char arr[] = word.toCharArray();
                        for (int i = 0; i < arr.length; i++) {
                                char ch = arr[i];
                                for (char j = 'a'; j <= 'z'; j++) {
                                        arr[i] = j;
                                        word = String.valueOf(arr);
                                        if (wordList.contains(word)) {
                                                queue.add(word);
                                                wordList.remove(word);
                                        }
                                        arr[i] = ch;
                                }
                        }
                  }
                  dist++;
                }
                return dist;
    }
```

N Queens

```
Void NQueensBackTrack(row,n){
      for(i = 1 to n)
            if(QueenSafe(row,i))
                  board[row][i] = true; //Place a queen
                  if(row == n)
                        Print Board //we're done
                  else
                        NQueensBackTrack(row+1,n);
}
```

```
Boolean QueenSafe(row,col){
        for(i=1 to n)
                if(board[row][i] has a queen || board[i][col] has a queen)
                        return false
        reset = min(row,col)-1
        for(i = row-reset, j = col-reset; i<=n && j<=n; i++, j++)
                if(board[i][j] has a queen)
                        return false
        for(i = row-reset, j = col+reset; i<=n && j>0; i++, j--)
                if(board[i][j] has a queen)
                        return false
        return true;
}
```

For rows and cols 2nd condition check on upper left and lower left of the current location.
Row++ Col—-
and row—- and col --

Longest Common Prefix

```
public String longestCommonPrefix(String[] strs) {
                if (strs.length < 1) {
                        return "";
                }
                String firstWord = strs[0];
                StringBuilder sb = new StringBuilder();
                for (int i = 0; i < firstWord.length(); i++) {
                        char curr = firstWord.charAt(i);
                        for (int j = 1; j < strs.length; j++) {
                                String word = strs[j];
                                int len = word.length();
                                if (i > len - 1 || curr != word.charAt(i)) {
                                        return sb.toString();
                                }
                        }
                        sb.append(curr);
                }
                return sb.toString();
          }
```

#Uppercase lowercase combination

```
public List<String> combString(String str) {
  List<String> result = new ArrayList<String>();
  combString(str, "", 0, result);
  return result;
}
public static void combString(String str, String partial, int index, List<Integer> result) {
  if (index = str.length()) {
    result.add(partial);
    return;
  }
  char ch = str.charAt(index);
  if (!Character.isAlphabet(ch)) {
    combString(str, partial + ch, index + 1, result);
  } else {
    combString(str, partial + Character.toLowerCase(ch), index + 1, result);
    combString(str, partial + Character.toUpperCase(ch), index + 1, result);
  }
}
```

http://www.geeksforgeeks.org/dynamic-programming-set-27-max-sum-rectangle-in-a-2d-matrix/

#Maximum Sub rectangle of a matrix

```
public void maxRectangle(int mat[][]) {
  int maxSum = Integer.MIN_VALUE;
  int topLeft = 0;
  int topRight = 0;
  int top = 0;
  int bottom = 0;
  int cols = mat[0].length;
```

```
  int arr[] = new int[cols];
  int start = 0;
  int end = 0;
  for (int left = 0; left < col; left++) {
    for (int i = 0; i < arr.length; i++) {
      arr[i] = 0;
    }
    for (int right = left; right < col; right++) {
      for (int i = 0; i < arr.length; i++) {
        arr[i] += mat[i][right];
      }
      int sum = kadenceSum(arr);
      if (sum > maxSum) {
        maxSum = sum;
        // get start and enf of max subarry from kadence
        topRight = end;
        topLeft = start;
        top = left;
        bottom = right;
      }
    }
  }
}
```

#Find if a path exists betweent two vertices in a graph - BFS

```
public boolean bfs(int start, int end) {
  Queue<Integer> queue = new LinkedList<Integer>();
  boolean visited[] = new boolean[n];
  visited[start] = true;
  queue.add(start);
  while (!queue.isEmpty()) {
    int temp = queue.remove();
    for (int i = adj[temp].begin; i <= adj[temp].endNode; i++) {
      if (i == end) {
        return true;
      }
      if (!visited[i]) {
        visited[i] = true;
        queue.add(i);
      }
    }
  }
}
```

Topological Sort
2nd Method using DFS and Stack

1) **For each vertex recursively calls adjacent vertices using dfs**
2) **When a vertex without and adjacent vertices is reached add it to the stack**
3) **Print the stack**

Order of the Algorithm

Time Complexity is O(|E| + |V|)
Space Complexity is O(|V|)

Josephus Problem

```
int josephus(int n, int k) {
if (n == 1)
          return 1;
 } else {
/* After the k-th person is killed, we're left with a circle of n-1, and we start the next count with
the person whose number in the original problem was (
k%n) + 1
*/
          return (josephus(n - 1, k) + k) % n + 1;
}
```

Random Number based on Weight

```
for (Item i : items) {
    totalWeight += i.getWeight();
}
```

```
// Now choose a random item
int randomIndex = -1;
double random = Math.random() * totalWeight;
for (int i = 0; i < items.length; ++i)
{
    random -= items[i].getWeight();
    if (random <= 0.0d)
    {
        randomIndex = i;
        break;
    }
}
```

#Check if T2 is subtree of T1

```
public boolean checkTree(TreeNode t1, TreeNode t2) {
  if (t2 == null) {
    return true;
  }
  return subTree(t1, t2);
}
public boolean subTree(TreeNode t1, TreeNode t2) {
  if (t1 == null) {
    return false;
  }
  if (t1.data == t2.data && matchTree(t1.left, t2.left)) {
    return true;
  }
  return subTree(t1.left, t2) || subTree(t1.right, t2);
}
public boolean matchTree(TreeNode t1, TreeNode t2) {
  if (t1 == null && t2 == null) {
    return true;
  }
  if (t1 != null && t2 != null) {
    return matchTree(t1.left, t2.left) && return matchTree(t1.right, t2.right);
  }
  return false;
}
```

Binary Search without length

```
int searchIndex(int list[], int value) {
 // Find the end of the array
  int index = 1;
  while (list.getElement(index) != -1 && list.getElement(index) < value) {
    index *= 2;
  }
  return binarySearch(list, value, index);
}
int binarySearch(int list[], int value, int index) {
  int low = 0;
  int high = index;
  while (low <= middle) {
     middle = low+high/2;
    if (list.elementAt(middle) == value) {
      return mid;
    } else if (middle < value) {
      low = middle + 1;
    } else if (middle > value || middle == -1) {
      high = middle - 1;
  }
  return -1;
}
```

Peaks and valleys

1) Start at 1 and increment by 2
2) Find the max index from i-1, i, i+1. Check for the index < len of array. If greater than length of array use a minimum value to that index
3) If index != the max index swap

Closest element to the right of a number in a sorted array

1) If a[mid] > k then store res = mid and high = mid-1;
2) Else low = mid+1
3) Return res;

Find an element in sorted array a[i] = i

1) Take mid and check if a[mid] – mid == 0 return mid;
2) If a[mid] – mid > 0 high = mid-1
3) Else low = mid+1;
4) Return -1;

Product Without * or / O(logs) s – size of smaller number

```
public static int minProduct(int a, int b) {
     int bigger = a< b? b: a;
     int smaller = a< b? a : b;
     return minProductHelper(smaller, bigger);
}

public static int minProductHelper(int smaller, int bigger) {
     if (smaller == 0) {
             return 0;
     }
     else if (smaller == 1) {
             return bigger;
     }
     int s =smaller >> 1; // Divide by 2
     int halfProd = minProductHelper(s, bigger);

     if (smaller% 2 == 0) {
             return halfProd + halfProd;
     } else {
             return halfProd + halfProd + bigger;
     }
}
```

```
#Longest Increasing Subarray in O(max(N/L, L) TC
#Beginning and Ending
public int lis(int a[]) {
  int count = 1;
  int i = 0;
  while (i < a.length) {
    isSkip = false;
    for (int j = i + count - 1; j >= 0; j--) {
      if (A[j] > A[j+1]) {
        i = j + 1;
        isSkip = true;
        break;
      }
    }
    if (!isSkip) {
      i = i + count - 1;
      while (i + 1 < a.length && a[i] < a[i+1]) {
        i++;
        count++;
      }
      int startingIndex = i - count + 1;
      int endingIndex = i;
    }

  }
  return startingIndex+endingIndex;
}
```

```
#Huffman Decode
void decode(String S, Node root)
{
    StringBuilder sb = new StringBuilder();
    Node c = root;
    for (int i = 0; i < S.length(); i++) {
        c = S.charAt(i) == '1' ? c.right : c.left;
        if (c.left == null && c.right == null) {
            sb.append(c.data);
            c = root;
        }
    }
    System.out.print(sb);
}
#Remove b and replace a with dd
```

```
public int removeAndReplace(int a[]) {
  int count = 0;
  int countOfA = 0;
  for (int i = 0; i < a.length; i++) {
    if (a[i] != 'b') {
      a[count] = a[i];
    }
    if (a[i] == 'a') {
      countOfA++;
    }
  }
  int currIndex = count;
  int newIndex = currIndex + countOfA;
  while (currIndex >= 0) {
    if (a[currIndex] == 'a') {
      a[newIndex] = 'd';
      a[newIndex - 1] = 'd';
      newIndex = newIndex - 2;
    } else {
      a[newIndex] = a[currIndex];
      newIndex--;
    currIndex--;
  }
}
```

Find common elements in two sorted equal arrays – O(m+n)

1) Are the arrays of equal length
2) Are the arrays sorted?

```
public List<Integers commonElements(int m[], int n[]) {
 int mLen = m.length;
 int nLen = n.length;
 List<Integer> common = new ArrayList<Integer>();
 int i = 0;
 int j = 0;
 while (i < mLen && j < nLen) {
   if (m[i] == n[j]) {
     common.add(m[i]);
     i++;
     j++;
   } else if (m[i] < n[j]) {
     i++;
   } else {
     j++;
   }
 }
 return common;
}
```

#Insert in BST
#Insert in Binary Search Tree

```
 Node insertRec(Node root, int key) {

        /* If the tree is empty, return a new node */
        if (root == null) {
            root = new Node(key);
            return root;
        }

        /* Otherwise, recur down the tree */
        if (key < root.key)
            root.left = insertRec(root.left, key);
        else if (key > root.key)
            root.right = insertRec(root.right, key);

        /* return the (unchanged) node pointer */
        return root;
    }
```

#First Entry Larger than k in bst

```
public int largetThanKinBST(TreeNode node, int k) {
  if (node == null) {
    return null;
  }
  boolean isFound = false;
```

```
  int value = 0;
  while (node != null) {
    if (node.data == k) {
      isFound = true;
      node = node.right;
    } else if (node.data > k) {
      first = node.data;
      node = node.left;
    } else if (node.data < k) {
      node = node.right;
    }
  }
  return isFound ? first : null;
}
```

Find three closest elements from given three sorted arrays – O(m + n + o)

```
#Find three closest elements from given three sorted arrays
public int findClosest(int a[], int b[], int c[]) {
  int aLen = a.length;
  int bLen = b.length;
  int cLen = c.length;
  int i = 0;
  int j = 0;
  int k = 0;
  int minDiff = Integer.MAX_VALUE;
  while (i < aLen && j < bLen && k < cLen) {
    int min = Math.min(a[i], Math.min(b[j], c[k]));
    int max = Math.max(a[i], Math.max(b[j], c[k]));
    int diff = max - min;
    if (diff < minDiff) {
      minDiff = diff;
    }
    if (a[i] == min) {
      i++;
    } else if (b[j] == min) {
      j++;
    } else {
      k++;
    }
    return minDiff;
  }
}
```

#Find the closest pair of equal entries in a list of strings

```
public int closestPair(int a[]) {
  HashMap<String, Integer> map = new HashMap<String, Integer>();
  int minDiff = Integer.MAX_VALUE;
  for (int i = 0; i < a.length; i++) {
    String str = a[i];
    if (map.containsKeys(str)) {
      int prevIndex = map.get(str);
      int diff = i - prevIndex;
      int minDiff = Math.min(minDiff, diff);
    }
    map.put(str, i);
  }
  return minDiff;
}
```

#Longest Consecutive Sequence in a Binary Tree

```
public int lcs(TreeNode root) {
  int res = 0;
  lcs(root, 0, root.data, res);
  return res;
}
public void lcs(TreeNode root, count, int expectedData, int res) {
  if (root == null) {
    return null;
  }
  if (root.data == expectedData) {
    count++;
  } else {
    count = 1;
  }
```

```
  res = Math.max(res, count);
  lcs(root.left, count, root.data + 1, res);
  lcs(root.right, count, root.data + 1 res);
}
```

#Dining philosopher
https://www.youtube.com/watch?v=_ruovgwXyYs

```
Pick the smaller fork before picking the bigger fork and put down the bigger fork before the smaller
fork to avoid circular wait.
takeLeft(fork[Math.min(I, i+1 mod 5)]);
takeRight(fork[Math.max(I, I+1 mod 5)]);
eat
putRight(fork[i+1 mod 5])
putLeft(fork[i])
#Powerset
public List<List<Integer>> powerSet(List<List<Integer> list, int index) {
  List<List<Integer>> allSubsets = new ArrayList<List<Integer>();
  if (index = list.size()) {
    allSubsets = new ArrayList<List<Integer>>();
    allSubsets.add(new ArrayList<List<Integer>());
  } else {
    allSubsets = powerSet(list, index + 1);
    int item = list.get(index);
    List<List<Integer>> tempList = new ArrayList<List<Integer>();
    for (List<Integer> setSoFar : allSubsets) {
      List<Integer> newSet = new ArrayList<Integer>();
      newSet.addAll(setSoFar);
      newSet.add(item);
      tempList.add(newSet);
    }
    allSubsets.addAll(tempList);
  }
  return allSubsets;
}
```

Reconstruct Itinerary

```
public List<String> findItinerary(String[][] tickets) {
        LinkedList<String> list = new LinkedList<String>();
        HashMap<String, PriorityQueue<String>> map = new HashMap<String, PriorityQueue<String>>();
        for (int i = 0; i < tickets.length; i++) {
            String[] ticket = tickets[i];
            String origin = ticket[0];
            String dest = ticket[1];
            if (!map.containsKey(origin)) {
              map.put(origin, new PriorityQueue<String>());
            }
              map.get(origin).add(dest);

        }
        dfs("JFK", list, map);
        return list;


    }
    public void dfs(String origin, LinkedList<String> list, HashMap<String, PriorityQueue<String>>
map) {
        PriorityQueue<String> pq = map.get(origin);
        while (pq != null && !pq.isEmpty()) {
            dfs(pq.poll(), list, map);
        }
        list.addFirst(origin);
    }
```

Majority Element II

```
public List<Integer> majorityElement(int[] nums) {
        List<Integer> list = new ArrayList<Integer>();
        if (nums.length == 0) {
            return list;
        }
        int num1 = nums[0];
        int count1= 0;
        int num2 = nums[0];
        int count2 = 0;
```

```
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == num1) {
                count1++;
            } else if (nums[i] == num2) {
                count2++;
            } else if (count1 == 0) {
                num1 = nums[i];
                count1 = 1;
            } else if (count2 == 0) {
                num2 = nums[i];
                count2 = 1;
            } else {
                count1--;
                count2--;
            }
        }
         count1 = 0;
         count2 = 0;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == num1) {
                count1++;
            } else if (nums[i] == num2) {
                count2++;
            }
        }
        if (count1 > Math.ceil(nums.length/3)) {
            list.add(num1);
        }
        if (count2 > Math.ceil(nums.length/3)) {
            list.add(num2);
        }
        return list;
    }
#  Find All Numbers Disappeared in an Array

public List<Integer> findDisappearedNumbers(int[] nums) {
        List<Integer> list = new ArrayList<Integer>();
        for (int i = 0; i < nums.length; i++) {
            int val = Math.abs(nums[i]) - 1;
            if (nums[val] > 0) {
                nums[val] = -nums[val];
            }
        }
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] > 0) {
                list.add(i+1);
            }
        }
        return list;
    }



#Find all duplicates in array
public List<Integer> findDuplicates(int[] nums) {
        List<Integer> list = new ArrayList<Integer>();
        for (int i = 0; i < nums.length; i++) {
            int val = Math.abs(nums[i]) - 1;
            if (nums[val] > 0) {
              nums[val] = -nums[val];
            } else {
                list.add(Math.abs(nums[i]));
            }
        }
        return list;
    }




#Invert a Binary Tree
public TreeNode invertTree(TreeNode root) {
```

```
        if (root == null) {
            return null;
        }

        final Deque<TreeNode> stack = new LinkedList<>();
        stack.push(root);


        while(!stack.isEmpty()) {
            final TreeNode node = stack.pop();
            final TreeNode left = node.left;
            node.left = node.right;
            node.right = left;


            if(node.left != null) {
                stack.push(node.left);
            }
            if(node.right != null) {
                stack.push(node.right);
            }
        }
        return root;
    }
```

Digital Root - Ex:- 38 = 3+8 = 11 = 1+1 = 2 = num - 9 * ((num - 1) / 9);
Josephus Number - return (josephus(n-1, k) + k - 1)% n+ 1;

Longest Palindrome Length

```
public int longestPalindrome(String s) {
        HashSet<Character> list = new HashSet<Character>();
        int count = 0;
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (list.contains(ch)) {
                count++;
                list.remove(ch);
            } else {
                list.add(ch);
            }
        }
        if (list.size() != 0) {
            return count * 2 + 1;
        }
        return count * 2;
    }
```

Can construct a string from another string

```
public boolean canConstruct(String ransomNote, String magazine) {
        if (ransomNote.length() == 0 && magazine.length() == 0) {
            return true;
        }
        if (magazine.length() == 0) {
            return false;
        }
        if (ransomNote.length() == 0) {
            return true;
        }
       int arr[] = new int[128];
       for (int i = 0; i < magazine.length(); i++) {
           arr[magazine.charAt(i)]++;
       }
       for (int i = 0; i < ransomNote.length(); i++) {
           arr[ransomNote.charAt(i)]--;
           if (arr[ransomNote.charAt(i)] < 0) {
               return false;
           }
       }
        return true;
    }
```

Arranging Coins
The maximum length of sum of consecutive integer <= n
x.(x+1)/2 <= n

```
  public int arrangeCoins(int n) {
        int low = 0;
        int high = n;
        while (low <= high) {
            int mid = (low + high)/2;
            if ((0.5 * mid * mid + 0.5 * mid ) <= n) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return low - 1;
    }
```

Find all anagrams in a string

```
public List<Integer> findAnagrams(String s, String p) {
        if (p == null || p == "" || s == null || s == "") {
            return null;
        }
        List<Integer> list = new ArrayList<Integer>();
        int arr[] = new int[256];
        for (char ch : p.toCharArray()) {
            arr[ch]++;
        }
        int left = 0;
        int right = 0;
        int pLen = p.length();
        int sLen = s.length();
        int count = pLen;
        while (right < sLen) {
            char ch = s.charAt(right);
            if (arr[ch] >= 1) {
                count--;
            }
            arr[ch]--;
            right++;
            if (count == 0) {
                list.add(left);
            }
            if (right - left == pLen) {
                if (arr[s.charAt(left)] >= 0) {
                    count++;
                }
                arr[s.charAt(left)]++;
                left++;
            }
        }
        return list;
    }
```

Cows and Bulls

```
public String getHint(String secret, String guess) {
        int arr[] = new int[10];
        int bull = 0;
        int cow = 0;
        for (int i = 0; i < secret.length(); i++) {
            int s = secret.charAt(i) - '0';
            int g = guess.charAt(i) - '0';
            if (s == g) {
                bull++;
            } else {
                if (arr[s] < 0) {
                    cow++;
                }
                if (arr[g] > 0) {
                    cow++;
                }
                arr[s]++;
                arr[g]--;
            }
        }
        StringBuilder sb = new StringBuilder();
        sb.append(bull);
        sb.append("A");
        sb.append(cow);
```

```
        sb.append("B");
        return sb.toString();
    }
```

Isomorphic Strings

```
Keep a track of last seen index
public boolean isIsomorphic(String s, String t) {
        int arr1[] = new int[256];
        int arr2[] = new int[256];
         for (int i = 0; i < s.length(); i++) {
            if (arr1[s.charAt(i)] != arr2[t.charAt(i)]) {
                return false;
            }
            if (arr1[s.charAt(i)] == 0) {
                arr1[s.charAt(i)] = i + 1; // 0 Is default in the array so marking with I + 1;
                arr2[t.charAt(i)] = i + 1 ;
            }
        }
        return true;
    }
```

Rectangle Area

Area of two rectangles – overlapping area

```
public int computeArea(int A, int B, int C, int D, int E, int F, int G, int H) {


        int areaOfSqrA = (C-A) * (D-B);
         int areaOfSqrB = (G-E) * (H-F);


        int left = Math.max(A, E);
        int right = Math.min(G, C);
        int bottom = Math.max(F, B);
        int top = Math.min(D, H);


        //If overlap
        int overlap = 0;
        if(right > left && top > bottom)
             overlap = (right - left) * (top - bottom);


        return areaOfSqrA + areaOfSqrB - overlap;
    }
```

Connected Components in an undirected graph

DFS with unconnected vertices. If (!visited[vertex]) result++

#Number of connected components in an undirected graph

```
public int dfsUtil(Node node) {
  boolean visited[] = new boolean[n];
  for (int i = 0; i < V; i++) {
    visited[node[i]] = false;
  }
  int result = 0;
  for (int i = 0; i < V; i++) {
    if (!visited[v]) {
      result++;
      dfs(v, visited);
    }
  }
}
public void dfs(Node node, boolean[] visited) {
  visited[node] = true;
  Iterator it = adj[node].listIterator;
  while (it.hasNext()) {
    Node temp = it.next();
    if (!visited[temp]) {
      dfs(temp, visited);
    }
  }
}
```

Linked List Random Node

```
public int getRandom() {
        int count = 0;
        ListNode curr = head;
```

```
        int res = 0;
        while (curr != null) {
            int currVal = curr.val;
            int randVal = (int) Math.random() * (count + 1);
            if (randVal == count) {
                res = currVal;
            }
            curr = curr.next;
            count++;
        }
        return res;
    }
```

Count Numbers with Unique Digigts

```
public int countNumbersWithUniqueDigits(int n) {
        if (n == 0) {
            return 1;
        }
        int available = 9;
        int res = 10;
        int uniqueDigits = 9;
        int i = n;
        while (i > 1 && available > 0) {
            uniqueDigits = uniqueDigits * available;
            res += uniqueDigits;
            available--;
            i--;
        }
        return res;
    }
```

Shuffle Array

1) Make copy of the array . int[] newArray = Arrays.copyOf(array, array.length);
2) Swap with random index. int swapIndex = rand.nextInt(i + 1)
3) Rand.nextInt gives a rand number from 0 to i.

Kth smallest element in a sorted matrix

O(n + k logn)

O(nlogn + klogn)

Creating pq of size n and polling k times so klogn

```
public int kthSmallest(int[][] matrix, int k) {
        int n = matrix.length;
        PriorityQueue<Tuple> pq = new PriorityQueue<Tuple>(n);
        for (int i = 0; i < n; i++) {
            pq.add(new Tuple(0, i,matrix[0][i]));
        }
        for (int i = 0; i < k-1; i++) {
            Tuple temp = pq.remove();
            if (temp.x == n - 1) {
                continue;
            }
            Tuple newTup = new Tuple(temp.x + 1, temp.y, matrix[temp.x + 1][temp.y]);
            pq.add(newTup);
        }
        return pq.remove().value;
    }
```

Find Right Interval – O(n * logN)

TC of TreeMap insert is O(logN)

```
public int[] findRightInterval(Interval[] intervals) {
        int result[] = new int[intervals.length];
        TreeMap<Integer, Integer> map = new TreeMap<Integer, Integer>();
        //HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        for (int i = 0; i < intervals.length; i++) {
            map.put(intervals[i].start, i);
        }
        for (int i = 0; i < intervals.length; i++) {
            Integer val = map.ceilingKey(intervals[i].end);
            result[i] = (val == null) ? -1 : map.get(val);
        }
```

```
        return result;
    }
```

4Sum ii - Number of tuples in 4 arrays equal to 0

1. Calculate sum for all pairs in A and B and add it to hashmap and value is how many times that sum is formed
2. Calculate sum for all pairs and check if –(sum) is present in hashMap. If yes increment count += map.get(sum);

Max product of word length

```
public int maxProduct(String[] words) {
        int value[] = new int[words.length];
        if (words == null || words.length == 0) {
            return 0;
        }
        for (int i = 0; i < words.length; i++) {
            value[i] = 0;
            for (int j = 0; j < words[i].length(); i++) {
                value[i] |= 1 <<(words[i].charAt(j) - 'a');// Generates how many  type of lower case
letters
            }
        }
        int max = Integer.MIN_VALUE;
        for (int i = 0; i < words.length; i++) {
            for (int j = i+1; j < words.length; j++) {
                if (value[i] & value[j] == 0) { // If two strings have different characters
                    max= Math.max(max, words[i].length() * words[j].length());
                }
            }
        }
        return max;
    }
```

Binary to gray

```
num ^ (num >> 1);
```

Josephus Circle

```
while (p <= n) {
            p = 2 * p; // least pot greater than n
        }
int res = (2 * n) - p + 1;
```

Increasing Triplet Subsequence

```
public boolean increasingTriplet(int[] nums) {
        if (nums.length == 0) {
            return false;
        }
        int min = Integer.MAX_VALUE;
        int secMin = Integer.MAX_VALUE;
        for (int num : nums) {
           if (num <= min && num > secMin) {
               min = num;
           } else if (num < secMin) {
               secMin = num;
           } else if (num > secMin) {
               return true;
           }
        }
        return false;
    }
```

Longest Increasing Subsequence

```
public int lengthOfLIS(int[] nums) {
        int res[] = new int[nums.length];
        if (nums.length == 1) {
            return 1;
        }
        if (nums.length == 0) {
            return 0;
        }

        for(int i=0; i < nums.length; i++) {
                        res[i] = 1;
        }
```

```
        for (int i = 1; i < nums.length; i++) {
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j]) {
                    if (res[i] < res[j] + 1) {
                        res[i] = res[j] + 1;
                    }
                }
            }
        }
        int longest = 0;
        for (int i = 0; i < res.length; i++) {
            longest = Math.max(longest, res[i]);
        }
        return longest;
    }
```

Valid Perfect Square

```
public boolean isPerfectSquare(int num) {
        int low = 1;
        int high = num;
        while (low <= high) {
          long mid = low + ((high - low)/2);
          if (mid * mid == num) {
              return true;
          } else if (mid * mid > num) {
              high = (int) mid - 1;
          } else {
              low = (int) mid + 1;
          }
        }
        return false;
    }
```

Minimum Path Sum

```
public int minPathSum(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;
        int dp[][] = new int[m][n];
        dp[0][0] = grid[0][0];
        for (int i = 1; i < m; i++) {
            dp[i][0] = dp[i-1][0] + grid[i][0];
        }
        for (int i = 1; i < n; i++) {
            dp[0][i] = dp[0][i-1] + grid[0][i];
        }
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                dp[i][j] = Math.min(dp[i-1][j], dp[i][j-1]) + grid[i][j];
            }
        }
        return dp[m-1][n-1];
    }
```

Populating Next Right Pointers in Each Node

```
  public void connect(TreeLinkNode root) {
        if (root == null) {
            return;
        }
        TreeLinkNode pre = root;
        TreeLinkNode curr = null;
        while (pre.left != null) {
            curr = pre;
            while (curr != null) {
                curr.left.next = curr.right;
                if (curr.next != null) {
                    curr.right.next = curr.next.left;
                }
                curr = curr.next;
            }
            pre = pre.left;
        }
    }
```

Conversion of matrix element to array index –

In m x n matrix mat[i][j] = a[i * n + j]

a[i] = mat[i/n][i%n]

Container with most water – O(n)

```
public int maxArea(int[] height) {
        int left = 0;
        int right = height.length - 1;
        int maxArea = 0;
        while (left <= right) {
            int currArea = Math.min(height[left], height[right]) * (right - left);
            maxArea = Math.max(maxArea, currArea);
            if (height[left] < height[right]) {
                left = left + 1;
            } else {
                right = right - 1;
            }
        }
        return maxArea;
    }
```

Set Matrix Zeroes

```
int row = matrix.length;
        int col = matrix[0].length;
        boolean firstRow = false;
        boolean firstCol = false;
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
             if (matrix[i][j] == 0) {
                 if (i == 0) {
                     firstRow = true;
                 }
                 if (j == 0) {
                     firstCol = true;
                 }
                 matrix[i][0] = 0;
                 matrix[0][j] = 0;
             }
            }
         }
         for (int i = 1; i < row; i++) {
             for (int j = 1; j < col; j++) {
                 if (matrix[i][0] == 0 || matrix[0][j] == 0) {
                     matrix[i][j] = 0;
                 }
             }
         }
         if (firstRow) {
             for (int i = 0; i < matrix[0].length; i++) {
                 matrix[0][i] = 0;
             }
         }
         if (firstCol) {
             for (int i = 0; i < matrix.length; i++) {
                 matrix[i][0] = 0;
             }
         }
    }
```

Verify Preorder Serialization of Binary Tree

```
public boolean isValidSerialization(String preorder) {
        Stack<String> stack = new Stack<String>();
        String arr[] = preorder.split(",");
        for (int i = 0; i < arr.length; i++) {
            String curr = arr[i];
            while (curr.equals("#") && !stack.isEmpty() && stack.peek().equals(curr)) {
                stack.pop();
                if (stack.isEmpty()) {
                    return false;
                }
                stack.pop();
            }
            stack.push(curr);
        }
```

```
        return stack.size() == 1 && stack.peek().equals("#");
    }
```

Remove Duplicates from sorted array ii

```
public int removeDups(int nums[]) {
  int i = 1;
  int j = 1;
  int count = 1;
  while (j < nums.length) {
    if (nums[j] != nums[j-1]) {
      count = 1;
      nums[i] = nums[j];
      i++;
    } else {
      if (count < k) {
        count++;
        nums[i] = nums[j];
        i++;
      }
    }
  }
  return i;
```

House Robber II - Maximum of non-adjacent elements if the numbers are in a circle

0 and n-1 are neighbours so max of 0 – n-2 , 1 - n-1

```
public int rob(int[] nums) {
        if (nums.length == 1) {
            return nums[0];
        }
        return Math.max(rob(nums, 0, nums.length - 2), rob(nums, 1, nums.length - 1));
    }
    public int rob(int[] nums, int left, int right) {
        int exc = 0;
        int inc = 0;
        for (int i = left; i <= right; i++) {
            int old_inc = inc;
            inc = exc + nums[i];
            exc = Math.max(exc, old_inc);
        }
        return Math.max(exc, inc);
    }
```

Sum Root to Leaf

```
public class Solution {
    public int sumNumbers(TreeNode root) {
        return sum(root, 0);
    }
    public int sum(TreeNode root, int s) {
        if (root == null) {
            return 0;
        }
        if (root.left == null && root.right == null) {
            return s * 10 + root.val;
        }
        return sum(root.left, s * 10 + root.val) + sum(root.right, s * 10 + root.val);
    }
}
```

Longest Absolute File Path

```
  public int lengthLongestPath(String input) {
        if (input.length() == 0) {
            return 0;
        }
        int len = 0;
        int longest = 0;
        Stack<Integer> stack = new Stack<Integer>();
        String arr[] = input.split("\n");
        for (int i = 0; i < arr.length; i++) {
            String str = arr[i];
            String tempStr = str.replaceAll("\t","");
            int level = str.length() - tempStr.length();
```

```
            while (stack.size() > level) {
                len -= stack.pop();
            }
            len += tempStr.length() + 1;

            if (str.contains(".")) {
                if (len - 1 > longest) {
                    longest = len - 1;
                }
            }
            stack.push(tempStr.length() + 1);
        }
        return longest;
    }
```

Wiggle Subsequence

```
public int wiggleMaxLength(int[] nums) {
        if (nums.length <= 1) {
            return nums.length;
        }
        int k = 0;
        while (k < nums.length - 1 && nums[k] == nums[k+1]) {
            k++;
        }
        if (k == nums.length - 1) {
            return 1;
        }
        boolean isInc = false;
        if (nums[k] < nums[k + 1]) {
            isInc = true;
        }
        int len = 2;
        for (int i = k + 1; i < nums.length - 1; i++) {
            if (isInc && nums[i] > nums[i + 1]) {
                len++;
                isInc = !isInc;
            }else if (!isInc && nums[i] < nums[i + 1]) {
                len++;
                isInc = !isInc;
            }
        }
        return len;
    }
```

H-index ii - Sorted citations

```
public int hIndex(int[] citations) {
              if(citations == null || citations.length == 0) return 0;

        int left = 0;
        int right = citations.length - 1;
        while (left <= right) {
            int mid = (left + right)/2;
            if (citations[mid] == citations.length - mid) {
                return citations.length - mid;
            } else if (citations[mid] > citations.length - mid) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }
        return citations.length - (right + 1);
    }
```

Path Sum ii

List of paths equal to a sum

```
public List<List<Integer>> pathSum(TreeNode root, int sum) {
    List<List<Integer>>ret = new ArrayList<List<Integer>>();
    List<Integer> cur = new ArrayList<Integer>();
    pathSum(root, sum, cur, ret);
```

```
    return ret;
}

public void pathSum(TreeNode root, int sum, List<Integer>cur, List<List<Integer>>ret){
    if (root == null){
        return;
    }
    cur.add(root.val);
    if (root.left == null && root.right == null && root.val == sum){
        ret.add(new ArrayList(cur));
    }else{
        pathSum(root.left, sum - root.val, cur, ret);
        pathSum(root.right, sum - root.val, cur, ret);
    }
    cur.remove(cur.size()-1);
}
```

Unique Substrings in Wraparound String

```
public int findSubstringInWraproundString(String p) {
        // count[i] is the maximum unique substring end with ith letter.
        // 0 - 'a', 1 - 'b', ..., 25 - 'z'.
        int[] count = new int[26];

        // store longest contiguous substring ending at current position.
        int maxLengthCur = 0;

        for (int i = 0; i < p.length(); i++) {
          if (i > 0 && (p.charAt(i) - p.charAt(i - 1) == 1 || (p.charAt(i - 1) - p.charAt(i) ==25))) {
                maxLengthCur++;
            }
            else {
                maxLengthCur = 1;
            }

            int index = p.charAt(i) - 'a';
            count[index] = Math.max(count[index], maxLengthCur);
        }

        // Sum to get result
        int sum = 0;
        for (int i = 0; i < 26; i++) {
            sum += count[i];
        }
        return sum;
    }
```

Gas Station

GasSum > cost sum valid solution. If A cannot reach B then B is the next start station

```
        int gasSum = 0;
        int costSum = 0;
        int total = 0;
        int start = 0;
        for (int i = 0; i < gas.length; i++) {
            gasSum += gas[i];
            costSum += cost[i];
            total += gas[i] - cost[i];
            if (total < 0) {
                start = i + 1;
                total = 0;
            }
        }
        if (gasSum < costSum) {
          return -1;
        } else {
          return start;      }
```

Unique Binary Search Trees - Number of subtrees that can be formed by n

G(i) = G(i-1) * G(n – i)

F(i, **N**) = G(i-1) * G(**N**-i)

```
public int numTrees(int n) {
    int [] G = new int[n+1];
    G[0] = G[1] = 1;

    for(int i=2; i<=n; ++i) {
           for(int j=1; j<=i; ++j) {
                G[i] += G[j-1] * G[i-j];
           }
    }

    return G[n];
}
```

132 Pattern

```
public boolean find132pattern(int[] nums) {
        Stack<Integer> stack = new Stack<Integer>();
        int s3 = Integer.MIN_VALUE;
        for (int i = nums.length - 1; i >= 0 ; i--) {
          if (nums[i] < s3) {
              return true;
          } else {
              while (!stack.isEmpty() && nums[i] > stack.peek()) {
                  s3 = stack.pop();
              }
          }
          stack.push(nums[i]);
         }
         return false;
        }
```

Water and JUG – GCD

```
public boolean canMeasureWater(int x, int y, int z) {
        if (x + y < z) {
            return false;
        }
        if (x == z || y == z || x + y == z) {
            return true;
        }
        return z % GCD(x, y) == 0;
    }
    public int GCD(int x, int y) {
        while (y != 0) {
            int temp = y;
            y = x % y;
            x = temp;
        }
        return x;
    }
```

Permutation sequence

```
public String getPermutation(int n, int k) {

        int fact[] = new int[n + 1];
        fact[0] = 1;
        for (int i = 1; i <= n; i++) {
            fact[i] = i * fact[i - 1];
        }
        List<Integer> list = new ArrayList<Integer>();
        for (int i = 1; i <= n; i++) {
            list.add(i);
        }
        k--;
        StringBuilder sb = new StringBuilder();
        for (int i = 1; i <= n; i++) {
            int index = k/fact[n - i];
            int num = list.get(index);
            sb.append(num);
            list.remove(index);
            k = k - index * fact[n - i];
        }
        return sb.toString();
    }
```

Word Search – O(m*n*len(word))

```
public class Solution {
    static boolean[][] visited;
    public boolean exist(char[][] board, String word) {
        visited = new boolean[board.length][board[0].length];

        for(int i = 0; i < board.length; i++){
            for(int j = 0; j < board[i].length; j++){
                if((word.charAt(0) == board[i][j]) && search(board, word, i, j, 0)){
                    return true;
                }
            }
        }

        return false;
    }

    private boolean search(char[][]board, String word, int i, int j, int index){
        if(index == word.length()){
            return true;
        }

        if(i >= board.length || i < 0 || j >= board[i].length || j < 0 || board[i][j] !=
word.charAt(index) || visited[i][j]){
            return false;
        }

        visited[i][j] = true;
        if(search(board, word, i-1, j, index+1) ||
           search(board, word, i+1, j, index+1) ||
           search(board, word, i, j-1, index+1) ||
           search(board, word, i, j+1, index+1)){
            return true;
        }

        visited[i][j] = false;
        return false;
    }
}
```

Clone Graph

```
public HashMap<Integer, UndirectedGraphNode> map = new HashMap();
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        if (node == null) {
            return null;
        }
        if (map.containsKey(node.label)) {
            return map.get(node.label);
        }
        UndirectedGraphNode clonedNode = new UndirectedGraphNode(node.label);
        map.put(clonedNode.label, clonedNode);
        for (int i = 0; i < node.neighbors.size(); i++) {
            clonedNode.neighbors.add(cloneGraph(node.neighbors.get(i)));
        }
        return clonedNode;
    }
```

Missing Range

[0, 1, 3, 50, 75], return [“2”, “4->49”, “51->74”, “76->99”]

```
public List<String> findMissingRanges(int[] a, int lo, int hi) {
  List<String> res = new ArrayList<String>();

  // the next number we need to find
  int next = lo;

  for (int i = 0; i < a.length; i++) {
    // not within the range yet
    if (a[i] < next) continue;

    // continue to find the next one
```

```
    if (a[i] == next) {
      next++;
      continue;
    }

    // get the missing range string format
    res.add(getRange(next, a[i] - 1));

    // now we need to find the next number
    next = a[i] + 1;
  }

  // do a final check
  if (next <= hi) res.add(getRange(next, hi));

  return res;
}

String getRange(int n1, int n2) {
  return (n1 == n2) ? String.valueOf(n1) : String.format("%d->%d", n1, n2);
}
```

Summary Range

[0,1,2,4,5,7], return ["0->2","4->5","7"]

```
public List<String> summaryRanges(int[] nums) {
        List<String> list = new ArrayList<String>();
        if (nums.length == 0) {
            return list;
        }
        if (nums.length == 1) {
            list.add(""+nums[0]);
            return list;
        }
        int start = 0;
        for (int i = 0; i < nums.length; i++) {
            if (i  == nums.length - 1 || nums[i] + 1 != nums[i + 1]) {
                String str = "" + nums[start] ;
                if (i != start) {
                    str += "->" + nums[i];
                }
                list.add(str);
                start = i  + 1;
                }
            }
            return list;

        }
    }
```

Best Meeting Point

```
public int minTotalDistance(int[][] grid) {
    List<Integer> rows = collectRows(grid);
    List<Integer> cols = collectCols(grid);
    return minDistance1D(rows) + minDistance1D(cols);
}

private int minDistance1D(List<Integer> points) {
    int distance = 0;
    int i = 0;
    int j = points.size() - 1;
    while (i < j) {
        distance += points.get(j) - points.get(i);
        i++;
        j--;
    }
    return distance;
}
private List<Integer> collectRows(int[][] grid) {
    List<Integer> rows = new ArrayList<>();
```

```
    for (int row = 0; row < grid.length; row++) {
        for (int col = 0; col < grid[0].length; col++) {
            if (grid[row][col] == 1) {
                rows.add(row);
            }
        }
    }
    return rows;
}

private List<Integer> collectCols(int[][] grid) {
    List<Integer> cols = new ArrayList<>();
    for (int col = 0; col < grid[0].length; col++) {
        for (int row = 0; row < grid.length; row++) {
            if (grid[row][col] == 1) {
                cols.add(col);
            }
        }
    }
    return cols;
}
```

Find Duplicate Numbers
Given an array nums containing n + 1 integers where each integer is between 1 and n (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

```
public int findDuplicate(int[] nums) {
        if (nums.length == 0) {
            return -1;
        }
        int slow = nums[0];
        int fast = nums[nums[0]];
        while (slow != fast) {
            slow = nums[slow];
            fast = nums[nums[fast]];
        }
        fast = 0;
        while (fast != slow) {
            slow = nums[slow];
            fast = nums[fast];
        }
        return slow;
    }
```

TreeMap get and containsKey(Ologn)

Find longest increasing path in a matrix

```
public int longestIncreasingPath(int[][] matrix) {
        if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
            return 0;
        }
        int[][] cache = new int[matrix.length][matrix[0].length];
        int max = 0;
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                int length = findSmallAround(i, j, matrix, cache, Integer.MAX_VALUE);
                max = Math.max(length, max);
            }
        }
        return max;
    }
    private int findSmallAround(int i, int j, int[][] matrix, int[][] cache, int pre) {
        // if out of bond OR current cell value larger than previous cell value.
        if (i < 0 || i >= matrix.length || j < 0 || j >= matrix[0].length || matrix[i][j] >= pre) {
            return 0;
```

```
        }
        // if calculated before, no need to do it again
        if (cache[i][j] > 0) {
            return cache[i][j];
        } else {
            int cur = matrix[i][j];
            int tempMax = 0;
            tempMax = Math.max(findSmallAround(i - 1, j, matrix, cache, cur), tempMax);
            tempMax = Math.max(findSmallAround(i + 1, j, matrix, cache, cur), tempMax);
            tempMax = Math.max(findSmallAround(i, j - 1, matrix, cache, cur), tempMax);
            tempMax = Math.max(findSmallAround(i, j + 1, matrix, cache, cur), tempMax);
            cache[i][j] = ++tempMax;
            return tempMax;
        }
      }
```

Find cycle in a graph

Course Schedule

```
public class Solution {
        public boolean canFinish(int numCourses, int[][] prerequisites) {
            ArrayList[] graph = new ArrayList[numCourses];
            for(int i=0;i<numCourses;i++)
                graph[i] = new ArrayList();


            boolean[] visited = new boolean[numCourses];
            for(int i=0; i<prerequisites.length;i++){
                graph[prerequisites[i][1]].add(prerequisites[i][0]);
            }

            for(int i=0; i<numCourses; i++){
                if(!dfs(graph,visited,i))
                    return false;
            }
            return true;
        }

        private boolean dfs(ArrayList[] graph, boolean[] visited, int course){
            if(visited[course])
                return false;
            else
                visited[course] = true;;

            for(int i=0; i<graph[course].size();i++){
                if(!dfs(graph,visited,(int)graph[course].get(i)))
                    return false;
            }
            visited[course] = false;
            return true;
        }
```

Longest Consecutive Sequence

Given [100, 4, 200, 1, 3, 2],The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.

```
public int longestConsecutive(int[] nums) {
         Set<Integer> set = new HashSet<>();
            for(int n : nums) {
                set.add(n);
            }
            int best = 0;
            for(int n : set) {
                if(!set.contains(n - 1)) {  // only check for one direction
                    int m = n + 1;
                    while(set.contains(m)) {
                        m++;
                    }
                    best = Math.max(best, m - n);
                }
            }
            return best;
    }
```

Split Array Largest Sum

```
public int splitArray(int[] nums, int m) {
        if (nums.length == 0) {
```

```
            return 0;
        }
        int max = 0;
        int sum = 0;
        for (int num : nums) {
          max = Math.max(max, num);
          sum += num;
        }
        if (m == 1) {
            return sum;
        }
        int left = max;
        int right = sum;
        while (left <= right) {
          int mid = (left + right)/2;
          if (valid(nums, m, mid)) {
            right = mid - 1;
          } else {
            left = mid + 1;
          }
        }
        return left;
    }
    public boolean valid(int[] nums, int m, int mid) {
        int total = 0;
        int count = 1;
        for (int num : nums) {
            total += num;
            if (total > mid) {
              total = num;
              count++;
              if (count > m) {
                  return false;
              }
            }
        }
        return true;
    }
```

Self Crossing

```
  public boolean isSelfCrossing(int[] x) {
        for (int i = 3; i < x.length; i++) {
            if (x[i] >= x[i-2] && x[i-1] <= x[i-3]) {
                return true;
            }
            else if (i >=4  && x[i-1] == x[i-3] && x[i] + x[i-4] >= x[i-2]) {
                return true;
            } else if (i >= 5 && x[i-2] >= x[i-4] && x[i] + x[i-4] >= x[i-2] && x[i-1] + x[i-5] >=
x[i-3] && x[i-1] <= x[i-3]) {
                return true;
            }
        }
    return false;
```

First Missing Positive

```
public int firstMissingPositive(int[] nums) {
        int len = nums.length;
        for (int i = 0; i < len; i++) {
            while (nums[i] > 0 && nums[i] <= len && nums[nums[i] - 1] != nums[i]) {
                swap(nums, i, nums[i] - 1);


            }
        }
        for (int i = 0; i < len; i++) {
            if (nums[i] != i + 1) {
                return i + 1;
            }
        }
        return len + 1;
    }
```

Maximum Number of Points that lie on a straight line

```
public int maxPoints(Point[] points) {
        if(points.length <= 0) return 0;
```

```
        if(points.length <= 2) return points.length;
        int result = 0;
        for(int i = 0; i < points.length; i++){
            HashMap<Double, Integer> hm = new HashMap<Double, Integer>();
            int samex = 1;
            int samep = 0;
            for(int j = 0; j < points.length; j++){
                if(j != i){
                    if((points[j].x == points[i].x) && (points[j].y == points[i].y)){
                        samep++;
                    }
                    if(points[j].x == points[i].x){
                        samex++;
                        continue;
                    }
                    double k = (double)(points[j].y - points[i].y) / (double)(points[j].x -
points[i].x);
                    if(hm.containsKey(k)){
                        hm.put(k,hm.get(k) + 1);
                    }else{
                        hm.put(k, 2);
                    }
                    result = Math.max(result, hm.get(k) + samep);
                }
            }
            result = Math.max(result, samex);
        }
        return result;
    }
```

Paint Houses

```
public int minPaintCost(int[][] cost) {
    if (cost == null || cost.length == 0) return 0;
    int[][] dp = new int[cost.length][3];
    dp[0][0] = cost[0][0], dp[0][1] = cost[0][1], dp[0][2] = cost[0][2];
    for (int i = 1; i < cost.length; ++i) {
        dp[i][0] = cost[i][0] + Math.min(dp[i-1][1], dp[i-1][2]);
        dp[i][1] = cost[i][1] + Math.min(dp[i-1][0], dp[i-1][2]);
        dp[i][2] = cost[i][2] + Math.min(dp[i-1][0], dp[i-1][1]);
    }
    return Math.min(dp[dp.length-1][0], Math.min(dp[dp.length-1][1],[dp.length-1][2]));
}
```

Paint Fences

Binary Tree Paths from Root to Leaf

```
public List<String> binaryTreePaths(TreeNode root) {
        List<String> result = new ArrayList<String>();
        if (root == null) {
            return result;
        }
        binaryTreePaths(root, "", result);
        return result;
    }
    public void binaryTreePaths(TreeNode root, String path, List<String> result) {

        if (root.left == null && root.right == null) {
            result.add(path + root.val);
        }
        if (root.left != null) {
            binaryTreePaths(root.left, path + root.val + "->", result);
        }
        if (root.right != null) {
            binaryTreePaths(root.right, path + root.val + "->", result);
        }
    }
```

#Valid Word Abbrevation

```
public static boolean isValidAbbr(String word, String abbr) {
  int i = 0;
  int j = 0;
  int wLen = word.length();
  int abbrLen = abbr.length();
  // apple - a2le
```

```
  while (i < wLen && j < abbrLen) {
    if (word.charAt(i) == abbr.charAt(j) {
      i++;
      j++;
    } else if (abbr.charAt(j) >= '0' && abbr.charAt(j) <= '9') {
      int k = j + 1;
      while (Character.isDigit(abbr.charAt(k)) {
        k++;
      }
      int len = Integer.valueOf(abbr.substring(j, k));
      i = i + len;
      j = k;
   } else {
     return false;
   }
   if (i == wLen && j == abbrLen) {
     return true;
   }
}
```

One Edit Distance

```
#One Edit Distance
# Two lengths equal length
public boolean isOneEditDistance(String s, String t) {
  int m = s.length();
  int n = t.length();
  int len = Math.min(m, n);
  for (int i = 0; i < len; i++) {
    if (s.charAt(i) != t.charAt(i)) {
      if (s.substring(i + 1).equals(t.substring(i + 1))) {
        return true;
       }
      } else {
      if (m > n) {
        return s.substring(i + 1).equals(t.substring(i));
      }
      if (n > m) {
        return t.substring(i + 1).equals(s.substring(i));
      }
    }
  }
}
return (Math.abs(m - n) == 1);
```

Factor Combinations
http://buttercola.blogspot.com/2015/08/leetcode-factor-combinations.html

Merge Two Strings

```
public static void mergeTwoStrings(String s1, String s2, int i, int j, String curr, List<String>
result) {
                if (curr.length() == s1.length() + s2.length()) {
                        result.add(curr);
                        return;
                }
                if (i < s1.length()) {
                        mergeTwoStrings(s1, s2, i + 1, j, curr + s1.charAt(i), result);
                }
                if (j < s2.length()) {
                        mergeTwoStrings(s1, s2, i, j + 1, curr + s2.charAt(j), result);
                }
        }
```

Binet's Formula Fibonacci numbers less than n

Minimum Total in a Triangle

```
public int minimumTotal(List<List<Integer>> triangle) {
        int min[] = new int[triangle.size() + 1];
        for (int i = triangle.size() - 1; i >= 0; i--) {
            for (int j = 0; j <= i; j++) {
                min[j] = Math.min(min[j + 1], min[j]) + triangle.get(i).get(j);
            }
        }
        return min[0];
```

```
    }
```

Given unsigned integer 'x', write an algorithm thet returns unsigned integer 'y' such that it has the same number of bits set as 'x' and is not equal to 'x' and the distance |x-y| is minimized.

Exchange lowest order 1 and 0

Count Triangles in a graph

In directed graph 3 permuations in undirected graph 6 permutations.

Three for loops I, j, k.

if (graph[i][j] && graph[j][k] && graph[k][i]) { count++

}

Lagrange's Theorem

The sum can be formed by sum of upto 4 squares of integers

4*k * (8m + 7) then 4. N- I * I = 2 and remaining 3

Path Sum II - Print the tree having a root to leaf path adding to a target value

```
  public List<List<Integer>> pathSum(TreeNode root, int sum) {
        List<Integer> list = new ArrayList<Integer>();
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        if (root == null) {
            return result;
        }
        pathSum(root, sum, list, result);
        return result;
    }
    public void pathSum(TreeNode root, int sum, List<Integer> list, List<List<Integer>> result) {
        if (root == null) {
            return;
        }
        list.add(root.val);
        if (root.left == null && root.right == null) {
            if (sum == root.val) {
                result.add(new ArrayList(list));
            }
        } else {
            pathSum(root.left, sum - root.val, list, result);
            pathSum(root.right, sum - root.val, list, result);
        }
        list.remove(list.size() - 1);
    }
```

Sum of numbers smaller than N and divisible by 3 and 5

```
public int SumOfNumber(int N){
        N--;
           int divisibleBy3 = N/3;
           divisibleBy3 = (divisibleBy3 * (divisibleBy3 + 1) /2)*3;

           int divisibleBy5 = N/5;
           divisibleBy5 = (divisibleBy5 * (divisibleBy5 + 1) /2)*5;

           int divisibleBy15 = N/15;
           divisibleBy15 = (divisibleBy15 * (divisibleBy15 + 1) /2)*15;


           return divisibleBy3 + divisibleBy5 - divisibleBy15
}
```

Longest Increasing Continuous Subsequence

```
public int longestIncreasingContinuousSubsequence(int[] A) {
        // Write your code here
        if (A.length == 0) {
            return 0;
        }
        if (A.length == 1) {
            return 1;
        }
        if (A.length == 2) {
            if (A[0] != A[1]) {
                return 2;
            }
            return 1;
        }
        int length = 1;
```

```
        int maxLength = Integer.MIN_VALUE;
        for (int i = 1; i < A.length; i++) {
            if (A[i] > A[i - 1] ) {
                length++;
                maxLength = Math.max(maxLength, length);
            } else {
                length = 1;
            }
        }
        length = 1;
        for (int i = A.length - 1; i > 0; i--) {
            if (A[i] < A[i - 1] ) {
                length++;
                maxLength = Math.max(maxLength, length);
            } else {
                length = 1;
            }
        }
        return maxLength;
    }
}
```

Find the missing element in O(n) and O(1) space

```
  public int findElement(int a[],n) {
     int x1=a[0];
     int x2=0;
     for(int i=1;i<a.length;i++) {
         x1 = x1 ^ a[i];
     }
     for(int i=1;i<= n;i++) {
         x2 = x2 ^ i;
     }
     return (x1 ^ x2)

 }
```

Find minimum in sorted rotated array wth duplicates

```
while (low < high) {
            if (nums[low] < nums[high]) {
              return nums[low];
            }
            int mid = low + (high - low)/2;
            if (nums[mid] >= nums[low]) {
                low = mid + 1;
            } else {
                high = mid;
            }
        }
        return nums[low];
```

Find Peak element

```
while (left <= right) {
            int mid = left + (right - left) / 2;
            if ((mid == 0 || nums[mid] >= nums[mid - 1]) && (mid == nums.length - 1 || nums[mid] >=
nums[mid + 1])) {
                return mid;
            } else if (mid > 0 && nums[mid] < nums[mid - 1]) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }
```

Finding Celebrity

Letter Combinations of a Phone Number

```
public List<String> letterCombinations(String digits) {
    List<String> result = new ArrayList<String>();

    if (digits.length() == 0) {
        return result;
    }
```

```
    HashMap<Character, char[]> map = new HashMap<Character, char[]>();
    map.put('2', new char[]{'a','b','c'});
    map.put('3', new char[]{'d','e','f'});
    map.put('4', new char[]{'g','h','i'});
    map.put('5', new char[]{'j','k','l'});
    map.put('6', new char[]{'m','n','o'});
    map.put('7', new char[]{'p','q','r','s'});
    map.put('8', new char[]{'t','u','v'});
    map.put('9', new char[]{'w','x','y','z'});



    letterComb(result, new StringBuilder(), digits, 0, map);

    return result;

    }
    public void letterComb(List<String> result, StringBuilder sb, String digits, int index,
HashMap<Character, char[]> map) {
        if (sb.length() >= digits.length()) {
            result.add(sb.toString());
            return;
        }
        char ch = digits.charAt(index);
        char arr[] = map.get(ch);
        for (int i = 0; i < arr.length; i++) {
            sb.append(arr[i]);
            letterComb(result, sb, digits, index + 1, map);
            sb.deleteCharAt(sb.length() - 1);
        }
     }
```

Permutations

```
public void permuate(int[] nums, List<List<Integer>> result, List<Integer> curr) {
        if (curr.size() == nums.length) {
            result.add(new ArrayList<>(curr));
            return;
        }
        for (int i = 0; i < nums.length; i++) {
            if (curr.contains(nums[i])) {
                continue;
            }
            curr.add(nums[i]);
            permuate(nums, result, curr);
            curr.remove(curr.size() - 1);
        }
    }
```

https://discuss.leetcode.com/topic/46162/a-general-approach-to-backtracking-questions-in-java-subsets-permutations-combination-sum-palindrome-partioning/2

HashSet is not synchronized. So use Collections.synchronizedSet(new HashSet());
Thread safe arraylist is CopyOnWriteArrayList

Binary Tree Maximum Path Sum

```
public int maxPathSum(TreeNode root) {
      value = Integer.MIN_VALUE;
      maxPath(root);
      return value;
    }
    private int maxPath(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int left = Math.max(0, maxPath(root.left));
        int right = Math.max(0, maxPath(root.right));
        value = Math.max(value, left + right + root.val);
        return Math.max(left, right) + root.val;
    }
```

Subset Powerset

```
public List<List<Integer>> subsets(int[] nums) {
```

```
    List<List<Integer>> list = new ArrayList<>();
    Arrays.sort(nums);
    backtrack(list, new ArrayList<>(), nums, 0);
    return list;
}

private void backtrack(List<List<Integer>> list , List<Integer> tempList, int [] nums, int start){
    list.add(new ArrayList<>(tempList));
    for(int i = start; i < nums.length; i++){
        tempList.add(nums[i]);
        backtrack(list, tempList, nums, i + 1);
        tempList.remove(tempList.size() - 1);
    }
}
```

Remove Duplicate Letters O(hn)

```
public String removeDuplicateLetters(String s) {
        HashMap<Character, Integer> map = new HashMap<Character, Integer>();
        for (int i = 0; i < s.length(); i++) {
            map.put(s.charAt(i), i);
        }
        StringBuilder sb = new StringBuilder();
        int start = 0;
        int end = minValue(map);
        while (!map.isEmpty()) {
            char minChar = 'z';
            for (int i = start; i <= end; i++) {
                char ch = s.charAt(i);
                if (ch < minChar && map.containsKey(ch)) {
                    minChar = ch;
                    start = i + 1;
                }
            }
            sb.append(minChar);
            map.remove(minChar);
            end = minValue(map);
        }
        return sb.toString();
    }
    public int minValue(HashMap<Character, Integer> map) {
        int val = Integer.MAX_VALUE;
        for(int i : map.values()) {
            val = Math.min(val, i);
        }
        return val;
    }
```

Reverse Linked List Recursion

```
public ListNode reverse(ListNode head) {
        // write your code here
        if ((head == null) || (head.next == null)) {
            return head;
        }
        ListNode ret = reverse(head.next);
        head.next.next = head;
        head.next = null;
        return ret;
    }
```

Linked List Reverse Between m to n nodes
Reverse a linked list from m to n

```
public ListNode reverseBetween(ListNode head, int m, int n) {
   if(head == null) return null;
    ListNode dummy = new ListNode(0); // create a dummy node to mark the head of this list
    dummy.next   = head;
    ListNode pre = dummy; // make a pointer pre as a marker for the node before reversing
    for(int i = 0; i<m-1; i++) pre = pre.next;
    ListNode start = pre.next; // a pointer to the beginning of a sub-list that will be reversed
    ListNode then = start.next; // a pointer to a node that will be reverse
    // 1 - 2 -3 - 4 - 5 ; m=2; n =4 ---> pre = 1, start = 2, then = 3
    // dummy-> 1 -> 2 -> 3 -> 4 -> 5
    for(int i=0; i<n-m; i++) {
        start.next = then.next;
        then.next = pre.next;
        pre.next = then;
```

```
        then = start.next;
    }
        return dummy.next;

    // first reversing : dummy->1 - 3 - 2 - 4 - 5; pre = 1, start = 2, then = 4
    // second reversing: dummy->1 - 4 - 3 - 2 - 5; pre = 1, start = 2, then = 5 (finish)
}
```

Balanced Binary Tree

```
int diff = Math.abs(left - right);
        if (diff <= 1 && isBalanced(root.left) && isBalanced(root.right)) {
            return true;
        }
```

Maximum Subarray II – Lintcode

Given an array of integers, find two non-overlapping subarrays which have the largest sum.

```
public int maxTwoSubArrays(ArrayList<Integer> nums) {
        // write your code
        if (nums.size() == 1) {
            return nums.get(0);
        }
        if (nums.size() == 0) {
            return 0;
        }
        int len = nums.size();
        int left[] = new int[len];
        int right[] = new int[len];
        leftSubArray(nums, left);
        rightSubArray(nums, right);
        int max  = Integer.MIN_VALUE;
        for (int i = 0; i < nums.size() - 1; i++) {
            max = Math.max(max, left[i] + right[i + 1]);
        }
        return max;
    }
    public void leftSubArray(ArrayList<Integer> nums, int[] arr) {
        int curr = nums.get(0);
        int maxSoFar = nums.get(0);
        arr[0] = maxSoFar;
        for (int i = 1; i < nums.size(); i++) {
            curr = Math.max(nums.get(i), curr + nums.get(i));
            maxSoFar = Math.max(maxSoFar, curr);
            arr[i] = maxSoFar;
        }
   }
    public void rightSubArray(ArrayList<Integer> nums, int[] arr) {
        int len = nums.size() - 1;
        int curr = nums.get(len);
        int maxSoFar = nums.get(len);
        arr[len] = maxSoFar;
        for (int i = len - 1; i >= 0; i--) {
            curr = Math.max(nums.get(i), curr + nums.get(i));
            maxSoFar = Math.max(maxSoFar, curr);
            arr[i] = maxSoFar;
        }
    }
```

Delete Node in BST

DeleteBSt Delete BST

```
public TreeNode removeNode(TreeNode root, int value) {
        // write your code here
        if (root == null) {
            return null;
        }
        if (value < root.val) {
            root.left = removeNode(root.left, value);
        } else if (value > root.val) {
            root.right = removeNode(root.right, value);
        } else {
            if (root.left == null) {
                return root.right;
            } else if (root.right == null) {
                return root.left;
            } else {
```

```
                root.val = FindMin(root.right).val;
                root.right = removeNode(root.right, root.val);
            }
        }
        return root;
    }
    public TreeNode FindMin(TreeNode node) {
        if (node == null) {
            return null;
        }
        while (node.left != null) {
            node = node.left;
        }
        return node;
    }
```

Nth Ugly Number Prime factors include only 2, 3, 5

```
public int nthUglyNumber(int n) {
        int ugly2 = 2;
        int ugly3 = 3;
        int ugly5 = 5;
        int i = 0;
        int j = 0;
        int k = 0;
        int ugly[] = new int[n];
        int nextUgly = 1;
        ugly[0] = 1;
        for (int l = 1; l < n; l++) {
            nextUgly = Math.min(ugly2, Math.min(ugly3, ugly5));
            ugly[l] = nextUgly;
            if (nextUgly == ugly2) {
                i++;
                ugly2 = ugly[i] * 2;
            }
            if (nextUgly == ugly3) {
                j++;
                ugly3 = ugly[j] * 3;
            }
            if (nextUgly == ugly5) {
                k++;
                ugly5 = ugly[k] * 5;
            }
        }
        return ugly[n - 1];
    }
```

Matrix Zigzag Traversal

```
public int[] printZMatrix(int[][] matrix) {
    int m = matrix.length;
    int n = matrix[0].length;
    int[] result = new int[m * n];
    int i, j;
    int index = 0;
    for (int sum = 0; sum <= m + n - 2; sum++) {
      if (sum % 2 == 0) {
        i = Math.min(sum, m - 1);
        j = sum - i;
        while (i >= 0 && j < n) {
          result[index++] = matrix[i--][j++];
        }
      } else {
        j = Math.min(sum, n - 1);
        i = sum - j;
        while (j >= 0 && i < m) {
          result[index++] = matrix[i++][j--];
        }
      }
    }
    return result;
  }
```

Number of Inversions

```
public class Solution {
    public long reversePairs(int[] A) {
        return mergeSort(A, 0, A.length - 1);
    }
    public int mergeSort(int[] A, int left, int end) {
```

```
        if (left >= end) {
            return 0;
        }
        int count = 0;
        int mid = (left + end) / 2;
        count += mergeSort(A, left, mid);
        count += mergeSort(A, mid + 1, end);
        count += merge(A, left, end);
        return count;
    }
    public int merge(int[] A, int left, int end) {
        int count = 0;
        int temp[] = new int[A.length];
        int mid  = (left + end)/ 2;
        int i = left;
        int j = mid + 1;
        int k = left;
        while (i <= mid && j <= end) {
            if (A[i] <= A[j]) {
                temp[k] = A[i];
                i++;
            } else {
                temp[k] = A[j];
                count += mid - i + 1;
                j++;
            }
            k++;
        }
        while (i <= mid) {
            temp[k] = A[i];
            i++;
            k++;
        }
        while (j <= end) {
            temp[k] = A[j];
            j++;
            k++;
        }
        for (i = left; i <= end; i++) {
            A[i] = temp[i];
        }
        return count;}}
```

Maximum Continuous Subarray sum Indexes - Kadence

```
int sum = 0;
        int start = 0;
        int end = 0;
        int max = Integer.MIN_VALUE;
        for (int i = 0; i < A.length; i++) {
            if (sum >= 0) {
                sum += A[i];
                end = i;
            } else if (sum < 0) {
                sum = A[i];
                start = i;
                end = i;
            }
            if (sum >= max) {
                max = sum;
                list.set(0, start);
                list.set(1, end);
            }
        }
        return list;
```

Number of Islands - O(nm)

```
function islandCounter(M):
   islands = 0
   if (M == null OR M.length == 0):
      return 0
   m = M.length # number of rows
   n = M[0].length # number of columns
for i from 0 to m-1
      for j from 0 to n-1
         if (M[i][j] == 1):
```

```
            markIsland(M, m, n, i, j)
            islands++
function markIsland(M, m, n, i, j):
   q = new Queue
   q.push([i,j])
   while (!q.isEmpty()):
      item = q.pop()
      x = item[0]
      y = item[1]
      if (M[x][y] == 1):
         M[x][y] = 2
         pushIfValid(q, m, n, x-1, y)
         pushIfValid(q, m, n, x, y-1)
         pushIfValid(q, m, n, x+1, y)
         pushIfValid(q, m, n, x, y+1)

function pushIfValid(q, m, n, x, y):
   if (x>=0 AND x<m AND y>=0 AND y<n):
      q.push([x,y])
```

Shortest Palindrome
Minimum Insertions to form a Palindrome

Str1
Create reverse of str1.
Take Longest Common Subsequence of str1 and str2.
Length of str1 – return of lcs = minimum number of insertions to form a palindrome

Rectangle Overlap

```
if ((x1 > x4 || x3 >= x2) && (y1 >= y4 || y3 >=  y2)) {
            System.out.println("no overlap");
         } else {
            System.out.println("overlap");
         }
```

Encode/ Decode String

```
public String encode(List<String> strs) {
        StringBuilder sb = new StringBuilder();
        for(String s : strs) {
            sb.append(s.length()).append('/').append(s);
        }
        return sb.toString();
    }

    // Decodes a single string to a list of strings.
    public List<String> decode(String s) {
        List<String> ret = new ArrayList<String>();
        int i = 0;
        while(i < s.length()) {
            int slash = s.indexOf('/', i);
            int size = Integer.valueOf(s.substring(i, slash));
            ret.add(s.substring(slash + 1, slash + size + 1));
            i = slash + size + 1;
        }
        return ret;
    }
```

Longest Substring Without Repeating

```
public int lengthOfLongestSubstring(String s) {
         if (s.length() == 0) {
            return 0;
        }
        if (s.length() == 1) {
            return 1;
        }
        int n = s.length();
        int i = 0, j = 0;
        int maxLen = 0;
        int arr[] = new int[256];
        int count = 0;
        while (j < n) {
            char ch = s.charAt(j);
```

```
            arr[ch]++;
            j++;
            if (arr[ch] > 1) {
                count++;
            }
            while (count > 0) {
               char startCh = s.charAt(i);
               arr[startCh]--;
               if (arr[startCh] == 1) {
                   count--;
               }
               i++;
            }
            maxLen = Math.max(maxLen, j - i);
        }
        return maxLen;
    }
```

Longest Substring with K unique characters

```
Count[s[0]]++;
for (int i=1; i<n; i++)
    {
        // Add the character 's[i]' to current window
        count[s[i]-'a']++;
        curr_end++;

        // If there are more than k unique characters in
        // current window, remove from left side
        while (!isValid(count, k))
        {
            count[s[curr_start]-'a']--;
            curr_start++;
        }

        // Update the max window size if required
        if (curr_end-curr_start+1 > max_window_size)
        {
            max_window_size = curr_end-curr_start+1;
            max_window_start = curr_start;
        }
    }
```

Longest Substring with Atmost 2/k different characters

```
public int lengthOfLongestSubstringTwoDistinct(String s) {
        int start = 0, end = 0, counter = 0, len = 0;
        while(end < s.length()){
            char c = s.charAt(end);
            if (arr[c] == 0) {
              counter++;
            }
            arr[c]++;//new char
            end++;
            while(counter > 2){
                char cTemp = s.charAt(start);
                arr[cTemp]--;
                if(arr[cTemp] == 0){
                    counter--;
                }
                start++;
            }
            len = Math.max(len, end-start);
        }
        return len;
    }
```

Sorted Linked List to BST

```
public TreeNode sortedListToBST(ListNode head) {
    if(head==null) return null;
    return toBST(head,null);
}
public TreeNode toBST(ListNode head, ListNode tail){
    ListNode slow = head;
    ListNode fast = head;
    if(head==tail) return null;


    while(fast!=tail&&fast.next!=tail){
```

```
        fast = fast.next.next;
        slow = slow.next;
    }
    TreeNode thead = new TreeNode(slow.val);
    thead.left = toBST(head,slow);
    thead.right = toBST(slow.next,tail);
    return thead;
}
```

Insert Interval

```
public List<Interval> insert(List<Interval> intervals, Interval newInterval) {
        List<Interval> list = new ArrayList<Interval>();
        int i = 0;
        while (i < intervals.size() && intervals.get(i).end < newInterval.start) {
            i++;
        }
        while (i < intervals.size() && intervals.get(i).start <= newInterval.end) {
            newInterval.start = Math.min(intervals.get(i).start, newInterval.start);
            newInterval.end = Math.max(intervals.get(i).end, newInterval.end);
            newInterval = new Interval(newInterval.start, newInterval.end);
            intervals.remove(i); // i++
        }
        intervals.add(i, newInterval); // add to list
        // add remaining elements to list for not in-place solution
        return intervals;
    }
```

Non-Overlapping Intervals

```
public int eraseOverlapIntervals(Interval[] intervals) {
        int len = intervals.length;
        if (len == 0) {
            return 0;
        }
        Arrays.sort(intervals, new Comparator<Interval>() {
           @Override
           public int compare(Interval a, Interval b) {
               return a.end - b.end;
           }
        });
        int count = 1;
        int end = intervals[0].end;
        for (int i = 1; i < len; i++) {
            if (intervals[i].start >= end) {
                count++;
                end = intervals[i].end;
            }
        }
        return len - count;
    }
```

Rotate Matrix – O(n^2)

```
public void rotate(int[][] matrix) {
        if (matrix.length == 0) {
            return;
        }
        int len = matrix.length;
        for (int i = 0; i < len / 2; i++) {
            for (int j = i; j < len - i - 1; j++) {
                int temp = matrix[i][j];
                matrix[i][j] = matrix[len - 1 - j][i];
                matrix[len - 1 -j][i] = matrix[len - 1 - i][len - 1 - j];
                matrix[len - 1 - i][len - 1 - j] = matrix[j][len - 1 - i];
                matrix[j][len - 1 -i] = temp;
            }
        }
    }
```

Sum of Left Leaves

```
int sum = 0;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()) {
            TreeNode temp = queue.remove();
```

```
            if (temp.left != null) {
                if (temp.left.left == null && temp.left.right == null) {
                    sum += temp.left.val;
                }
                queue.add(temp.left);
            }
            if (temp.right != null) {
                queue.add(temp.right);
            }
        }
        return sum;
```

Compare Two Strings – Find the difference in strings

```
public char findTheDifference(String s, String t) {
        int arr1[] = new int[256];
        for (int i = 0; i < s.length(); i++) {
            arr1[s.charAt(i)]++;
        }
        for (int i = 0; i < t.length(); i++) {
            arr1[t.charAt(i)]--;
            if (arr1[t.charAt(i)] < 0) {
                return t.charAt(i); // return false for compare two strings
            }

        }
        return ' ';
    }
```

Next Greater Element in a circular array

```
public int[] nextGreaterElements(int[] nums) {
        int n = nums.length;
        int result[] = new int[nums.length];
        Arrays.fill(result, -1);
        Stack<Integer> stack = new Stack<>();
        for (int i = 0; i < n * 2; i++) {
            int num = nums[i % n];
            while (!stack.isEmpty() && nums[stack.peek()] < num) {
                result[stack.peek()] = num;
                stack.pop();
            }
            if (i < n) {
                stack.push(i);
            }
        }
        return result;
    }
```

Mirror – is Symmetric Tree

```
public boolean isSymmetric(TreeNode root) {
        if (root == null) {
            return true;
        }
        return isSymmetric(root.left, root.right);
    }
    public boolean isSymmetric(TreeNode leftNode, TreeNode rightNode) {
        if (leftNode == null && rightNode == null) {
            return true;
        }

        if (leftNode != null && rightNode != null && leftNode.val == rightNode.val &&
isSymmetric(leftNode.left, rightNode.right) && isSymmetric(leftNode.right, rightNode.left)) {
            return true;
        }
        return false;
    }
```

Plus One

```
public int[] plusOne(int[] digits) {
        int len = digits.length;
```

```
        for (int i = len - 1; i >= 0; i--) {
           if (digits[i] != 9) {
               digits[i]++;
               break;
           } else {
               digits[i] = 0;
           }
        }
        if (digits[0] == 0) {
            int res[] = new int[len + 1];
            res[0] = 1;
            return res;
        }
        return digits;
    }
```

3Sum

```
public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        int len = nums.length;
        if (len < 3) {
            return result;
        }
        Arrays.sort(nums);
        for (int i = 0; i < len - 2; i++) {
            if (i != 0 && nums[i] == nums[i - 1]) {
                continue;
            }
            int left = i + 1;
            int right = len - 1;
            while (left < right) {
                int sum = nums[i] + nums[left] + nums[right];
                if (sum == 0) {
                    List<Integer> list = new ArrayList<>();
                    list.add(nums[i]);
                    list.add(nums[left]);
                    list.add(nums[right]);
                    result.add(list);
                    left++;
                    right--;
                    while (left < len && nums[left] == nums[left - 1]) {
                        left++;
                    }
                    while (left < len && nums[right] == nums[right + 1]) {
                        right--;
                    }
                } else if (sum < 0) {
                    left++;
                } else {
                    right--;
                }
            }
        }
        return result;
    }
```

Intersection of two Linked Lists

```
if (headA == null || headB == null) {
            return null;
        }
        ListNode nodeA = headA;
        ListNode nodeB = headB;
        while (nodeA != null && nodeB != null && nodeA != nodeB) {
            nodeA = nodeA.next;
            nodeB = nodeB.next;
            if (nodeA == nodeB) {
                return nodeA;
            }
            if (nodeA == null) {
                nodeA = headB;
            }
            if (nodeB == null) {
                nodeB = headA;
```

```
            }
        }
        return nodeA;
```

Add Binary - Add to Binary Strings

```
public String addBinary(String a, String b) {
        int aLen = a.length();
        int bLen = b.length();
        int i = aLen - 1;
        int j = bLen - 1;
        int carry = 0;
        StringBuilder sb = new StringBuilder();
        while (i >= 0 || j >= 0) {
            int sum = carry;
            if (i >= 0) {
               sum += Character.getNumericValue(a.charAt(i));
               i--;
            }
            if (j >= 0) {
                sum += Character.getNumericValue(b.charAt(j));
                j--;
            }
            sb.append(sum % 2);
            carry = sum / 2;
        }
        if (carry != 0) {
            sb.append(1);
        }
        return sb.reverse().toString();

    }
```

Sieve Of Erathosthenes- Prime Numbers less than N – O(n * log *logN)

```
public int countPrimes(int n) {
        if (n <= 2) {
            return 0;
        }

        boolean[] isPrime = new boolean[n + 1];
        Arrays.fill(isPrime, true);
        for (int i = 2; i < Math.sqrt(n); i++) {
            if (isPrime[i]) {
                for (int j = 2 * i; j < n; j = j + i) {
                    isPrime[j] = false;
                }
            }
        }
        int count = 0;
        for (int i = 2; i < n; i++) {
            if (isPrime[i] == true) {
                count++;
            }
        }
        return count;
    }
```

Island Perimeter

```
public int islandPerimeter(int[][] grid) {
        int neighbor = 0;
        int island = 0;
        for (int i = 0; i < grid.length; i++) {
            for (int j = 0; j < grid[0].length; j++) {
                if (grid[i][j] == 1) {
                    island++;
                    if (i < grid.length - 1 && grid[i + 1][j] == 1) {
                        neighbor++;
                    }
                    if (j < grid[0].length - 1 && grid[i][j + 1] == 1) {
```

```
                    neighbor++;
                }
            }
        }
    }
    return island * 4 - neighbor * 2;
}
```

Repeated substring pattern

```
public boolean repeatedSubstringPattern(String str) {
    if (str.length() == 0) {
        return false;
    }
    int len = str.length();
    for (int i = len / 2; i >= 1; i--) {
        if (len % i == 0) {
            int n = len / i;
            StringBuilder sb = new StringBuilder();
            String sub = str.substring(0, i);
            for (int j = 0; j < n; j++) {
                sb.append(sub);
            }
            if (str.equals(sb.toString())) {
                return true;
            }
        }
    }
    return false;
}
```

Find Content Children

```
public int findContentChildren(int[] g, int[] s) {
    Arrays.sort(g);
    Arrays.sort(s);
    int i = 0;
    int j = 0;
    while (i < g.length && j < s.length) {
        if (g[i] <= s[j]) {
            i++;
            j++;
        } else {
            j++;
        }
    }
    return i;
}
```

Construct Rectangle when area is given

```
public int[] constructRectangle(int area) {
    int res[] = new int[2];
    int w = (int) Math.sqrt(area);
    while (area % w != 0) {
        w--;
    }
    int len = area / w;
    res[0] = len;
    res[1] = w;
    return res;
}
```

Min Stack

```
public void push(int x) {
   if (x <= min) {
       s1.push(min);
       min = x;
   }
```

```
        s1.push(x);
    }

    public void pop() {
        int temp = s1.pop();
        if (temp == min) {
            min = s1.pop();
        }

    }
```

Flatten a Linked List

```
private TreeNode prev = null;

public void flatten(TreeNode root) {
    if (root == null)
        return;
    flatten(root.right);
    flatten(root.left);
    root.right = prev;
    root.left = null;
    prev = root;
}
```

Find row with maximum number of 1s – O(m + n)

```
int rowWithMax1s(bool mat[R][C])
{
    int max_row_index = 0;

    int j = first(mat[0], 0, C-1);
    if (j == -1) // if 1 is not present in first row
      j = C - 1;

    for (int i = 1; i < R; i++)
    {
        // Move left until a 0 is found
        while (j >= 0 && mat[i][j] == 1)
        {
           j = j-1;  // Update the index of leftmost 1 seen so far
           max_row_index = i;  // Update max_row_index
        }
    }
    return max_row_index;
}
```

Next Permutation

```
public void nextPermutation(int[] nums) {
        if (nums.length <= 1) {
            return;
        }
        int len = nums.length;
        int i = len - 2;
        while (i >= 0 && nums[i] >= nums[i + 1]) {
            i--;
        }
        if (i >= 0) {
            int j = len - 1;
            while (j >= 0 && nums[i]  >= nums[j]) {
                j--;
            }
            swap(nums, i, j);
        }
        reverse(nums, i + 1, len - 1);
    }
```

Permutation with Duplicates

```
public List<List<Integer>> permuteUnique(int[] nums) {
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        if (nums.length == 0) {
            return new ArrayList<List<Integer>>();
        }
        List<Integer> list = new ArrayList<Integer>();
        HashMap<Integer, Integer> map = new HashMap<>();
```

```
        for (int i = 0; i < nums.length; i++) {
            int count = 1;
            if (map.containsKey(nums[i])) {
                count += map.get(nums[i]);
            }
            map.put(nums[i], count);
        }
        permuate(nums, result, list, nums.length, map);
        return result;
    }
    public void permuate(int[] nums, List<List<Integer>> result, List<Integer> curr, int length,
HashMap<Integer, Integer> map) {
        if (length == 0) {
            result.add(new ArrayList<>(curr));
            return;
        }
        for (Integer num : map.keySet()) {
            int count = map.get(num);
            if (count > 0) {
                map.put(num, count - 1);
                curr.add(num);
                permuate(nums, result, curr, length - 1, map);
                curr.remove(curr.size() - 1);
                map.put(num, count);
            }
        }
```

BattleShips in a Board

```
public int countBattleships(char[][] board) {
        int rowLen = board.length;
        int colLen = board[0].length;
        int count = 0;
        if (rowLen == 0) {
            return count;
        }
        for (int i = 0; i < rowLen; i++) {
            for (int j = 0; j < colLen; j++) {
                if (board[i][j] == '.') {
                    continue;
                }
                if (i < rowLen - 1 && board[i + 1][j] == 'X') {
                    continue;
                }
                if (j > 0 && board[i][j - 1] == 'X') {
                    continue;
                }
                count++;
            }
        }
        return count;
    }
```

Find Bottom Left Value of Tree

```
public int findBottomLeftValue(TreeNode root) {
        if (root == null) {
            return 0;
        }
        Queue<TreeNode> q = new LinkedList<>();
        int result = 0;
        q.add(root);
        while (!q.isEmpty()) {
            int size = q.size();
            for (int i = 0; i < size; i++) {
                TreeNode node = q.poll();
                if (i == 0) {
                    result = node.val;
                }
                if (node.left != null) {
                    q.add(node.left);
                }
                if (node.right != null) {
                    q.add(node.right);
                }
            }
```

```
        }
        return result;
    }
```

Largest Value in Each Tree Row

```
public List<Integer> largestValues(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) {
            return result;
        }
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);
        while (!q.isEmpty()) {
            int size = q.size();
            int max = Integer.MIN_VALUE;
            for (int i = 0; i < size; i++) {
                TreeNode node = q.poll();
                max = Math.max(max, node.val);
                if (i == size - 1) {
                    result.add(max);
                }
                if (node.left != null) {
                    q.add(node.left);
                }
                if (node.right != null) {
                    q.add(node.right);
                }
            }
        }
        return result;
    }
```

Longest Repeating Character Replacement

```
public int characterReplacement(String s, int k) {
        int length = s.length();
        if (length == 0) {
            return 0;
        }
        int start = 0;
        int end = 0;
        int maxCount = 0;
        int maxLen = Integer.MIN_VALUE;
        int arr[] = new int[256];
        for (int i = 0; i < length; i++) {
            char ch = s.charAt(i);
            arr[ch]++;
            if (maxCount < arr[ch]) {
                maxCount = arr[ch];
            }
            while (i - start - maxCount + 1 > k) {
                arr[s.charAt(start)]--;
                start++;
            }
            maxLen = Math.max(maxLen, i - start + 1);
        }
        return maxLen;
    }
```

Lexographical Numbers

```
public List<Integer> lexicalOrder(int n) {
        List<Integer> list = new ArrayList<Integer>();
        if (n == 0) {
            return list;
        }
        list.add(1);
        int curr = 1;
        for (int i = 1; i < n; i++) {
            if (curr * 10 <= n) {
                curr = curr * 10;
            } else {
                while ((curr % 10 == 9) || curr == n) {
                    curr = curr / 10;
```

```
                }
                curr++;

            }
            list.add(curr);
        }
        return list;
    }
```

Palindrome Partitioning

```
public List<List<String>> partition(String s) {
        List<List<String>> result = new ArrayList<List<String>>();
        if (s.length() == 0) {
          return result;
        }
        List<String> curr = new ArrayList<String>();
        dfs(s, 0, curr, result);
        return result;
    }
    public void dfs(String s, int index, List<String> curr, List<List<String>> result) {
        if (index == s.length()) {
            result.add(new ArrayList<>(curr));
            return;
        }
        for (int i = index; i < s.length(); i++) {
            if (isPalindrome(s, index, i)) {
                curr.add(s.substring(index, i + 1));
                dfs(s, i + 1, curr, result);
                curr.remove(curr.size() - 1);
            }
        }
    }
    public boolean isPalindrome(String s, int start, int end) {
        while (start < end) {
            if (s.charAt(start) != s.charAt(end)) {
                return false;
            }
            start++;
            end--;
        }
        return true;
    }
```

Increasing Subsequences – (O 2^n)

```
public List<List<Integer>> findSubsequences(int[] nums) {
         Set<List<Integer>> res= new HashSet<List<Integer>>();
         List<Integer> holder = new ArrayList<Integer>();
         findSequence(res, holder, 0, nums);
         List result = new ArrayList(res);
         return result;
     }

    public void findSequence(Set<List<Integer>> res, List<Integer> holder, int index, int[] nums) {
        if (holder.size() >= 2) {
            res.add(new ArrayList(holder));
        }
        for (int i = index; i < nums.length; i++) {
            if(holder.size() == 0 || holder.get(holder.size() - 1) <= nums[i]) {
                holder.add(nums[i]);
                findSequence(res, holder, i + 1, nums);
                holder.remove(holder.size() - 1);
            }
        }
    }
```

Print kth element in spiral order in a matrix – O(c) number of outer circular rings wrt to k

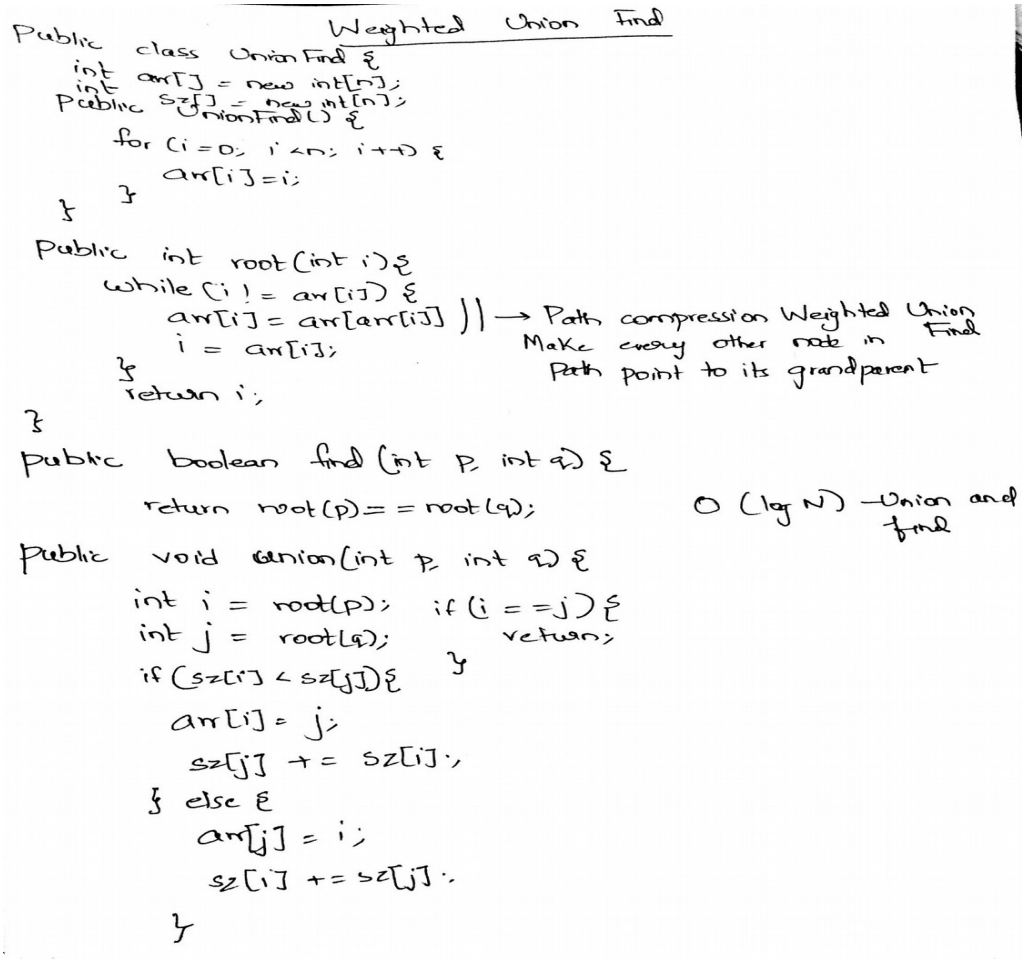
```
int findK(int A[MAX][MAX], int m, int n, int k)
{
    if (n < 1 || m < 1)
        return -1;

    /*.........If element is in outermost ring .......*/
```

```
    /* Element is in first row */
    if (k <= n)
        return A[0][k-1];

    /* Element is in last column */
    if (k <= (m+n-1))
        return A[(k-n)][n-1];

    /* Element is in last row */
    if (k <= (m+n-1+n-1))
        return A[m-1][n-1-(k-(m+n-1))];

    /* Element is in first column */
    if (k <= (m+n-1+n-1+m-2))
        return A[m-1-(k-(m+n-1+n-1))][0];

    /*.........If element is NOT in outermost ring .......*/
    /* Recursion for sub-matrix. &A[1][1] is
       address to next inside sub matrix.*/
    return findK((int (*)[MAX])(&(A[1][1])), m-2,
                 n-2, k-(2*n+2*m-4));
}
```

Palindrome Linked List

```
public boolean isPalindrome(ListNode head) {
        if (head == null) {
            return true;
        }
        if (head.next == null) {
            return true;
        }
        ListNode slow = head;
        ListNode fast = head;

while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        if (fast != null) {
            slow = slow.next;
        }
        slow = reverse(slow);
        fast = head;
        while (slow != null) {
            if (fast.val != slow.val) {
                return false;
            }
            slow = slow.next;
            fast = fast.next;
        }
        return true;
    }
    public ListNode reverse(ListNode curr) {
        ListNode prev = null;
        while (curr != null) {
            ListNode next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
```

Union Find



Graph Valid Tree – O(k logn)
Course Schedule

```
public static int root(int arr[], int i) {
        while (i != arr[i]) {
            arr[i] = arr[arr[i]];
            i = arr[i];
        }
        return i;
    }
    public boolean validTree(int n, int[][] edges) {
        // Write your code here

        if (edges.length == 0) {
            return true;
        }
        if (edges.length == 0 && n > 1) {
            return false;
        }
        int arr[] = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = i;
        }
        for (int i = 0; i < edges.length; i++) {
            int x = root(arr, edges[i][0]);
            int y = root(arr, edges[i][1]);
            if (x == y) { // if( not equal )count-- for number of connected components
                return false;
            }
            arr[x] = y;
```

```
        }
        return edges.length == n - 1;
    }
```

Trie Class

```
class TrieNode {
    public char val;
    public TrieNode[] children = new TrieNode[26];
    public boolean isWord;
    public TrieNode() {

    }
    TrieNode(char ch) {
        TrieNode node = new TrieNode();
        node.val = ch;
    }
}

public class Trie {
    private TrieNode root;
    /** Initialize your data structure here. */
    public Trie() {
       root = new TrieNode();
       root.val = ' ';
    }

    /** Inserts a word into the trie. */
    public void insert(String word) {
        TrieNode curr = root;
        for (int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            if (curr.children[ch - 'a'] == null) {
                TrieNode newNode = new TrieNode(ch);
                curr.children[ch - 'a'] = newNode;
            }
            curr = curr.children[ch - 'a'];
        }
        curr.isWord = true;
    }

    /** Returns if the word is in the trie. */
    public boolean search(String word) {
        TrieNode curr = root;
        for (int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);
            if (curr.children[ch - 'a'] == null) {
                return false;
            }
            curr = curr.children[ch - 'a'];
        }
        return curr.isWord;
    }

    /** Returns if there is any word in the trie that starts with the given prefix. */
    public boolean startsWith(String prefix) {
        TrieNode curr = root;
        for (int i = 0; i < prefix.length(); i++) {
            char ch = prefix.charAt(i);
            if (curr.children[ch - 'a'] == null) {
                return false;
            }
            curr = curr.children[ch - 'a'];
        }
        return true;
    }
}
```

Surrounded Regions

```
public class Solution {
    public void solve(char[][] board) {
```

```
        if (board.length == 0 || board[0].length == 0) {
            return;
        }
        if (board.length < 2 || board[0].length < 2)     return;
        int m = board.length;
        int n = board[0].length;
        for (int i = 0; i < m; i++) {
            if (board[i][0] == 'O') {
                dfs(board, i , 0);
            }
            if (board[i][n - 1] == 'O') {
                dfs(board, i, n - 1);
            }
        }
        for (int i = 0; i < n; i++) {
            if (board[0][i] == 'O') {
                dfs(board, 0, i);
            }
            if (board[m - 1][i] == 'O') {
                dfs(board, m - 1, i);
            }
        }
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (board[i][j] == 'O') {
                    board[i][j] = 'X';
                } else if (board[i][j] == '+') {
                    board[i][j] = 'O';
                }
            }
        }

    }
    public void dfs(char[][] board, int i, int j) {
        if (i < 0 || i >= board.length || j < 0 || j >= board[0].length) {
            return;
        }
        if (board[i][j] == 'O') {
            board[i][j] = '+';
        }
            if (i > 1 && board[i - 1][j] == 'O') {
                dfs(board, i - 1, j);
            }
            if (i < board.length - 2 && board[i + 1][j] == 'O') {
                dfs(board, i + 1, j);
            }
            if (j > 1 && board[i][j - 1] == 'O') {
                dfs(board, i, j - 1);
            }
            if (j < board[i].length - 2 && board[i][j + 1] == 'O') {
                dfs(board, i, j + 1);
            }
    }
}
```

Word Search II

```
public class Solution {
    public static TrieNode buildTree(String[] words) {
       TrieNode root = new TrieNode();
       for (String w : words) {
         TrieNode p = root;
         for (char c : w.toCharArray()) {
            int i = c - 'a';
            if (p.children[i] == null) p.children[i] = new TrieNode();
            p = p.children[i];
         }
         p.word = w;
       }
       return root;
    }

    public List<String> findWords(char[][] board, String[] words) {
        List<String> result = new ArrayList<String>();
```

```
        TrieNode root = buildTree(words);
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                dfs(board, i, j, root, result);
            }
        }
        return result;
    }
    public void dfs(char[][] board, int i, int j, TrieNode curr, List<String> result) {
        char c = board[i][j];

       if  (c == '#' || curr.children[c -'a'] == null) {
            return;
        }
        curr = curr.children[c - 'a'];
        if (curr.word != null) {
            result.add(curr.word);
            curr.word = null;
        }
        board[i][j] = '#';
        if (i > 0) {
            dfs(board, i - 1, j, curr, result);
        }
        if (i < board.length - 1) {
            dfs(board, i + 1, j, curr, result);
        }
        if (j > 0) {
            dfs(board, i, j - 1, curr, result);
        }
        if (j < board[0].length - 1) {
            dfs(board, i, j + 1, curr, result);
        }
        board[i][j] = c;
    }
}
```

Valid Word Square

```
public class Solution {
    public boolean validWordSquare(List<String> words) {
        if(words == null || words.size() == 0){
            return true;
        }
        int n = words.size();
        for(int i=0; i<n; i++){
            for(int j=0; j<words.get(i).length(); j++){
                if(j >= n || words.get(j).length() <= i || words.get(j).charAt(i) !=
words.get(i).charAt(j))
                    return false;
            }
        }
        return true;
    }
}
```

AutoComplete

```
do {
    tn = temp.children[cArray[index] - 'A'];
    // if you reached the end of the string, then no words with this prefix
    if (tn == null) {
      return null;
    }

    index++;
    temp = tn;
  } while (index < cArray.length);

  // temp is at the node representing the last char of the prefix
  // do a traversal for all paths below this.
  List<String> words = new ArrayList<String>();
  Deque<TrieNode> DQ = new ArrayDeque<TrieNode>();
  DQ.addLast(temp);
  while (!DQ.isEmpty()) {
    TrieNode first = DQ.removeFirst();
           if(first.terminal){
             words.add(first.word);
```

```
    }

    for(TrieNode n : first.children){
      if(n != null){
        DQ.add(n);
      }
    }
  }

  return words.toArray(new String[0]);
```

Swap Nodes in Pairs
Swap Linked List Nodes in Pairs

```
public ListNode swapPairs(ListNode head) {
        if (head == null) {
            return null;
        }
        if (head.next == null) {
            return head;
        }
        ListNode prev = head;
        ListNode curr = prev.next;
        head = curr;


        while (true) {
            ListNode next = curr.next;
            curr.next = prev;
            if (next == null || next.next == null) {
                prev.next = next;
                break;
            }
            prev.next = next.next;
            prev = next;
            curr = prev.next;
        }
        return head;
    }
```

HashMap put
HashMap get

```
public K put(K key, K value) {
  int hash = hash(key);
  int index = hash & (table.length - 1);
  if (key already present ) {
    return oldValue;
  }
  addToHashTable(hash, key, value, index);
  return null;
}

public K get(K key) {
  int hash = hash(key);
  int index = hash & (table.length - 1);
  if (at that index ) {
    if (key == key, hash == hash) {
      return value of key;
    }
    //index of null key is 0
    return null;
}
```

Reservior Sampling
Randomly choose k samples from n elements

```
public void selectKItems(int stream[], int n, int k) {
    Random rand = new Random();
    int reservoir[k];
    for (int i = 0; i < k; i++) {
        reservoir[i] = stream[i];
    }
    for (int i = k; i < n; i++) {
        // Pick a random index from 0 to i.
        int j = rand.nextInt(i + 1);
```

```
        if (j < k) {
          reservoir[j] = stream[i];
        }
    }
 }
```
The probability of ith element going to second position =
(probability that ith element is not picked in previous iteration) x (probability that ith element is picked in this iteration)
So the probability = ((n-1)/n) x (1/(n-1)) = 1/n

Find Longest subarray with equal number of letters and numbers

```
public int longestSubstring(char[] arr) {
  if(arr.length == 0) {
    return 0;
  }
  if (arr.length == 1) {
    return 1;
  }
  int digits[] = getDifference(arr);
  int max[] = getLongestSubstring(digits);

  return max[1] - max[0];
}
public int[] getDifference(char[] arr) {
  int digits[] = new int[arr.length];
  int digit = 0;
  for (int i = 0; i < arr.length; i++) {
    char ch = arr.charAt(i);
    if (Character.isDigit(ch)) {
      digit++;
    } else if (Character.isLetter(ch)) {
      digit--;
    }
    digits[i] = digit;
  }
  return digits;
}
public int[] getLongestSubstring(int[] digits) {
  HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
  int longest = Integer.MIN_VALUE;
  for (int i = 0; i < digits.length; i++) {
    if (!map.containsKey(digits[i])) {
      map.put(digits[i], i);
    } else {
      int len = i - map.get(digits[i]);
      if (len > longest) {
        longest = len;
        max[0] = map.get(digits[i]);
        max[1] = i;
      }
    }
  }
  return max;
}
```

Time Planner

Implement a meeting planner that can schedule meetings between two persons at a time.

```
public int[] meetingPoint(int[][] timesA, int[][] timesB, int dur) {
  if (timesA == 0 || timesB == 0) {
     return null;
  }
  int i = 0;
  int j = 0;
  int result[] = new int[2];
  while (i < timesA.length && j < timesB.length) {
     int startA = timesA[i][0];
     int endA = timesA[i][1];
     int startB = timesB[j][0];
     int endB = timesB[j][1];
     int maxStart = Math.max(startA, startB);
     int minEnd = Math.min(startB, endB);
```

```
     int diff = minEnd - maxStart;
     if (diff >= dur) {
        int[0] = maxStart;
        int[1] = minEnd;
        break;
     }
     if (endA < endB) {
        i++;
     } else {
        j++;
     }
  }
  return result;
}
```

Busiest Time in the Mall

Walls and Gates

```
#Walls and Gates
public void walls(int board[][]) {
  if (board.length == 0 || board[0].length) {
    return;
  }
  boolean visited[][] = new boolean[board.length][board[0].length];
  for (int i = 0; i < board.length; i++) {
    for (int j = 0; j < board[0].length; j++) {
      if (board[i][j] == 0) {
        distance(board, i, j, 0, visited);
      }
    }
  }
}
public void distance(int board[][], int i, int j, int distance, boolean visited[][]) {
  if (i < 0 || i >= board.length || j < 0 || j >= board[0].length ) {
    return;
  }
  if (visited[i][j] || distance > board[i][j]) {
    return;
  }

  visited[i][j] = true;
  if (distance < board[i][j]) {
    board[i][j] = distance;
  }
  distance(board, i + 1, j, distance + 1, visited);
  distance(board, i - 1, j, distance + 1, visited);
  distance(board, i, j - 1, distance + 1, visited);
  distance(board, i, j + 1, distance + 1, visited);
  visited[i][j] =false;
}
```

Add all greater values to every node in a given BST

```
void modifyBSTUtil(struct node *root, int *sum)
{
    // Base Case
    if (root == NULL)  return;

    // Recur for right subtree
    modifyBSTUtil(root->right, sum);

    // Now *sum has sum of nodes in right subtree, add
    // root->data to sum and update root->data
    *sum = *sum + root->data;
    root->data = *sum;

    // Recur for left subtree
    modifyBSTUtil(root->left, sum);
}

// A wrapper over modifyBSTUtil()
```

```
void modifyBST(struct node *root)
{
    int sum = 0;
    modifyBSTUtil(root, &sum);
}
```

String to Integer atoi

```
public int myAtoi(String str) {
        if (str.isEmpty()) {
            return 0;
        }
        int sign = 1;
        int base = 0;
        int i = 0;
        while (str.charAt(i) == ' ') {
            i++;
        }
        if (str.charAt(i) == '+' || str.charAt(i) == '-') {
            sign = str.charAt(i) == '-' ? -1 : 1;
            i++;
        }
        while (i < str.length() && str.charAt(i) >= '0' && str.charAt(i) <= '9') {
            if (base > Integer.MAX_VALUE / 10 || (base == Integer.MAX_VALUE / 10 && str.charAt(i) >
'7')) {

                return (sign == 1) ? Integer.MAX_VALUE : Integer.MIN_VALUE;

            }
            base = base * 10 + (str.charAt(i) - '0');
            i++;
        }
        return sign * base;
    }
```

POW

```
public double myPow(double x, int n) {
         if (n == Integer.MIN_VALUE && x == 2.0) {
            return 0;
        }
        if(n == 0)
            return 1;
        if(n<0){
            n = -n;
            x = 1/x;
        }
        return (n%2 == 0) ? myPow(x*x, n/2) : x*myPow(x*x, n/2);
```

Serialize and Deserialize Binary Tree and BST

```
public String serialize(TreeNode root) {
        if (root == null) {
            return "$#";
        }
        String result = root.val + "#";
        result += serialize(root.left);
        result += serialize(root.right);
        return result;
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        String arr[] = data.split("#");
        Queue<String> queue = new LinkedList<String>();
        for (String str : arr) {
            queue.add(str);
        }
        return deserialize(queue);
    }
    public TreeNode deserialize(Queue<String> queue) {
        String val = queue.remove();
        if (val.equals("$")) {
```

```
            return null;
        }
        TreeNode root = new TreeNode(Integer.valueOf(val));
        root.left = deserialize(queue);
        root.right = deserialize(queue);
        return root;
    }
```

All sets of consecutive Elements equal to a target

```
             int low = 0;
            int high = 0;
            int sum = a[0];
            while(high < a.length) {
                if(sum < 13) {
                        high++;
                        if(high < a.length) {
                                sum+= a[high];
                        }
                } else if(sum > 13) {
                        sum-=a[low];
                        low++;
                }
                if(sum == 13) {
                        for(int i=low;i<=high;i++) {
                                System.out.println(a[i]);
                        }
          //            System.out.println("new ");
                        low++;
                        high = low;
                        sum = a[low];
// sum -= a[low];, low++;


                }
```

Longest Subarray with sum less than or equal to k (sum <= k)

```
public int maxLength(a, k) {
    int sum = 0;
    int len = Integer.MIN_VALUE;
    int start = 0;
    int end = 0;

    while (end < a.length) {
        sum += a[end];
        end++;
        while (sum > k) {
            sum -= a[start];
            start++;
        }
        len = Math.max(len, end - start);
    }

    return len;
}
```

final marks the reference, not the object. You can't make that reference point to a different hash table. But you can do anything to that object, including adding and removing things.

Remove minimum number of characters so that two strings become anagram

```
int remAnagram(string str1, string str2) {
    // make hash array for both string and calculate
    // frequency of each character
    int count1[CHARS] = {0}, count2[CHARS] = {0};


    // count frequency of each charcter in first string
    for (int i=0; str1[i]!='\0'; i++)
        count1[str1[i]-'a']++;
```

```
    // count frequency of each charcter in second string
    for (int i=0; str2[i]!='\0'; i++)
        count2[str2[i]-'a']++;


    // traverse count arrays to find number of charcters
    // to be removed
    int result = 0;
    for (int i=0; i<26; i++)
        result += abs(count1[i] - count2[i]);
    return result;
}
```

Students Attendance

```
public boolean checkRecord(String s) {
        int count = 0;
        int continous = 0;
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (ch == 'A') {
                count++;
                continous = 0;
            } else if (ch == 'L') {
                continous++;
            } else {
                continous = 0;
            }
            if (count > 1 || continous > 2) {
              return false;
          }
        }
        return true;
    }
```

Longest Consecutive Substring with Maximum sum

```
public int maxSubstring(int arr[]) {
  if (arr.length == 0) {
    return 0;
  }
  if (arr.length == 1) {
    return 1;
  }
  int maxSum = Integer.MIN_VALUE;
  int start = 0;
  int end = 0;
  int startIndex = 0;
  int sum = 0;
  for (int i = 0; i < arr.length; i++) {
    sum += arr[i];
    if (sum > maxSum) {
      maxSum = sum;
      start = startIndex;
      end = i;
    }
    if (sum < 0) {
      startIndex = i + 1;
    }
  }
  return end - start - 1;
}
```

Number of Subarray sums equal k

```
public int subarraySum(int[] arr, int k) {
        if (arr.length  == 0) {
            return 0;
        }
        int count = 0;
        int sum = 0;
        HashMap<Integer, Integer> map = new HashMap<>();
        map.put(0, 1);
        for (int i = 0; i < arr.length; i++) {
            sum += arr[i];
```

```
            if (map.containsKey(sum - k)) {
                count += map.get(sum - k);
            }
            if (map.containsKey(sum)) {
                map.put(sum, map.get(sum) + 1);
            } else {
                map.put(sum, 1);
            }
        }
        return count;
    }
```

#Maximum Path Sum in a binary tree

```
int maxCount = Integer.MIN_VALUE;
public int maximumPathSum(TreeNode node) {
  if (node == null) {
    return 0;
  }
  helper(node, 0);
  return maxCount;
}
public void helper(TreeNode node, int currSum) {
  if (node == null) {
    return;
  }
  currSum += node.val;
  if (node.left == null && node.right == null) {
    maxCount = Math.max(maxCount, currSum);
  } else {
    helper(node.left, currSum);
    helper(node.right, currSum);
  }
// If you don't pass currSum as parameter, then currSum -= node.val;
```

Find the Pivot- Sorted Rotated Array

```
static int shiftedArrSearch(int[] shiftArr, int num) {
                if (shiftArr == null || shiftArr.length == 0) {
                        return -1;
                }
                int pivot = findPivot(shiftArr);

                if (shiftArr[0] <= num) {
                        return binarySearch(shiftArr, 0, pivot - 1, num);
                } else {
                        return binarySearch(shiftArr, pivot, shiftArr.length - 1, num);
                }
        }

        public static int findPivot(int[] arr) {
                int left = 0;
                int right = arr.length - 1;
                while (left < right) {
                        int mid = (right + left) / 2;
                        if (mid == 0 || arr[mid - 1] > arr[mid]) {
                                return mid;
                        }
                        if (arr[mid] >= arr[left]) {
                                left = mid + 1;
                        } else {
                                right = mid;
                        }
                }
                return 0;
        }
        public static int binarySearch(int[] arr, int left, int right, int num) {
                while (left < right) {
                        int mid = (left + right) / 2;
                        if (arr[mid] == num) {
                                return mid;
                        } else if (arr[mid] < num) {
                                left = mid + 1;
                        } else {
```

```
                                right = mid;
                        }
                }
                return -1;
        }
```

Consecutive Elements equal to target with negative numbers

```
public void consecutiveSum(int arr[], int target) {
        int sum = 0;
        int maxLen = Integer.MIN_VALUE;
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < arr.length; i++) {
                sum += arr[i];
                if (sum == target) {
                        maxLen = i + 1;
                        System.out.println("0 -" + i);
                }
                // 1, 4, 20, 3, 10
                // target = 33
                if (map.containsKey(Math.abs(target - sum))) {
                        maxLen = Math.max(maxLen, map.get(Math.abs(target - sum)));
                } else {
                        map.put(sum, i);
                }
        }
}
```

Longest subarray with equal number of 0s and 1s **

```
for(int i=0;i<len;i++){
                    int curr = list.get(i);
                    if(curr==0){
                        sum += -1;
                    }else{
                        sum += 1;
                    }
                    if(sum == 0){
                        max_len = i+1;
                    }
                    if(hm.containsKey(sum)){
                        int currLen = (i-hm.get(sum));
                        if(max_len<currLen)
                            max_len = currLen;
                    }else{
                        hm.put(sum,i);
                    }
```

Text Wrap
Word Wrap
Word Justification

```
private static String textJustify(String[] words, int numOfWords) {

        int spaces[][] = new int[words.length][words.length];
        int cost[][] = new int[words.length][words.length];

        /**
         * Find the number of remaining spaces after placing words from i to j in a line
         */
        for (int i = 0; i < words.length; i++) {
                spaces[i][i] = numOfWords - words[i].length();
                for (int j = i + 1; j < words.length; j++) {
                        spaces[i][j] = spaces[i][j - 1] - words[j].length() - 1;
                }
        }

        /**
         * Find the cost it takes to put words from i to j on a line
         */
        for (int i = 0; i < words.length; i++) {
                for (int j = i; j < words.length; j++) {
                        if (spaces[i][j] < 0) {
                                cost[i][j] = Integer.MAX_VALUE;
```

```
                        } else {
                                cost[i][j] = spaces[i][j] * spaces[i][j];
                        }
                }
        }
        /**
         * Check if i to j words stay on same line.
         * If they don't stay, find the word which breaks the lines
         */
        int minCost[] = new int[words.length];
        int result[] = new int[words.length];
        for(int i = words.length - 1; i >= 0 ; i--) {
                minCost[i] = cost[i][words.length-1];
                result[i] = words.length;
                for(int j = words.length - 1; j > i; j--){
                        if(cost[i][j-1] == Integer.MAX_VALUE){
                                continue;
                        }
                        if(minCost[i] > minCost[j] + cost[i][j-1]){
                                minCost[i] = minCost[j] + cost[i][j-1];
                                result[i] = j;
                        }
                }
        }
        int i = 0;
        int j;

        System.out.println("Minimum cost is " + minCost[0]);
        System.out.println("\n");
        //finally put all words with new line added in
        //string buffer and print it.
        StringBuilder builder = new StringBuilder();
        do {
                j = result[i];
                for(int k = i; k < j; k++){
                        builder.append(words[k] + " ");
                }
                builder.append("\n");
                i = j;
        }       while(j < words.length);

        return builder.toString();
    }
```

isSubTree
sub tree

```
public boolean isSubtree(TreeNode s, TreeNode t) {
        if (s == null) {
            return false;
        }
        if (isSameTree(s, t)) {
            return true;
        }
        return isSubtree(s.left, t) || isSubtree(s.right, t);
    }
    public boolean isSameTree(TreeNode s, TreeNode t) {
        if (s == null && t == null) {
            return true;
        }
        if (s != null && t != null && s.val == t.val && isSameTree(s.left, t.left) &&
isSameTree(s.right, t.right)) {
            return true;
        }
        return false;
    }
```

Valid Sudoku

```
public boolean isValidSudoku(char[][] board) {
        for (int i = 0; i < 9; i++) {
```

```
            HashSet<Character> rows = new HashSet<>();
            HashSet<Character> cols = new HashSet<>();
            HashSet<Character> grid = new HashSet<>();
            for (int j = 0; j < 9; j++) {
                if (board[i][j] != '.' && !rows.add(board[i][j])) {
                    return false;
                }
                if (board[j][i] != '.' && !cols.add(board[j][i])) {
                    return false;
                }
                int r = (3 * (i / 3));
                int c = (3 * (i % 3)) ;
                if (board[r + j / 3][c + j % 3] != '.' && !grid.add(board[r + j / 3][c + j % 3])) {
                    return false;
                }
            }
        }
        return true;
    }
```

Longest Increasing Subsequence – O(nlogn)

```
public int lengthOfLIS(int[] nums) {
        int[] dp = new int[nums.length];
        int len = 0;

        for(int x : nums) {
            int i = Arrays.binarySearch(dp, 0, len, x);
            if(i < 0) i = -(i + 1);
            dp[i] = x;
            if(i == len) len++;
        }

        return len;
    }
```

Permutation of Array of Arrays

```
public static List<List<Integer>> permutate(int[][] arr){
                List<List<Integer>> result = new ArrayList<List<Integer>>();
                permutateHelper(arr, 0, result, new ArrayList());
                return result;
            }
public static void permutateHelper(int[][] arr, int arrIndex, List<List<Integer>> result,
List<Integer> partial){
                if(partial.size() == arr.length){
                        result.add(new ArrayList(partial));
                        return;
                }

                int[] currArr = arr[arrIndex];
                for(int i = 0; i < currArr.length; i++ ){
                        partial.add(currArr[i]);
                        permutateHelper(arr, arrIndex + 1, result, partial);
                        partial.remove(partial.size() - 1);
                }
            }
```

Longest Word in the dictionary through deleting – TC (O (n * avg len of word) , SC – O(len of tmep)

```
public boolean isSubsequence(String s1, String s2) {
                int i = 0;
                int j = 0;
                while (i < s1.length() && j < s2.length()) {
                        if (s1.charAt(i) == s2.charAt(j)) {
                                j++;
                        }
                        i++;
                }
                return j == s2.length();
            }

            public String longestWord(String s, List<String> dict) {
                String temp = "";
                if (s == null || s.length() == 0) {
```

```
                        return temp;
                }
                if (dict.size() == 0) {
                        return temp;
                }
                for (String str : dict) {
                        if (isSubsequence(s, str) || (str.length() == temp.length() &&
str.compareTo(temp) < 0)) {
                                temp = str;
                        }
                }
                return temp;
           }
```

Minimum Depth of Binary Tree

```
public int minDepth(TreeNode root) {
        if (root == null) {
          return 0;
        }
        if (root.left == null) {
            return minDepth(root.right) + 1;
        }
        if (root.right == null) {
            return minDepth(root.left) + 1;
        }
        return Math.min(minDepth(root.left), (minDepth(root.right))) + 1;
    }
```

Weighted Random Number

```
public static int weightedSum(int[] weights, int[] nums) {
                int weightSum[] = new int[weights.length];
                weightSum[0] = weights[0];
                for (int i = 1; i < weights.length; i++) {
                        weightSum[i] = weightSum[i - 1] + weights[i];
                }
                Random rand = new Random();
                int randWeight = rand.nextInt(weightSum[weights.length - 1] + 1);
                int index = Arrays.binarySearch(weightSum, randWeight);
                if (index < 0) {
                        index = - (index + 1);
                }
                return nums[index];
           }
```

Quick Select

Kth largest/smallest element in an unsorted array – O(n) and worst case O(n^2)

```
public class Solution {
    public int findKthLargest(int[] nums, int k) {
        int start = 0;
        int end = nums.length - 1;
        k = nums.length - k;
        while (start < end) {
          int pivot = partion(nums, start, end);
          if (pivot == k) {
            return nums[k];
          }
          if (k < pivot) {
            end = pivot - 1;
          } else {
            start = pivot + 1;
          }
        }
        return nums[start];
    }
    private int partion(int[] nums, int start, int end) {
        int pivot = start;

        while (start <= end) {
            while (start <= end && nums[start] <= nums[pivot]) {
                start++;
            }
            while (start <= end && nums[end] > nums[pivot]) {
                end--;
```

```
            }
            if (start <= end) {;
              int temp = nums[start];
              nums[start] = nums[end];
              nums[end] = temp;
              start++;
              end--;
            }
        }
        int temp = nums[end];
        nums[end] = nums[pivot];
        nums[pivot] = temp;
        return end;
    }
}
```

Game of Life

```
    public void gameOfLife(int[][] board) {
        int row = board.length;
        int col = board[0].length;
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
             int lives = getNeighbors(board, i, j);
             if (board[i][j] == 1) {
                if (lives >= 2 && lives <= 3) {
                  board[i][j] = 3; // Make the 2nd bit 1: 01 ---> 11
                }
             } else {
                if (lives == 3) {
                    board[i][j] = 2;
                }
            }


          }
        }
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                board[i][j] >>= 1;
            }
        }
    }
    public int getNeighbors(int board[][], int i, int j) {
           int lives = 0;
                for(int x = i - 1; x <= i + 1; ++x) {
                  for(int y = j - 1; y <= j + 1; ++y) {
                    if(x >= 0 && x < board.length && y >= 0 && y < board[0].length) {
                        lives += board[x][y] & 1;
                    }
                }
            }
            lives -= board[i][j] & 1;
            return lives;
    }
}
```

#Concatenated Words

```
/**
  1. Create trie for all words in the list;
  2. Iterate over the words and check if they are a combination of words
*/
public boolean isConcatenatedWord(String word, int count, TrieNode root) {
  if (words == null || words.length() == 0) {
    return false;
  }
  TrieNode curr = root;
  for (int i = 0; i < word.length(); i++) {
    char ch = word.charAt(i);
    if (curr.children[ch - 'a'] == null) {
      return false;
    }
    if (curr.children[ch - 'a'].isWord) {
      if (i == word.length() - 1) {
        return count >= 1;
```

```
      }
      if (isConcatenatedWord(word.substring(i + 1), count + 1, root) {
        return true;
      }
    }
    curr = curr.children[ch - 'a'];
  }
  return false;
}
```

Combination Sum 4

```
public int combinationSum4(int[] nums, int target) {
    int[] comb = new int[target + 1];
    comb[0] = 1;
    for (int i = 1; i < comb.length; i++) {
        for (int j = 0; j < nums.length; j++) {
            if (i - nums[j] >= 0) {
                comb[i] += comb[i - nums[j]];
            }
        }
    }
    return comb[target];
}
```

Partition Linked List Stable

```
    public ListNode partition(ListNode head, int x) {
        if (head == null) {
            return null;
        }
        ListNode curr = head;
        ListNode dummy1 = new ListNode(0);
        ListNode dummy2 = new ListNode(0);
        ListNode curr1 = dummy1;
        ListNode curr2 = dummy2;
        while (curr != null) {
            ListNode temp = curr.next;
            if (curr.val < x) {
                curr1.next = curr;
                curr1 = curr;
            } else {
                curr2.next = curr;
                curr2 = curr;
            }
            curr = temp;
        }
        curr2.next = null;
        curr1.next = dummy2.next;
        return dummy1.next;
    }
```

Not in stable

```
start = head;
end = head;
        while (curr != null) {
            ListNode temp = curr.next;
            if (curr.val < x) {
                start.next = head;
                head = start;
            } else {
                end.next = curr;
                tail = curr;
            }
            curr = temp;
        }
        tail.next = null;
        return head;
    }
```

Num of Islands II

Find – O(1), Union – O(logn) – TC (Nlogn)

```
public class Solution {
    public List<Integer> numIslands2(int m, int n, int[][] positions) {
        List<Integer> result = new ArrayList<>();
        if (m == 0 || n == 0) {
```

```
            return null;
        }
        int dirs[][] = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
        int roots[] = new int[m * n];
        Arrays.fill(roots, -1);
        int count = 0;
        for (int pos[] : positions) {
            int root = n * pos[0] + pos[1]; // find the root
            roots[root] = root;
            count++;

            for (int dir[] : dirs) {
                int x = pos[0] + dir[0];
                int y = pos[1] + dir[1];
                int neighbor = n * x + y;
                if (x < 0 || x >= m || y < 0 || y >= n || roots[neighbor] == -1) {
                    continue;
                }
                int nbRoot = findNeighbors(roots, neighbor);
                if (root != nbRoot) {
                    count--;
                    roots[root] = nbRoot;
                    root = nbRoot;
                }

            }
            result.add(count);
        }
        return result;
    }
    public int findNeighbors(int[] roots, int i) {
        while (i != roots[i]) {
            roots[i] = roots[roots[i]];
            i = roots[i];
        }
        return i;
    }
}
```

Alien Dictionary

```
public class Solution {
    int N = 26;
    public String alienOrder(String[] words) {
        int visited[] = new int[N];
        Arrays.fill(visited, -1);
        boolean adj[][] = new boolean[N][N];
        buildGraph(words, adj, visited);

        StringBuilder sb = new StringBuilder();
        for(int i = 0; i < N; i++) {
            if (!dfs(i, adj, visited, sb)) {
                return "";
            }
        }
        return sb.reverse().toString();
    }
    public boolean dfs(int i, boolean[][] adj, int[] visited, StringBuilder sb) {
        visited[i] = 1;
        for (int j = 0; j < N; j++) {
            if (adj[i][j]) {
                if (visited[j] == 1) {
                    return false;
                }
                if (visited[j] == 0) {
                    if (!dfs(j, adj, visited, sb)) {
                        return false;
                    }
                }
            }
        }
        visited[i] = 2;
        sb.append((char) (i + 'a'));
```

```
        return true;
    }

    public void buildGraph(String[] words, boolean[][] adj, int[] visited) {
        for (int i = 0; i < words.length; i++) {
            for (char ch : words[i].toCharArray()) {
                visited[ch - 'a'] = 0;
            }
            if (i > 0) {
                String w1 = words[i - 1];
                String w2 = words[i];
                for (int j = 0; j < Math.min(w1.length(), w2.length()); j++) {
                    char ch1 = w1.charAt(j);
                    char ch2 = w2.charAt(j);
                    if (ch1 != ch2) {
                        adj[ch1 - 'a'][ch2 - 'a'] = true;
                        break;
                    }
                }
            }
        }
    }
}
```

Palindrome Permutation II

```
private List<String> list = new ArrayList<>();

public List<String> generatePalindromes(String s) {
    int numOdds = 0; // How many characters that have odd number of count
    int[] map = new int[256]; // Map from character to its frequency
    for (char c: s.toCharArray()) {
        map[c]++;
        numOdds = (map[c] & 1) == 1 ? numOdds+1 : numOdds-1;
    }
    if (numOdds > 1)   return list;


    String mid = "";
    int length = 0;
    for (int i = 0; i < 256; i++) {
        if (map[i] > 0) {
            if ((map[i] & 1) == 1) { // Char with odd count will be in the middle
                mid = "" + (char)i;
                map[i]--;
            }
            map[i] /= 2; // Cut in half since we only generate half string
            length += map[i]; // The length of half string
        }
    }
    generatePalindromesHelper(map, length, "", mid);
    return list;
}
private void generatePalindromesHelper(int[] map, int length, String s, String mid) {
    if (s.length() == length) {
        StringBuilder reverse = new StringBuilder(s).reverse(); // Second half
        list.add(s + mid + reverse);
        return;
    }
    for (int i = 0; i < 256; i++) { // backtracking just like permutation
        if (map[i] > 0) {
            map[i]--;
            generatePalindromesHelper(map, length, s + (char)i, mid);
            map[i]++;
        }
    }
}
```

Group Shifted Strings

```
public List<List<String>> groupStrings(String[] strings) {
        List<List<String>> result = new ArrayList<>();
        HashMap<String, List<String>> map = new HashMap<>();
        for (String s :  strings) {
            String key = getCode(s);
             if(!map.containsKey(key)) {
```

```
                map.put(key, new ArrayList<String>());
             }
            map.get(key).add(s);
    }
    for(String key: map.keySet()){
        List<String> list = map.get(key);
        Collections.sort(list);
        result.add(list);
    }
    return result;


 }
    public String getCode(String s) {
        int arr[] = new int[s.length()];
        arr[0] = 0;
        char ch = s.charAt(0);
        for (int i = 1; i < s.length(); i++) {
            char curr = s.charAt(i);
            arr[i] = curr - ch < 0 ? ((curr - ch) % 26) + 26 : curr - ch;
        }
        return Arrays.toString(arr);
    }
}
```

Factor Combinations

```
public List<List<Integer>> getFactors(int n) {
        List<List<Integer>> results = new ArrayList<>();
        List<Integer> curr = new ArrayList<>();
        perm(n, results, curr, 2);
        return results;
    }
    public void perm(int n, List<List<Integer>> results, List<Integer> curr, int start) {
        if (n == 1) {
            if (curr.size() > 1) {
              results.add(new ArrayList<>(curr));
            }
            return;
        }
        for (int i = start; i <= n; i++) {
            if (n % i == 0) {
                curr.add(i);
                perm(n / i, results, curr, i);
                curr.remove(curr.size() - 1); }}}
```

Happy Number

```
public int sqSum(int n) {
        int sum = 0;
        while (n != 0) {
            int temp = n % 10;
            sum += temp * temp;
            n = n / 10;
        }
        return sum;
    }
    public boolean isHappy(int n) {
        if (n == 1) {
            return true;
        }
        int slow = n;
        int fast = sqSum(n);
        while (slow != fast) {
            slow = sqSum(slow);
            fast = sqSum(sqSum(fast));
        }
        return slow == 1;
    }
```

Decode Ways

```
public int numDecodings(String s) {
        if (s== null || s.length() == 0) {
            return 0;
        }
        int dp[] = new int[s.length() + 1];
```

```
        dp[0] = 1;
        dp[1] = s.charAt(0) == '0' ? 0 : 1;
        for (int i = 2; i <= s.length(); i++) {
            int s1 = Integer.parseInt(s.substring(i - 1, i));
            int s2 = Integer.parseInt(s.substring(i - 2, i));
            if (s1 >= 1 && s1 <= 9) {
                dp[i] += dp[i - 1];
            }
            if (s2 >= 10 && s2 <= 26) {
                dp[i] += dp[i - 2];
            }   return dp[s.length()]; }
```

Word Pattern II

```
        public boolean wordPatternMatch(String pattern, String str) {
            Map<Character, String> map = new HashMap<>();
            Set<String> set = new HashSet<>();

            return isMatch(str, 0, pattern, 0, map, set);
        }

        boolean isMatch(String str, int i, String pat, int j, Map<Character, String> map,
Set<String> set) {
            // base case
            if (i == str.length() && j == pat.length()) {
                    return true;
            }
            if (i == str.length() || j == pat.length()) {
                    return false;
            }

            // get current pattern character
            char c = pat.charAt(j);

            // if the pattern character exists
            if (map.containsKey(c)) {
                    String s = map.get(c);

                    // then check if we can use it to match str[i...i+s.length()]
                    if (!str.startsWith(s, i)) {
                            return false;
                    }

                    // if it can match, great, continue to match the rest
                    return isMatch(str, i + s.length(), pat, j + 1, map, set);
            }

            // pattern character does not exist in the map
            for (int k = i; k < str.length(); k++) {
                    String p = str.substring(i, k + 1);

                    if (set.contains(p)) {
                            continue;
                    }

                    // create or update it
                    map.put(c, p);

                    // continue to match the rest
                    if (isMatch(str, k + 1, pat, j + 1, map, set)) {
                            return true;
                    }

                    // backtracking
                    map.remove(c);
            }

            // we've tried our best but still no luck
            return false
```

Shortest Distance From All Buildings

```
int[] delta = {0, -1, 0, 1, 0};
    int min = Integer.MAX_VALUE;
    public int shortestDistance(int[][] grid) {
        if (grid.length == 0 || grid[0].length == 0) {
```

```
            return 0;
        }
        int m = grid.length;
        int n = grid[0].length;
        int start = 1;
        int dist[][] = new int[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (grid[i][j] == 1) {
                    bfs(grid, i, j, --start, dist);
                }
            }
        }
        return min == Integer.MAX_VALUE ? -1 : min;
    }
    public void bfs(int[][] grid, int row, int col, int start, int[][] dist) {
        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{row, col});
        int level = 0;
        min = Integer.MAX_VALUE;
        while (!q.isEmpty()) {
            level++;
            int size = q.size();
            for (int k = 0; k < size; k++) {
                int[] p = q.remove();
                for (int d = 1; d < delta.length; d++) {
                    int newRow = p[0] + delta[d - 1];
                    int newCol = p[1] + delta[d];
                    if (newRow >= 0 && newRow < grid.length && newCol >= 0 && newCol < grid[0].length
&& grid[newRow][newCol] == start) {
                        q.add(new int[]{newRow, newCol});
                        grid[newRow][newCol]--;
                        dist[newRow][newCol] += level;
                        min = Math.min(min, dist[newRow][newCol]);
                    }
                }
            }
        }
    }
```

Maze I

```
public boolean hasPath(int[][] maze, int[] start, int[] destination) {

        if (maze.length == 0 || maze[0].length == 0) {
            return false;
        }
        int m = maze.length;
        int n = maze[0].length;
        boolean visited[][] = new boolean[m][n];
        int dirs[][] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
        Queue<int[]> q = new LinkedList<>();
        q.add(start);
        while (!q.isEmpty()) {
            int[] entry = q.remove();
            if (entry[0] == destination[0] && entry[1] == destination[1]) {
                return true;
            }
            for (int dir[] : dirs) {
                int x = entry[0] + dir[0];
                int y = entry[1] + dir[1];
                while (x >= 0 && x < maze.length && y >= 0 && y < maze[0].length && maze[x][y] == 0) {
                    x += dir[0];
                    y += dir[1];
                }
                x -= dir[0];
                y -= dir[1];
                if (!visited[x][y]) {
                    visited[x][y] = true;
                    q.add(new int[]{x, y});
                }
            }
        }
        return false;
    }
```

Maze II – O(mn log (mn)

```
public int shortestDistance(int[][] maze, int[] start, int[] destination) {
        int m = maze.length;
        int n = maze[0].length;
        int dist[][] = new int[m][n];
        for (int[] row: dist)
            Arrays.fill(row, Integer.MAX_VALUE);
        dist[start[0]][start[1]] = 0;
        shortest(maze, start, destination, dist);
        return dist[destination[0]][destination[1]] == Integer.MAX_VALUE ? -1 : dist[destination[0]]
[destination[1]];
    }

    public void shortest(int[][] maze, int[] start, int[] destination, int[][] dist) {
        PriorityQueue<int []> pq = new PriorityQueue<>((a, b) -> a[2] - b[2]);
        pq.offer(new int[]{start[0], start[1], 0});
        int[][] dirs = {{0, 1}, {0, -1}, {-1, 0}, {1, 0}};
        while (!pq.isEmpty()) {
            int locs[] = pq.poll();
            for (int dir[] : dirs) {
                int x = locs[0] + dir[0];
                int y = locs[1] + dir[1];
                int count = 0;
                while (x >= 0 && x < maze.length && y < maze[0].length && y >= 0 && maze[x][y] == 0) {
                    x += dir[0];
                    y += dir[1];
                    count++;
                }
                x -= dir[0];
                y -= dir[1];
                if (dist[start[0]][start[1]] + count < dist[x][y]) {
                    dist[x][y] = dist[start[0]][start[1]] + count;
                    pq.offer(new int[]{x, y, dist[x][y]});
                }
            }
        }
    }
```

Dungeons Game

```
public int calculateMinimumHP(int[][] dungeon) {
        int m = dungeon.length;
        int n = dungeon[0].length;
        int dp[] = new int[n + 1];
        Arrays.fill(dp, Integer.MAX_VALUE);
        dp[n - 1] = 1;
        for (int i = m - 1; i >= 0; i--) {
            for (int j = n - 1; j >= 0; j--) {
                dp[j] = Math.min(dp[j], dp[j + 1]) - dungeon[i][j];
                dp[j] = Math.max(dp[j], 1);
            }
        }
        return dp[0];
    }
```

Distinct Subsequences

```
public int numDistinct(String s, String t) {
        int m = s.length();
        int n = t.length();
        int dp[][] = new int[m + 1][n + 1];
        for (int i = 0; i < m; i++) {
            dp[i][0] = 1;
        }
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (s.charAt(i - 1) == t.charAt(j - 1)) {
                    dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];
                } else {
                    dp[i][j] = dp[i - 1][j];
                }
            }
        }
        return dp[m][n];
```

```
    }
```

Remove Invalid Paranthesis – O(nk)

```
public List<String> removeInvalidParentheses(String s) {
                List<String> ans = new ArrayList<>();
                remove(s, ans, 0, 0, new char[]{'(', ')'});
                return ans;
            }

            public void remove(String s, List<String> ans, int last_i, int last_j,  char[] par) {
                int stack = 0;
                for (int i = last_i; i < s.length(); ++i) {
                        if (s.charAt(i) == par[0]) {
                                stack++;
                        }
                        if (s.charAt(i) == par[1]) {
                                stack--;
                        }
                        if (stack >= 0) {
                                continue;
                        }
                        for (int j = last_j; j <= i; ++j) {
                                if (s.charAt(j) == par[1] && (j == last_j || s.charAt(j - 1) !=
par[1])) {
                                        remove(s.substring(0, j) + s.substring(j + 1, s.length()),
ans, i, j, par);
                                }
                        }
                        return;
                }
                String reversed = new StringBuilder(s).reverse().toString();
                if (par[0] == '(') {// finished left to right
                        remove(reversed, ans, 0, 0, new char[]{')', '('});
                } else {// finished right to left
                        ans.add(reversed);
                }
            }
```

Palindrome Paritioning II

Min cuts to partition a palindrome

```
public int minCut(String s) {
        char c[] = s.toCharArray();
        int n = s.length();
        int cut[] = new int[n];
        boolean pal[][] = new boolean[n][n];
        for (int i = 0; i < n; i++) {
            int min = i;
            for (int j = 0; j <= i; j++) {
                if ((c[j] == c[i]) && (j + 1 > i - 1 || pal[j + 1][i - 1])) {
                    pal[j][i] = true;
                    min = j == 0 ? 0 : Math.min(min, cut[j - 1] + 1);
                }
            }
            cut[i] = min;
        }
        return cut[n - 1];
    }
```

#Shortest Palindrome added to the front of the string - KMP

```
public String shortestPalindrome(String s) {
        String temp = s + "#" + new StringBuilder(s).reverse().toString();
        int[] table = getTable(temp);
        return new StringBuilder(s.substring(table[table.length - 1])).reverse().toString() + s;
    }
    public int[] getTable(String s) {
    //get lookup table
    int[] table = new int[s.length()];

    int i = 1;
    int j = 0;
    while (i < s.length()) {
        if (s.charAt(i) == s.charAt(j)) {
            table[i] = j + 1;
            i++;
            j++;
```

```
        } else {
            if (j != 0) {
                j = table[j - 1];
            } else {
                table[i] = 0;
                i++;
            }
        }
    }
    return table;
  }
```

Generalized Abbrevations

```
public List<String> generateAbbreviations(String word) {
        List<String> result = new ArrayList<>();
        if (word == null || word.length() == 0) {
            result.add("");
            return result;
        }
        generate(word, result, 0, "", 0);
        return result;
    }
    public void generate(String word, List<String> result, int index, String curr, int count) {
        if (index == word.length()) {
            if (count > 0) {
                curr += count;
            }
            result.add(curr);
            return;
        }
        generate(word, result, index + 1, curr, count + 1);
        generate(word, result, index + 1, curr + (count > 0 ? count : "") + word.charAt(index), 0);
    }
```

Russian Doll Envelope – Sort by ascending width. If widths are equal descend heights.

```
public int maxEnvelopes(int[][] envelopes) {
        Arrays.sort(envelopes, new Comparator<int[]>(){
        @Override
        public int compare(int[] arr1, int[] arr2){
            if(arr1[0] == arr2[0])
                return arr2[1] - arr1[1];
            else
                return arr1[0] - arr2[0];
       }
        });

        int dp[] = new int[envelopes.length];
        int len = 0;
        for (int envelope[] : envelopes) {
            int index = Arrays.binarySearch(dp, 0, len, envelope[1]);
            if (index < 0) {
                index = - (index + 1);
            }
            dp[index] = envelope[1];
            if (index == len) {
                len++;
            }
        }
        return len;
    }
```

Optimal Account Balancing

```
public int minTransfers(int[][] transactions) {
        Map<Integer, Long> map = new HashMap<>();
        for (int t[] : transactions) {
            Long val1 = 0L;
            Long val2 = 0L;
            if (map.containsKey(t[0])) {
                val1 = map.get(t[0]);
            }
            if (map.containsKey(t[1])) {
                val2 = map.get(t[1]);
            }
            map.put(t[0], val1 - t[2]);
            map.put(t[1], val2 + t[2]);
```

```
        }
            List<Long> list = new ArrayList<>();
            for (long val : map.values()) {
                if (val != 0) {
                    list.add(val);
                }
            }
            Long[] debts = new Long[list.size()];
            debts = list.toArray(debts);
            return dfs(debts, 0, 0);

    }
    public int dfs(Long[] debts, int pos, int count) {
        while (pos < debts.length && debts[pos] == 0) {
            pos++;
        }
        if (pos >= debts.length) {
            return count;
        }
        int res = Integer.MAX_VALUE;
        for (int i = pos + 1; i < debts.length; i++) {
            if (debts[i] * debts[pos] < 0) {
                debts[i] += debts[pos];
                res = Math.min(res, dfs(debts, pos + 1, count + 1));
                debts[i] = debts[i] - debts[pos];
            }
        } return res;
      }
```

Minesweeper

```
int[][] dirs = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}, {1, 1}, {1, -1}, {-1, 1}, {-1, -1}};
    public char[][] updateBoard(char[][] board, int[] click) {
        int m = board.length;
        int n = board[0].length;
        int x = click[0];
        int y = click[1];
        if (board[x][y] == 'M') {
            board[x][y] = 'X';
        } else {
            dfs(board, m, n, x, y);
        }
        return board;
    }
    public void dfs(char[][] board, int m, int n, int x, int y) {
        if (x < 0 || x >= m || y < 0 || y >= n || board[x][y] != 'E') {
            return;
        }
        int mine = getAdjMines(board, m, n, x, y);
        if (mine > 0) {
            board[x][y] = (char) ('0' + mine);
        } else {
            board[x][y] = 'B';
            for (int dir[] : dirs) {
                dfs(board, m, n, x + dir[0], y + dir[1]);
            }
        }
    }


    public int getAdjMines(char[][] board, int m, int n, int x, int y) {
        int count = 0;
        for (int i = x - 1; i <= x + 1; i++) {
            for (int j = y - 1; j <= y + 1; j++) {
                if (i < 0 || i >= m || j < 0 || j >= n || board[i][j] != 'M') {
                    continue;
                }
                if (board[i][j] == 'M') {
                   count++;
                }
            }
        }
        return count;
    }
```

Find Friends Circle – O(n^2 logn)

```
public int findCircleNum(int[][] M) {
        int n = M.length;
        int arr[] = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = i;
        }
        int count = n;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (M[i][j] == 1) {
                   int x = root(arr, i);
                   int y = root(arr, j);
                   if (x == y) {
                       continue;
                   }
                   arr[x] = y;
                   count--;
                }
            }
        }
        return count;
    }
```

Sparse Matrix Multiplication

```
public int[][] multiply(int[][] A, int[][] B) {
        int m = A.length;
        int n = A[0].length;
        int nB = B[0].length;
        int C[][] = new int[m][nB];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (A[i][j] != 0) {
                    for (int k = 0; k < nB; k++) {
                        if (B[j][k] != 0) {
                            C[i][k] += A[i][j] * B[j][k];
                        }return C
```

Maximal Square

```
if(matrix.length == 0) return 0;

        int m = matrix.length;
        int n = matrix[0].length;
        int res = 0;
        int dp[][] = new int[m + 1][n + 1];
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (matrix[i- 1][j - 1] == '1') {
                    dp[i][j] = Math.min(dp[i - 1][j], Math.min(dp[i][j - 1], dp[i - 1][j - 1])) + 1;
                    res = Math.max(res, dp[i][j]);
                }
            }
        }
        return res * res;
    }
```

01 Matrix

```
public int[][] updateMatrix(int[][] matrix) {
        int m = matrix.length;
        int n = matrix[0].length;
        int dist[][] = new int[m][n];
        for (int rows[] : dist) {
            Arrays.fill(rows, Integer.MAX_VALUE);
        }
        Queue<int[]> q = new LinkedList<>();
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (matrix[i][j] == 0) {
                   dist[i][j] = 0;
                    q.add(new int[]{i, j});
                }
            }
        }
        int dirs[][] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
        while (!q.isEmpty()) {
            int cell = q.remove();
            for (int dir[] : dirs) {
```

```
                int newRow = cell[0] + dir[0];
                int newCol = cell[1] + dir[1];
                if (newRow >= 0 || newRow < m || newCol >= 0 || newCol < n) {
                    if (dist[newRow][newCol] > 1 + dist[cell[0]][cell[1]]) {
                        dist[newRow][newCol] = 1 + dist[cell[0]][cell[1]]
                        q.add(new int[]{newRow, newCol});
                    }
                }
            }
        }
        return dist;
    }
```

Teemo Attacking

```
public int findPoisonedDuration(int[] timeSeries, int duration) {
        if (timeSeries == null || timeSeries.length == 0 || duration == 0) {
            return 0;
        }
        int start = timeSeries[0];
        int end = start + duration;
        int res = 0;
        for (int i = 1; i < timeSeries.length; i++) {
            if (timeSeries[i] > end) {
                res += end - start;
                start = timeSeries[i];
            }
            end = timeSeries[i] + duration;
        }
        res += end - start;
        return res;
    }
```

Binary Tree Longest Consecutive Sequence – O(n) and O(n)

```
int maxVal = 0;
    public int longestConsecutive(TreeNode root) {
        longestPath(root);
        return maxVal;
    }
    public int[] longestPath(TreeNode root) {
        if (root == null) {
            return new int[]{0, 0};
        }
        int inc = 1;
        int dec = 1;
        if (root.left != null) {
            int l[] = longestPath(root.left);
            if (root.val == root.left.val + 1) {
                inc = l[0] + 1;
            }
            if (root.val == root.left.val - 1) {
                dec = l[1] + 1;
            }
        }
        if (root.right != null) {
            int r[] = longestPath(root.right);
            if (root.val == root.right.val + 1) {
                inc = Math.max(inc, r[0] + 1);
            }
            if (root.val == root.right.val - 1) {
                dec = Math.max(dec, r[1] + 1);
            }
        }
        maxVal = Math.max(maxVal, inc + dec - 1);
        return new int[]{inc, dec};
    }
```

Decode String

```
public String decodeString(String s) {
        if (s == null || s.length() == 0) {
            return "";
        }
        Stack<Integer> count = new Stack<>();
        Stack<StringBuilder> sbStack = new Stack<>();
        StringBuilder curr = new StringBuilder();
        int k = 0;
        for (char ch : s.toCharArray()) {
            if (Character.isDigit(ch)) {
                k = k * 10 + ch - '0';
            } else if (ch == '[') {
                count.push(k);
                sbStack.push(curr);
                curr = new StringBuilder();
                k = 0;
            } else if (ch == ']') {
                StringBuilder tmp = curr;
                curr = sbStack.pop();
                int size = count.pop();
                for (int i = 0; i < size; i++) {
                    curr.append(tmp);
                }
            } else {
                curr.append(ch);
            }
        }
        return curr.toString();
    }
```

House Robber III

```
public int rob(TreeNode root) {
        int res[] = robSub(root);
        return Math.max(res[0], res[1]);
    }
    public int[] robSub(TreeNode root) {
        if (root == null) {
            return new int[2];
        }
        int l[] = robSub(root.left);
        int r[] = robSub(root.right);
        int res[] = new int[2];
        res[0] = Math.max(l[0], l[1]) + Math.max(r[0], r[1]);
        res[1] = root.val + l[0] + r[0];
        return res;
    }
```

Wildcard Matching

```
public boolean isMatch(String s, String p) {
                if(s == null || p == null) {
            return false;
        }
        boolean[][] state = new boolean[s.length() + 1][p.length() + 1];
        state[0][0] = true;
        for (int j = 1; j <= p.length(); j++) {
            if (p.charAt(j - 1) == '*' && state[0][j - 1]) {

                    state[0][j] = true;
            }
        }
        for (int i = 1; i < state.length; i++) {
            for (int j = 1; j < state[0].length; j++) {
                if (s.charAt(i - 1) == p.charAt(j - 1) || p.charAt(j - 1) == '?') {
                    state[i][j] = state[i - 1][j - 1];
                }
                if (p.charAt(j - 1) == '*') {
                        state[i][j] = state[i - 1][j] || state[i][j - 1] ;
                }
            }
        }
        return state[s.length()][p.length()];
    }
```

Regular Expression Matching

```
public boolean isMatch(String s, String p) {
        if(s == null || p == null) {
            return false;
        }
        boolean[][] state = new boolean[s.length() + 1][p.length() + 1];
        state[0][0] = true;
        // no need to initialize state[i][0] as false
        // initialize state[0][j]
        for (int j = 1; j <= p.length(); j++) {
            if (p.charAt(j - 1) == '*') {
                if (state[0][j - 1]  || (j > 1 && state[0][j - 2])) {
                    state[0][j] = true;
                }
            }
        }
        for (int i = 1; i < state.length; i++) {
            for (int j = 1; j < state[0].length; j++) {
                if (s.charAt(i - 1) == p.charAt(j - 1) || p.charAt(j - 1) == '.') {
                    state[i][j] = state[i - 1][j - 1];
                }
                if (p.charAt(j - 1) == '*') {
                    if (s.charAt(i - 1) != p.charAt(j - 2) && p.charAt(j - 2) != '.') {
                        state[i][j] = state[i][j - 2];
                    } else {
                        state[i][j] = state[i - 1][j] || state[i][j - 2] ;
                    }
                }
            }
        }
        return state[s.length()][p.length()];
    }
```

Word Break II - O(n^3)

```
static HashMap<String,List<String>> map = new HashMap<String,List<String>>();
           public static List<String>  wordBreak(String s, Set<String> wordDict) {
                List<String> res = new ArrayList<String>();
                if(s == null || s.length() == 0) {
                        return res;
                }
                if(map.containsKey(s)) {
                        return map.get(s);
                }
                if(wordDict.contains(s)) {
                        res.add(s);
                }
                for(int i = 1 ; i < s.length() ; i++) {
                        String t = s.substring(i);
                        if(wordDict.contains(t)) {
                                List<String> temp = wordBreak(s.substring(0 , i) , wordDict);
                                if(temp.size() != 0) {
                                        for(int j = 0 ; j < temp.size() ; j++) {
                                                res.add(temp.get(j) + " " + t);
                                        }
                                }
                        }
                }
                map.put(s , res);
                return res;
           }
```

Skyline Problem

```
public List<int[]> getSkyline(int[][] buildings) {
        List<int[]> results = new ArrayList<>();
        List<int[]> heights = new ArrayList<>();
        for (int[] building : buildings) {
            heights.add(new int[]{building[0], -building[2]});
            heights.add(new int[]{building[1], building[2]});
        }

        Collections.sort(heights, (a, b) -> {
            if(a[0] != b[0])
                return a[0] - b[0];
            return a[1] - b[1];
        });
```

```
        PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
        int prev = 0;
        pq.add(0);
        for (int[] height : heights) {
            if (height[1] < 0) {
                pq.add(-height[1]);
            } else {
                pq.remove(height[1]);
            }
            int curr = pq.peek();
            if (curr != prev) {
                results.add(new int[]{height[0], curr});
                prev = curr;
            }
        }
        return results;
    }
```

Rearrange String K distance apart

```
public String rearrangeString(String s, int k) {
        if (s == null || s.length() == 0) {
                        return null;
                }
                if (k == 0) {
                        return s;
                }
                HashMap<Character, Integer> map = new HashMap<>();
                for (int i = 0; i < s.length(); i++) {
                        char ch = s.charAt(i);
                        if (map.containsKey(ch)) {
                                map.put(ch, map.get(ch) + 1);
                        } else {
                                map.put(ch, 1);
                        }
                }
           PriorityQueue<Map.Entry<Character, Integer>> pq = new PriorityQueue<Map.Entry<Character,
Integer>>(new Comparator<Map.Entry<Character, Integer>> () {

                        @Override
                        public int compare(Map.Entry<Character, Integer> o1, Map.Entry<Character,
Integer> o2) {
                                if (o1.getValue() == o2.getValue()) {
                                        return Integer.compare(o1.getKey(), o2.getKey());
                                }
                                return Integer.compare(o2.getValue(), o1.getValue());
                        }

                });
                pq.addAll(map.entrySet());

                if (pq.isEmpty()) {
                        return "";
                }

                Queue<Map.Entry<Character, Integer>> q = new LinkedList<>();
                StringBuilder sb = new StringBuilder();
                while (!pq.isEmpty()) {
                        Map.Entry<Character, Integer> p = pq.remove();
                        sb.append(p.getKey());
                        int freq = p.getValue() - 1;
                        p.setValue(freq);
                        q.add(p);
                        if(q.size() == k) {
                                Map.Entry<Character, Integer> m = q.poll();
                                if (m.getValue() > 0) {
                                        pq.add(m);
                                }
                        }
                }
                return sb.length() == s.length() ? sb.toString() : "";
```

Zigzag Iterator

```
  LinkedList<Iterator> list;
    public ZigzagIterator(List<Integer> v1, List<Integer> v2) {
        list = new LinkedList<>();
        if (!v1.isEmpty()) {
            list.add(v1.iterator());
        }
        if (!v2.isEmpty()) {
            list.add(v2.iterator());
        }
    }

    public int next() {
        Iterator itr = list.remove();
        int num = (Integer) itr.next();
        if (itr.hasNext()) {
            list.add(itr);
        }
        return num;
    }

    public boolean hasNext() {
        return !list.isEmpty();
    }
```

LFU Cache

```
    HashMap<Integer, Integer> vals;
    HashMap<Integer, Integer> counts;
    HashMap<Integer, LinkedHashSet<Integer>> lists;
    int min = -1;
    int cap;
    public LFUCache(int capacity) {
        cap = capacity;
        vals = new HashMap<>();
        counts = new HashMap<>();
        lists = new HashMap<>();
        lists.put(1, new LinkedHashSet<>());
    }

    public int get(int key) {
        if (!vals.containsKey(key)) {
            return -1;
        }
        incrementCount(key);
        return vals.get(key);
    }

    public void put(int key, int value) {
        if(cap <= 0)  {
            return;
        }
        if (vals.containsKey(key)) {
            vals.put(key, value);
            incrementCount(key);
            return;
        }
        if (vals.size() >= cap) {
            int delKey = lists.get(min).iterator().next();
            lists.get(min).remove(delKey);
            vals.remove(delKey);


        }

        vals.put(key, value);
        counts.put(key, 1);
        min = 1;
        lists.get(1).add(key);
    }
    public void incrementCount(int key) {

                int count = counts.get(key);
```

```
                counts.put(key, count + 1);
                lists.get(count).remove(key);
                if (count == min && lists.get(count).size() == 0) {
                        min++;
                }
                count++;
                if (!lists.containsKey(count)) {
                        lists.put(count, new LinkedHashSet<Integer>());
                }
                lists.get(count).add(key);
        }
}
```