2. Recursion

Coding Ninjas Problems

Recursion (Beginner)

- function calling itself
- works of the Principle of Mathematical Induction
  - Base Case - Prove F(0) or F(1) is true
  - Induction Hypothesis - Assume that F(k) is true
  - Induction Step - Prove that F(k+1) is also true, assuming F(k) is true
- Procedure
  - First use PMI on the given formula.
  - If PMI applies, recursion can be used.
  - Three steps are required
    1. Base Case
    2. Recursion Step
    3. Small Calc
  - Steps 2 & 3 are interchangeable.
- Recursion can do mostly all the problems that iteration (for, while loops) can do and vice versa
  - Eg. Remove char '*' from a*bcd***efgh*i* uses O(n*n) like for loop within for loop or for loop within recursive function
- Eg 1. Find factorial

```
int factorial(int n){
   if (n==0) {return 1;}
   else return n*factorial(n-1);
}
int main(){
int n;
cin>>n;
cout<<Factorial(n);
}
```

- Eg 2. Find nth power of a given number x i.e. x^n

```
int power(int x, int n) {
       if (n==0){return 1;}
       else {
       return x*power(x,n-1);
   }
}
```

- Eg 3. Print 1 to n for given n

```
void print(int n){
     if(n == 1){
         cout << n << " ";
         return; // Imp as to break the recurse.
     }           // Else would continue in -ve nmbrs
     print(n-1);
     cout << n << " ";
}
```

- Eg 4. Count the number of digits of a given integer

```
    int count (int n){
      if (n==0){return 0;} // As 1-digit/10 is zero
      else
      return 1+count(n/10);
    }
```

- Eg 5. Print nth Fibonacci no.

```
    int fib(int n){
       if (n==0) return 0;
       else if (n==1) return 1;
       else return fib(n-1)+fib(n-2);
    }
```

- Eg 6. Find whether the given array is sorted (Ascndg).

```
    bool is_sorted(int a[],int size){
       if (size==0||size==1) return true;
       if (a[0]>a[1]) return false; // Compares the first two posn of
       the array
       return is_sorted(a+1,size-1);// Shifts initial posn and reduces
       size by 1
    }
```

- Eg 7. Find the sum of all elements of an array of length n.

```
    int sum(int a[], int n) {
          if (n==0) return 0; // If size is 0, add 0 to the sum.
          else return a[0]+sum(a+1, n-1); // Add the initial posn and
                                          // then shift initial posn
                   and reduces size by 1
    }
```

- Eg 8. Find if given int x is found or not in the given array a of length n.

```
        bool checkNumber(int a[], int size, int x) {
                if (size==0) return false; // Size = 0 means that all the
elements of the array has been compared
                else if (x==a[0]) return true; // Comparing x with first
element of the array a
                return checkNumber(a+1, size-1, x); // Recursively shifting
the first element of the array
        }                                             // and reducing the size
        by 1 and the comparing with x
```

- Eg 9. Given an array of length N and an integer x, you need to find and return the first index of integer x

```
        present in the array. Return -1 if it is not present in the array.
        int firstIndex(int a[], int size, int x) {
                if (size==0) return -1;
                if (a[0]==x) return 0;
                int rec_call=firstIndex(a+1, size-1, x);
                if (rec_call!=-1) {
                    return 1+rec_call;
                }
                else return rec_call;
        }
```

- Eg 10. Given an array of length N and an integer x, you need to find and return the last index of integer x present in the   array. Return -1 if it is not present in the array. You should start traversing your array from 0, not from (N - 1).

```
        int lastIndex(int a[], int size, int x) {
                    if (size>1){
                    int rec_call=lastIndex(a+1, size-1, x);
                    if (rec_call!=-1) return 1+rec_call;
                    }
                     if (a[0]==x) return 0;
                    else return -1;
        }

        or simple line change in Eg 9. as below
        int lastIndex(int a[], int size, int x) {
                if (size==0) return -1;
                //if (a[0]==x) return 0; // Just shift this line to after if
(rec_call!=1) condition
(rec_call!=-1) condition
            int rec_call=lastIndex(a+1, size-1, x);
            if (rec_call!=-1) {
                return 1+rec_call;
            }
            if (a[0]==x) return 0;
            return rec_call;
        }
```

- Eg 11. Given an array of length N and an integer x, you need to find all the indexes where x is present in the input array. Save all the indexes in an array (in increasing order).
  Do this recursively. Indexing in the array starts from 0.
  a.) RIGHT to LEFT

```
  int allIndexes(int input[], int size, int x, int output[]) {
       if (size==0) return 0;
       int rec_Call=allIndexes(input, size-1, x, output);
       if (input[size-1]==x){
           output[rec_Call]=size-1;
           return 1+rec_Call;
       }
       else return rec_Call;
  }
```

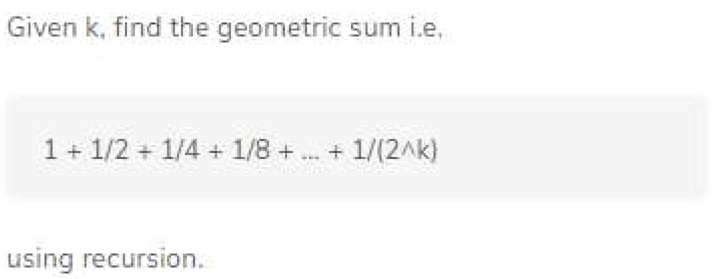
- Eg 12. Multiply two numbers m and n recursively. Hint - m*n = m + m + ... m (n times)

```
     int multiplyNumbers(int m, int n) {
          if (n==0) return 0;
          else return m + multiplyNumbers(m, n-1);
     }
```

- Eg 13. Count the number of zeros recursively in a given integer.

```
     int countZeros(int n) {
          if (n==0) return 1;
          else if (n<=9&&n>0) return 0;
          else {
                int rem=n%10;
                if (rem==0) return 1+countZeros(n/10);
                else return countZeros(n/10);
          }
     }
```

- Eg 14.



Given k, find the geometric sum i.e.

```
1 + 1/2 + 1/4 + 1/8 + ... + 1/(2^k)
```

using recursion.

```
A.) From left to right (using a pattern formula for sum)
    double geometricSum(int k) {
            if (k==0) return 1;
            if (k==1) return 1.5;
            else {
            double term = 0.5*(geometricSum(k-1)-geometricSum(k-
2))+geometricSum(k-1);
            return term;
            }
}
```

Hint - need to get the formula of the pattern

Note - f(n) = 1/2 * f(n-1) is incorrect as we need the sum itself needs to be returned.

Hence, f(n) = 0.5*(f(n-1)-f(n-2))+f(n-1) is the correct pattern where f(n) denotes the sum till k=n.

```
B.) From right to left (but uses power function)
    double geometricSum(int k){
            if (k==0) return 1;
            else {
                double rec_call=geometricSum(k-1);
                return rec_call+pow(0.5,k);
            }
        }
```

- Eg 15. Write a recursive function that returns the sum of the digits of a given integer.

```
    int sumOfDigits(int n) {
            if (n==0) return 0;
            int rec_call=sumOfDigits(n/10);
            return rec_call+n%10;
    }
```

- **Recursive operations on string**
- Eg 16. Write a recursive function that returns the length of a string as char arr[];

```
  int length(char s[]){
     if (s[0]=='\0') return 0;
     else return 1+length(s+1);
  }
```

- Eg 17. Given an input string S and two characters c1 and c2, you need to replace recursively every occurrence of character c1 with character c2 in the given string.

```
     void replaceCharacter(char s[], char c1, char c2) { // void
     function is used here because
            if (s[0]=='\0') return;    //An array passed into a function
is always passed by address,
            else if (s[0]==c1) s[0]=c2; //since the array's name IS a
variable that stores its address
             replaceCharacter(s+1, c1, c2);
     }
```

*** When we pass an array to a method as an argument, actually the address of the array in the memory is passed (reference). Therefore, any changes to this array in the method will affect the array.**
So we can pass the array as func(arr) or func(arr+1) to both
void func(int/char arr[]) or void func(int/char *arr) [IMPORTANT]

- Eg 18. Remove Consecutive Duplicates Recursively. wwxxxyyzz to wxyz

```
        void removeConsecutiveDuplicates(char *s) {
               if (s[0]=='\0') return;
               else if (s[0]!=s[1]) removeConsecutiveDuplicates(s+1);
               else if (s[0]==s[1]) {
                   for (int i=0; s[i]!='\0'; i++){
                       s[i]=s[i+1];
                   }
         removeConsecutiveDuplicates(s);
               }
        }
```

- Eg 19. Given a string, compute recursively a new string where all 'x' chars have been removed.

- (xaxxxbxxcxdefxxg to abcdefg)
- O(n^2) i.e. for loop within recursion

```
void removeX(char s[]) {
        if (s[0]=='\0') return;
        if (s[0]!='x') removeX(s+1);
        if (s[0]=='x'){
            for (int i=0; s[i]!='\0'; i++){
                    s[i]=s[i+1];
                }
            removeX(s);
        }
}
```

- Eg 20. Given a string S, compute recursively a new string where identical chars that are adjacent in the original string are separated from each other by a "*".
  - **Sample Input 1 :**

    ```
    hello
    ```

    Sample Output 1:

    ```
    hel*lo
    ```

    Sample Input 2 :

    ```
    aaaa
    ```

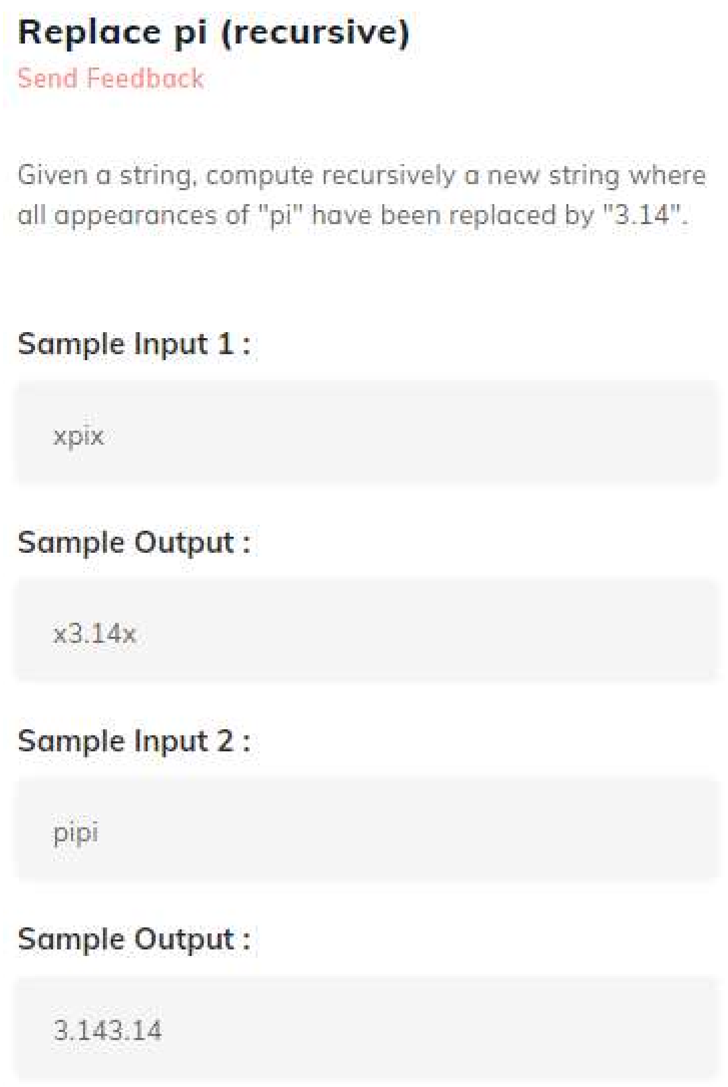
    Sample Output 2 :

    ```
    a*a*a*a
    ```

```
void pairStar(char s[]) {
      if (s[0]=='\n') return;
      else if (s[0]!=s[1]) pairStar(s+1);
      else if (s[0]==s[1]){
          int end = 0;
          for (int i=0; ; i++){
              if (s[i]=='\0'){
              end = i;
```

```
                        break;
                        }
                    }
                while (end!=0){
                    s[end+1]=s[end];
                    end--;
                }
                s[1]='*';
                pairStar(s+2); // Shifted by 2 because s[0] and s[1] is taken
care of. Now, recur from s[2].
            }
      }
```

- Eg 21.



Replace pi (recursive)

Send Feedback

Given a string, compute recursively a new string where all appearances of "pi" have been replaced by "3.14".

Sample Input 1 :

```
xpix
```

Sample Output :

```
x3.14x
```

Sample Input 2 :

```
pipi
```

Sample Output :

```
3.143.14
```

```
void replacePi(char s[]) {
         if (s[0]=='\n') return;
         bool Pi = (s[0]=='p'&&s[1]=='i')?true:false;
```

```
            if (!Pi) replacePi(s+1);
            else if (Pi) {
                int end = 0;
                for (int i=0; ; i++){
                    if (s[i]=='\0'){
                      end = i; // end now has the last index
                      break;
                    }
                }
                while (end>1){          // need the shifting at all posn
except at s[0] and s[1]
                    s[end+2]=s[end]; // pi (2 char) is replaced by 3.14 (4
char)
                    end--;                       // therefore shifted by 2
places
                }
            s[0]='3'; s[1]='.'; // Replacement done
            s[2]='1'; s[3]='4';
            replacePi(s+4);              // Shift by 4 places as 3.14 takes
the size of 4 char
            }
      }
```

- Eg 22. Write a recursive function to convert a given string into the number it represents. That is input will be a numeric string that contains only numbers, you need to convert the string into corresponding integer and return the answer. (Convert a string to integer)
    1. Method 1 - By calculating the length of the current string in recur. (Either left to right or right to left)

```
      int stringToNumber(char s[]) {
            if (s[1]=='\0') return (s[0]-'0'); // Base Case implies single
digit be returned as it is
            int r1 = stringToNumber(s+1); // Recurs till base case
            int end = 0;
            while (s[end]!='\0') end++; // end calculates the number of
digits in the current string
            int powerOf10=1;
            while(end>1) {
                powerOf10*=10; // powerOf10 calculates the power of 10 to be
multiplied with s[0];
                end--;              // powerof10 is one less than no. of
digits in the current string
            }
            int r2 = r1+(s[0]-'0')*powerOf10;
            return r2;
      }
```

// Note - char is converted to int as => int a = (character - '0'); Eg. here its int r = s[0]-'0';
or int a = int(character) - 48; As 48 is the ASCII code for '0'.

1. Method 2 - Without calculating the length of the current string but reducing its size from
 right to left i.e. s[end-1]=s[end] or '\0';

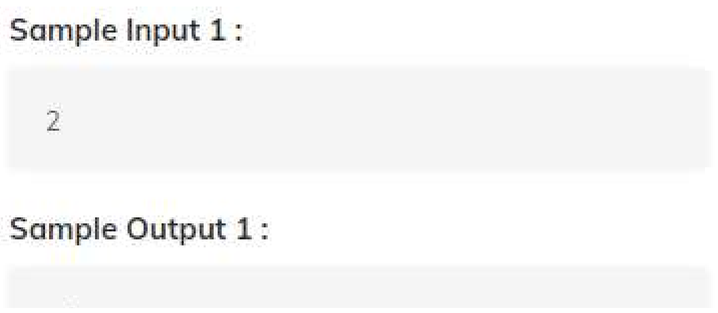
```
int stringToNumber(char s[]) {
      if (s[0]=='\0') return 0;
      int end=0;
      while (s[end]!='\0') end++;
      int e = s[end-1]-'0'; // Converting char to int
      s[end-1]=s[end]; // Reducing the size of the string from left
      int r = stringToNumber(s);
      return r*10+e;
}
```

- Eg 23. **<u>Tower of Hanoi</u>**

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move all disks from source rod to destination rod using third rod (say auxiliary). The rules are:

```
1) Only one disk can be moved at a time.
2) A disk can be moved only if it is on the top of a rod.
3) No disk can be placed on the top of a smaller disk.
```

Print the steps required to move n disks from source rod to destination rod. Source Rod is named as 'a', auxiliary rod as 'b' and destination rod as 'c'.



Sample Input 1 :

```
2
```

Sample Output 1 :

```
a b
a c
b c
```

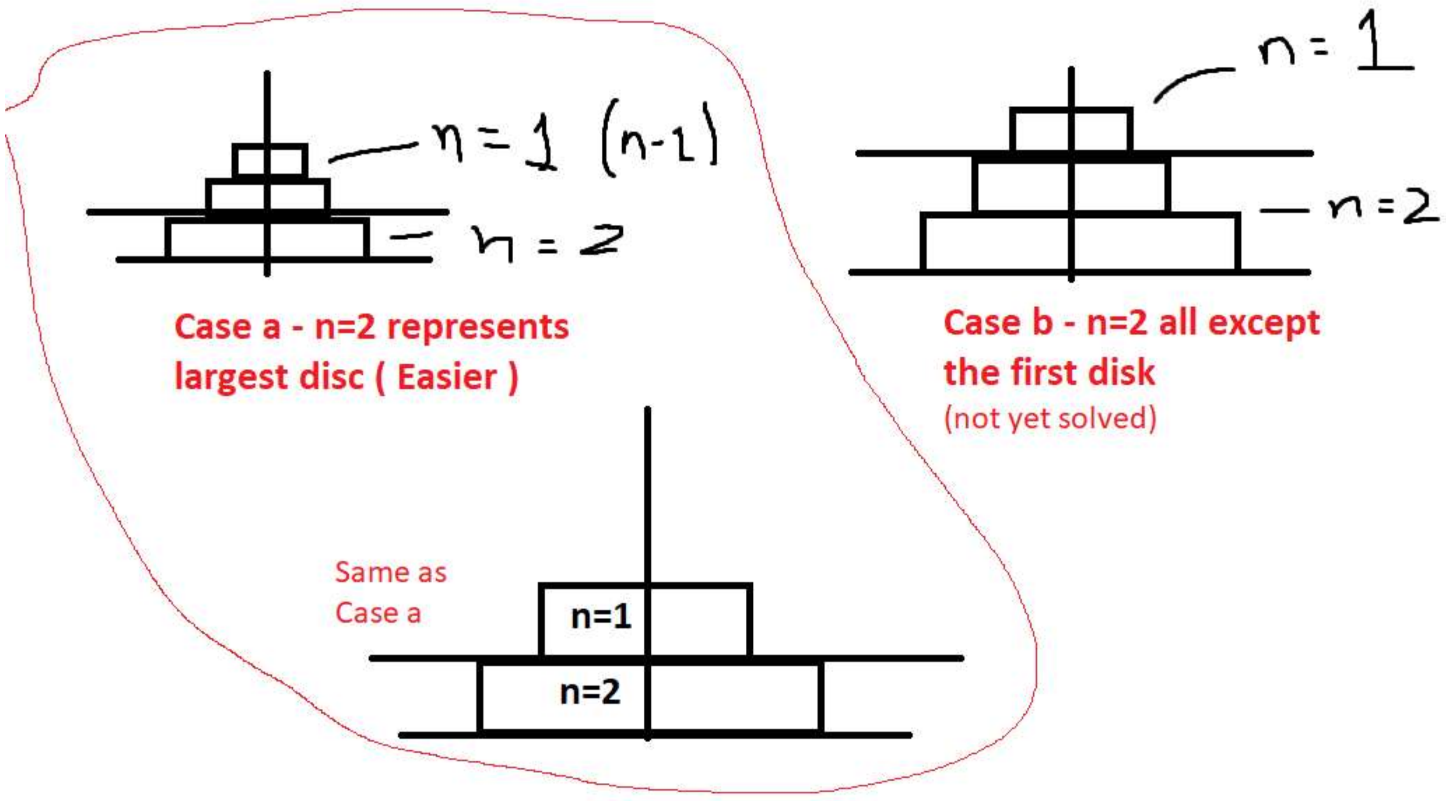
Sample Input 2 :

```
3
```

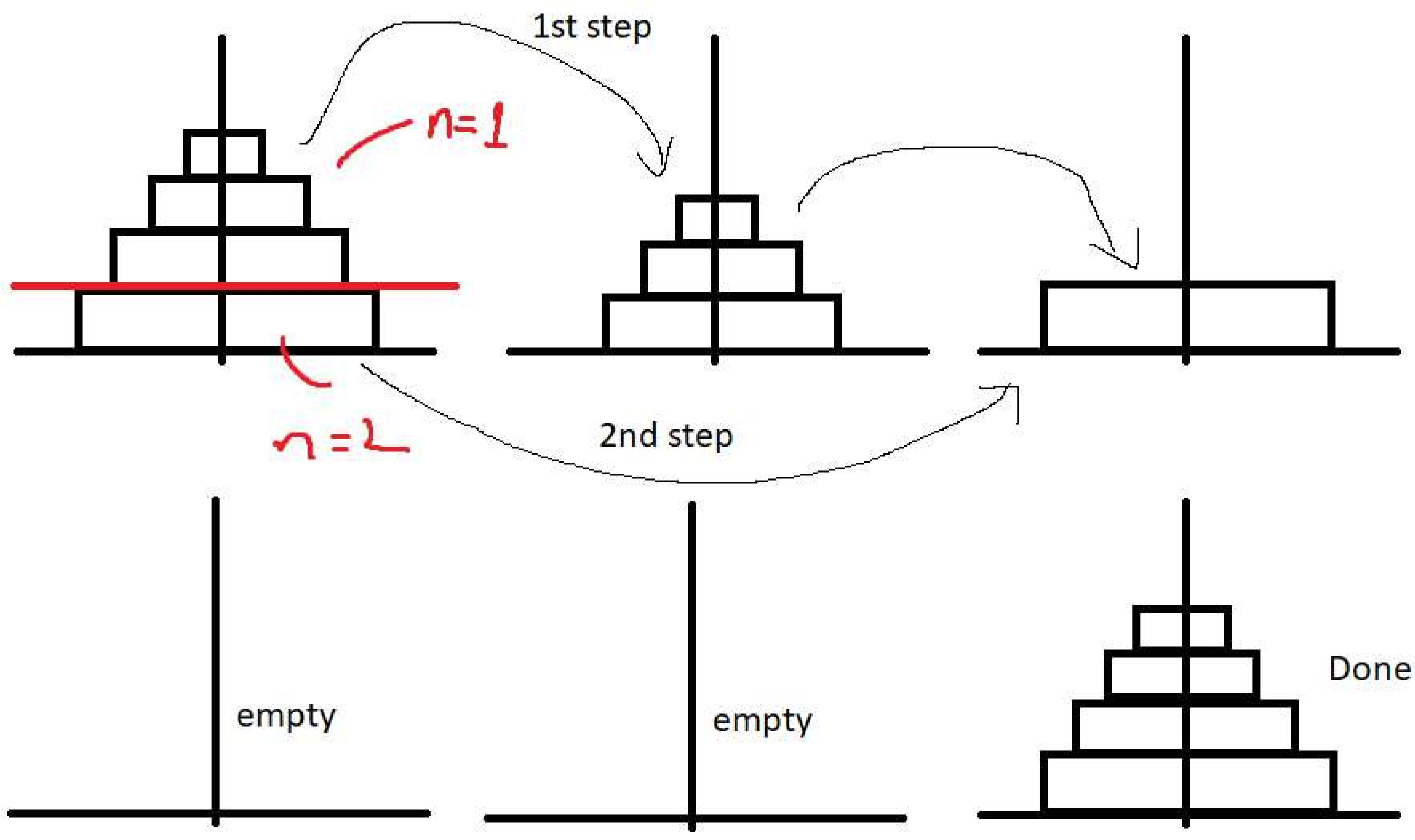
Sample Output 2 :

```
a c
a b
c b
a c
b a
b c
a c
```

Soln - Think of the disks to be divided into two parts where n=2 represents the largest disk ant n=1 represents (n-1) disks above the largest disk.


n=1 (n-1)
n=2
Case a - n=2 represents largest disc (Easier)
n=1
n=2
Case b - n=2 all except the first disk
(not yet solved)
Same as Case a
n=1
n=2

Tower of Hanoi - Soln through a recursive function



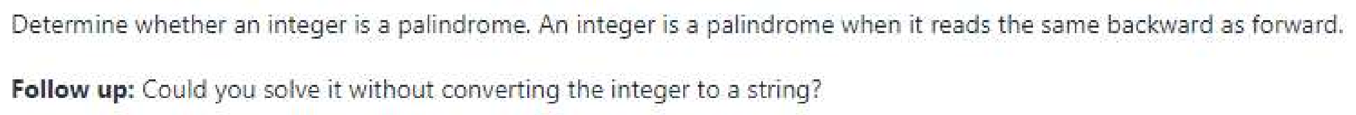
```
// Method A - When n==1 represents all the disks except the largest one
// & n==2 represents the largest disk
// i.e. recursive func handles all the disks except the largest one

void towerOfHanoi(int n, char src, char aux, char des) {
    //if (n==0) return;    // Either use n==0 or n==1 condition as both works
    if (n==1) cout<<src<<" "<<des<<"\n";
    else{
        towerOfHanoi(n-1, src, des, aux);
        cout<<src<<" "<<des<<"\n";
        towerOfHanoi(n-1, aux, src, des);
    }
}
```

- Eg 24. Check whether a given String S is a palindrome using recursion. Return true or false. (Eg. racecar - true, race - false)

```
      bool checkPalindrome(char s[]) {
         if (s[1]=='\0'||s[0]=='\0') return true; // Base case - Checks if
the string is empty or of just length 1
         int end=0;
         while (s[end]!='\0') end++;
```

```
            if (s[0]==s[end-1]){
                s[end-1]='\0';                        // Reduces the string
from the end thus changing the last character
                bool rec_Call = checkPalindrome(s+1); // Shifts the starting
position by 1
                return rec_Call;
            }
            else return false;
          }
```

- Eg 24b.



Determine whether an integer is a palindrome. An integer is a palindrome when it reads the same backward as forward.

Follow up: Could you solve it without converting the integer to a string?

```
    Method  A - Without converting the given integer to a string. (Best Soln)

bool isPalindrome(int x) {
        if (x<0) return false;
        int n=x;
        long long int rev=0;
        while (n>0){
            rev=10*rev+(n%10);     // Reversing the given integer and then
comparing
            n=n/10;
        }
        if (rev==x) return true;
        else return false;
    }

Method B - Converting the given integer to a string . (2 functions used as
recursion is used)

bool isStringPal(char s[]){
    int size=0;
    for (int i=0; s[i]!='\0'; i++){
        size++;
    }
    if (size==0||size==1) return true;
   if (s[0]==s[size-1]){
        s[size-1]='\0';               //Reducing the size of array from the
end
```

```
        return isStringPal(s+1);
    }
    else return false;

bool isPalindrome(int x) {
        if (x<0) return false;
        if (x/10==0) return true;     // i.e. if x is single digit => return
true
        char ch[10000000];
        int i=0;
        while (x>0){
            int rem = x%10;
            ch[i] = rem+'0';          // Converting integer digit to a
character digit
            //cout<<ch[i]<<" ";
            x=x/10;
            i++;
        }
        return isStringPal(ch);
    }
```

Advanced Recursion

- Eg 25. Merger sort using recursion
    - **Merge sort** is a **sorting** technique based on **divide and conquer** technique. With worst-case time complexity being O(n log n), it is one of the most respected algorithms.
    - (Uses two functions
        - a. merge_sort(int a[], int start, int end) or merge_sort(int a[], int size)
        - b. merge(int a[], int start, int end, int size)

```
void merge(int a[], int s, int e, int n){
   // int* b= new int[n]; Better way to create an array
    int b[n]; // A temporary array to store the merged array
    int i=s;
    int j=e;
    for(int k=0; k<n; k++){
        if (i>=e && j<n) { // 1st half is already copied to b
            b[k]=a[j];
            j++;
            continue; // Continue is really important
        }
        if (j==n && i<e){ // 2nd half is already copied to b
            b[k]=a[i];
            i++;
            continue;
        }
        if (a[i]<=a[j]){ // smaller values in 1st half
            b[k]=a[i];
            i++;
            continue;
        }
        if (a[i]>a[j]){ // smaller values in 2nd half
            b[k]=a[j];
            j++;
            continue;
        }
    }
    for (int k=0; k<n; k++) a[k]=b[k]; // Copy the contents of temp array b
back to main array a
}

void mergeSort(int a[], int size){
    if (size==0||size==1) return;
    int start = 0;
    int end = size-1;
    int mid = (start+end)/2;
    mid=mid+1; // As size is passed as a parameter
    mergeSort(a, mid);             // 1st half of the array
    mergeSort(a+mid, size-mid);    // 2nd half of the array
    merge(a, 0, mid, size);
}
```

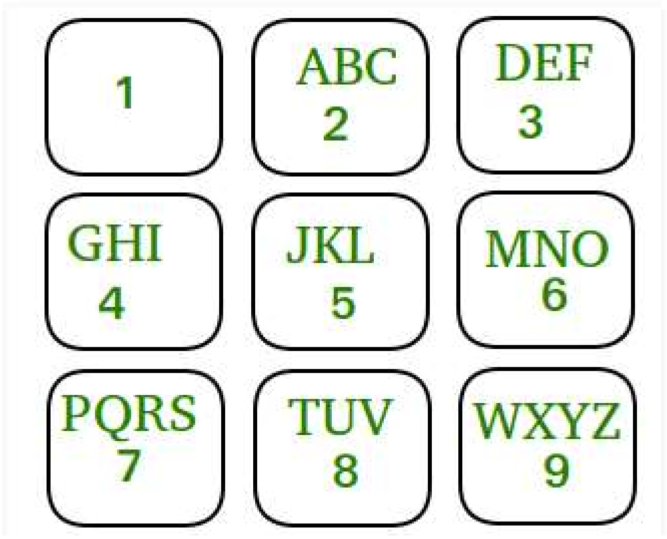
- Eg 26. **Quick Sort** - Like MergeSort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.
    - Always pick first element as pivot. // Used here
    - Always pick last element as pivot (implemented below)
    - Pick a random element as pivot.
    - Pick median as pivot.

```
int partition(int a[], int s, int e){
    int count = 0; // To keep the posn of the pivot which is a larger number
beginning from the left
    int i = 0;
    for (int j = 0; j <= e; j++){ // i = 0 is always the pivot
        if (a[i] > a[j]) count++; // Counting the position of the pivot for
the swap
    }                              // variable count only stores the posn for
values lower than pivot and not equal to
    if (count > 0) {
        int temp = a[i];
        a[i] = a[count];
        a[count] = temp; // Swapping the current larger number to create a
partition
    }
    i = 0; // Because i already has a non-zero value so, reseting it to zero
    int j = e;
    if (count < e){ // Elements left to the partition are already lower than
the pivot value if count == e
        while (i<count){
            if (a[i] >= a[count]){ // Values equal to or greater than pivot
are shifed to the right side
                int temp = a[i];
                a[i] = a[j];
                a[j] = temp;
                j--; // Because a[j] is greater than or equal to the pivot,
so dont need it to be shifted to the left
                continue;
            }
            else i++; // Only if a[i] < a[count] should i be increased
        }
    }
    return count; // Returning the pivot position after all the lower values
in the left side
}

void quickSort(int a[], int size) {
    int start = 0;
    int end = size - 1;
    if (start >= end) return;
    int p = partition(a, start, end);
    quickSort(a, p);
    quickSort(a+p+1, size-p-1);
}
```

- Eg 27. Print all the subsequences of a given string and also store the subsequence.
    - Given length = n => total ss = 2^n

```
/* This program prints the duplicates elements if the input contains
duplicate elements like
Input  "aaa"
Output a a aa a aa aa aaa
*/

#include<iostream>
#include<string>
using namespace std;

int subSq(string s, string ssq[]){
        if (s.size()==0){
        ssq[0]="";
        return 1;    // As even an empty string will return an empty string
so output string is of size 1
        }
        int rec_Call = subSq(s.substr(1), ssq);
        for (int i=0; i<rec_Call; i++){
                ssq[i+rec_Call] = s[0] + ssq[i];
                }
        return 2*rec_Call; // Returns the current size of the output array
(not 1024 but what is actually filled)
}

int main(){
        string s;
        cin>>s;
        int size = s.size();
        cout<<size<<"\n";
        // string* ssg = new string[size] => Does not work therefore using
1024
        string* ssq = new string[1000]; // Creating a string array with size
1024
        int count = subSq(s, ssq);      // Assuming 2^n is less than 1024
i.e. n <=10
        for (int i=0; i<count; i++){
                cout<<ssq[i]<<"\n";
        }
        cout<<"Total number of subsequences "<<count;
}
```

- Eg 28. Given an integer n, using phone keypad find out all the possible strings that can be made using digits of input n. Eg. 23 => 9 combinations



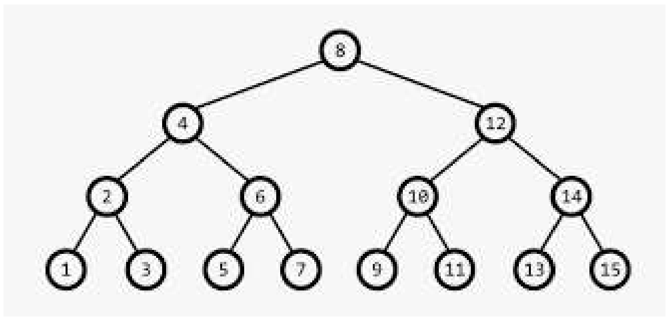
```
#include <string>
   using namespace std;

   int keypad(int num, string output[]){
           if (num == 0) {
           output[0] = "";
           return 1;
       }
       int rem = num%10;
       if (rem==1 || rem==0) return 0;
       int rec_Call = keypad(num/10, output);
       if (rec_Call){
           char char1, char2, char3, char4;
           if (rem!=7 && rem!=9){
                   if (rem==2){
                   char1='a';
                   char2='b';
```

```
                    char3='c';
                    }
                    else if (rem==3){
                    char1='d';
                    char2='e';
                    char3='f';
                    }
                    else if (rem==4){
                    char1='g';
                    char2='h';
                    char3='i';
                    }
                    else if (rem==5){
                    char1='j';
                    char2='k';
                    char3='l';
                    }
                    else if (rem==6){
                    char1='m';
                    char2='n';
                    char3='o';
                    }
                    else if (rem==8){
                    char1='t';
                    char2='u';
                    char3='v';
                    }
                    int size = 3*rec_Call;
                    int psize = rec_Call;
                    string* cp = new string[10000];
                    for (int i=0; i<rec_Call; i++){
                    cp[i]=output[i];
                    }
                    int j=0;
                    for (int i=0; i<size; i++){
                            if (i<psize) output[i] = cp[j] + char1;
                            else if (i>=psize && i<psize*2) output[i] = cp[j]
    + char2;

                            else if (i>=psize*2) output[i] = cp[j] + char3;
                            j++;
                            if (j==rec_Call) j=0;
```

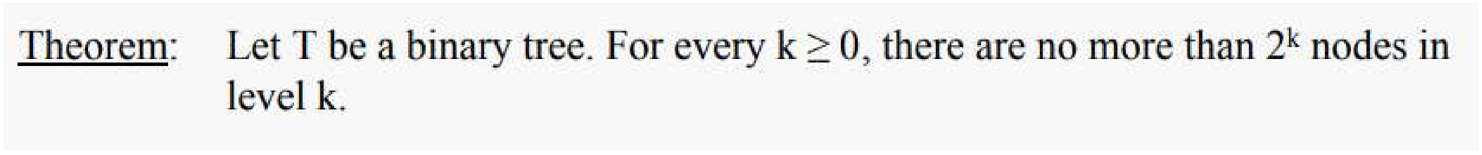
```
                }
                return 3*rec_Call;
        }
        if (rem==7 || rem==9){
                if (rem==7){
                char1='p';
                char2='q';
                char3='r';
                char4='s';
                }
                else if (rem==9){
                char1='w';
                char2='x';
                char3='y';
                char4='z';
                }
                int size = 4*rec_Call;
                int psize = rec_Call;
                string* cp2 = new string[10000];
                for (int i=0; i<rec_Call; i++){
                cp2[i]=output[i];
                }
                int j=0;
                for (int i=0; i<size; i++){
                                if (i<psize) output[i] = cp2[j] + char1;
                else if (i>=psize && i<psize*2) output[i] = cp2[j] +
char2;
                else if (i>=psize*2 && i<psize*3) output[i] = cp2[j] +
char3;
                else if (i>=psize*3) output[i] = cp2[j] + char4;
                j++;
                if (j==rec_Call) j=0;
                }
                return 4*rec_Call;
        }
    }
    else {
        output[0]="";
        return 1;
    }
}
```

- Eg 29. Print all the subsequences of a given string without the need to store it.

  Recursion works on a perfect binary tree or 2-tree.



A perfect binary tree (sometimes proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children.


Theorem: Let T be a binary tree. For every k ≥ 0, there are no more than 2^k nodes in level k.

```
#include<iostream>
using namespace std;

void print_Subseq(string input, string output){
    if (input.empty()){ // Base Case
        cout<<output<<"\n"; // Print the output if string is empty
        return;
    }
    print_Subseq(input.substr(1), output);          // Without the leftmost
character
    print_Subseq(input.substr(1), output+input[0]); // With the leftmost
character
}

int main(){
    string input;
    cin>>input;
    string output="";
    print_Subseq(input, output);
}
```

- Eg 30. Given an integer n, using phone keypad, only print all the possible strings that can be made using digits of input n. Eg. 23 => 9 combinations

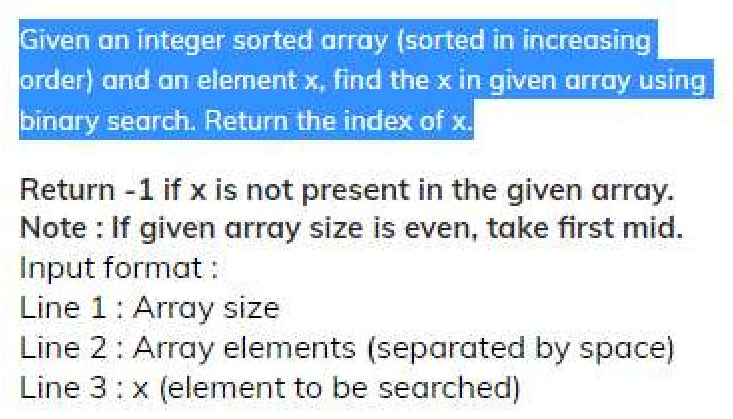
  Recursion works on a full tertiary-quaternary tree.

```
#include <iostream>
#include <string>
using namespace std;

int count=0;
void kp_new(int n, string s){
    if (n==0) {
        cout<<s<<"\n";
        count++;
        return;
    }
    int rem=n%10;               // Last digit stored
    char c1, c2, c3, c4;
    if (rem!=0 && rem!=1){ // 0 & 1 do not correspond to any characters
        if (rem==2){
            c1='a'; c2='b'; c3='c';
        }
        if (rem==3){
            c1='d'; c2='e'; c3='f';
        }
        if (rem==4){
            c1='g'; c2='h'; c3='i';
        }
        if (rem==5){
            c1='j'; c2='k'; c3='l';
        }
        if (rem==6){
            c1='m'; c2='n'; c3='o';
        }
        if (rem==8){
            c1='t'; c2='u'; c3='v';
        }
        if (rem==7){
            c1='p'; c2='q'; c3='r'; c4='s';
        }
        if (rem==9){
            c1='w'; c2='x'; c3='y'; c4='z';
        }
        if (rem!=7&&rem!=9){    // 2, 3, 4, 5, 6, 8 correspond to three
charcters
            kp_new(n/10, c1+s); // therefore the function is called thrice
            kp_new(n/10, c2+s); // note - n/10 is passed
            kp_new(n/10, c3+s);
        }
        if (rem==7||rem==9){    // 7 & 9 correspond to four characters
            kp_new(n/10, c1+s); // therefore the function is called four times
            kp_new(n/10, c2+s); // note - n/10 is passed
            kp_new(n/10, c3+s);
            kp_new(n/10, c4+s);
        }
    }
}

void printKeypad(int num){
    string s="";
    kp_new(num, s);             // just kp_new(num, "") also works
}

int main(){
    int num; cin >> num;
    printKeypad(num);
    cout<<"Total combinations = "<<count;
    return 0;
}
```

- Eg 31.

```
bool checkAB(char s[]) {
        if (s[0]=='\0') return false;
    if (s[0]=='a'){
        if(s[1]=='\0') {
            return true;
        }
        if (s[1]=='a'){
                bool rec_Call = checkAB(s+1);
                return rec_Call;
        }
        if (s[1]=='b'&&s[2]=='b'){
            bool rec_Call = checkAB(s+1);
            return rec_Call;
        }
    }
    if(s[0]=='b'&&s[1]=='b'){
        if(s[2]=='\0'){
            return true;
        }
        if (s[2]=='a'){
            bool rec_Call = checkAB(s+2);
            return rec_Call;
        }
    }
    bool rec_Call = checkAB(s+1);
    if((rec_Call)&&s[0]=='a') return true;
    else return false;
}
```

- Eg 32. - Binary Search on a given ascending sorted array


Given an integer sorted array (sorted in increasing order) and an element x, find the x in given array using binary search. Return the index of x.

Return -1 if x is not present in the given array.
Note : If given array size is even, take first mid.
Input format :
Line 1 : Array size
Line 2 : Array elements (separated by space)
Line 3 : x (element to be searched)

```
// input - input array
// size - length of input array
// element - value to be searched
int binarySearch(int a[], int size, int x) {
        if(size==1){
        if(a[0]==x) return 0;
        else return -1;
    }
    int mid=0;
    if (size%2==0) mid = size/2-1;
    else mid=size/2;
    if (a[mid]==x) return mid;
    else if (a[mid]>x){
```

```
    else if (a[mid]>x){
        int rec_Call = binarySearch(a, mid, x);
        if (rec_Call!=-1) return rec_Call;
        else  return -1;
    }
    else {
        int rec_Call=binarySearch(a+mid+1, size-mid-1, x);
        if (rec_Call!=-1) return rec_Call+mid+1;
        else return -1;
    }
}
```

- Eg 32. A child is running up a staircase with N steps, and can hop either 1 step, 2 steps or 3 steps at a time. Implement a method to count how many possible ways the child can run up to the stairs. You need to return number of possible ways W. **#IMP** Eg. 4 => 7; 5 => 13

```
int staircase(int n){
    if(n<0) return 0;
    if(n==0) return 1;
    int rec_Call = staircase(n-1)+staircase(n-2)+staircase(n-3);
    return rec_Call;
}
```

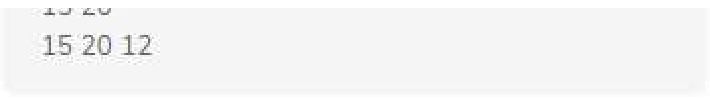
- Eg 33. Return subset of an array - Given an integer array (of length n), find and return all the subsets of input array. Subsets are of length varying from 0 to n, that contain elements of the array. But the order of elements should remain same as in the input array. Note : The order of subsets are not important.

Sample Input:

```
3
15 20 12
```

Sample Output:

```
[] (this just represents an empty array, don't worry
about the square brackets)
12
20
20 12
15
15 12
15 20
```



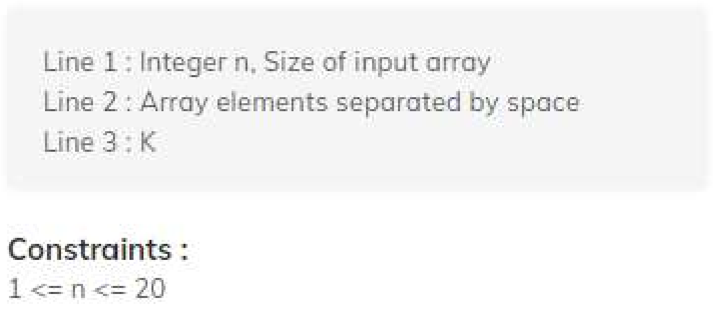
```
15 20
15 20 12
```

```
/***
You need to save all the subsets in the given 2D output array. And return the
number of subsets(i.e. number of rows filled in output) from the given
function.

In ith row of output array, 1st column contains length of the ith subset. And
from 1st column actual subset follows.
For eg. Input : {1, 2}, then output should contain
        {{0},           // Length of this subset is 0
        {1, 2},         // Length of this subset is 1
        {1, 1},         // Length of this subset is also 1
        {2, 1, 2}}      // Length of this subset is 2

Don't print the subsets, just save them in output. // Bottom Up approach
***/
#include <iostream>
using namespace std;

int subset(int a[], int n, int b[][20]) { // a - input array, n - size of the
array a, b - output 2D array
        if (n==0) {
        b[0][0]=0;                        // Base Case - output[0] i.e. first
row is empty
        return 1;
    }
    int rec=subset(a+1, n-1, b);
    for (int i=0; i<rec; i++){           // rec is current the size of the
output array
        int j=0;
        while (j<=b[i][0]){              // As the length of the current row
is stored in b[row][0]
            if (j==0) {
                b[i+rec][j+1] = a[0];    // Just add the first term
                b[i+rec][0]=1;           // i+rec is used to fill the new rows
            }
            else {
                b[i+rec][j+1] = b[i][j]; // Copy the terms of the previous
array
                b[i+rec][0]++;
            }
            j++;
        }
    }
    return 2*rec;                       // size of the output array doubles
and returns the size
}

int main() {
  int input[20],length, output[35000][20];
  cin >> length;
  for(int i=0; i < length; i++)
  cin >> input[i];
  int size = subset(input, length, output);
  for( int i = 0; i < size; i++) {
        for( int j = 1; j <= output[i][0]; j++) {
                cout << output[i][j] << " ";
    }
    cout << endl;
  }
}
```

- Eg. 34. Print subsets of an array. Given an integer array (of length n), find and print all the subsets of input array.

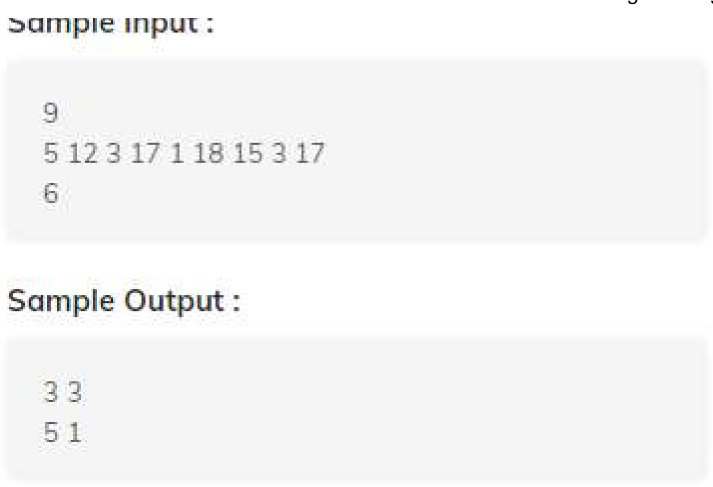
Easier method than returning (Top Down Approach) Like a full binary tree (with or without first element)

```
#include <iostream>
using namespace std;

void printSOA(int a[], int as, int b[], int bs){ // One input array and
another output array with their sizes as arguments
    if (as==0){
        for(int i=0; i<bs; i++){                // Base Cases are the leaf
nodes here
            cout<<b[i]<<" ";                    // i.e. when size of a == 0,
output is ready thus print it
        }
        cout<<"\n";
        return;
    }
    printSOA (a+1, as-1, b, bs);                // 1st branch = without
including the first element of the input array
    b[bs++]=a[0];                               // updating output with the
first element of the input array
    printSOA (a+1, as-1, b, bs);                // 2nd branch = including the
first element of the input array
}                                               // also first posn is shifted
by one

void printSubsetsOfArray(int input[], int size) { // Redundant Function -
need only two function
        int b[1000];
        int bs=0;                         // Initial size of the output array is
zero
        printSOA(input, size, b, bs);  // Calling the main function
}
int main() {
  int input[1000],length;
  cin >> length;
  for(int i=0; i < length; i++)  cin >> input[i];
  printSubsetsOfArray(input, length);
}
```

- Eg. 35. Return subsets to sum k. Given an array A of size n and an integer K, return all subsets of A which sum to K. **[IMPORTANT]**
  - Subsets are of length varying from 0 to n, that contain elements of the array. But the order of elements should remain same as in the input array.
  - Note : The order of subsets in the answer is not important.
  - Input format :
  - 

    Line 1 : Integer n, Size of input array
    Line 2 : Array elements separated by space
    Line 3 : K

    Constraints :
    1 <= n <= 20



Sample Input :

```
9
5 12 3 17 1 18 15 3 17
6
```

Sample Output :

```
3 3
5 1
```

```
/***
You need to save all the subsets in the given 2D output array. And return the
number of subsets(i.e. number of rows filled in output) from the given
function.

In ith row of output array, 1st column contains length of the ith subset. And
from 1st column actual subset follows.
For eg. Input : {1, 3, 4, 2} and K = 5, then output array should contain
        {{2, 1, 4},     // Length of this subset is 2
        {2, 3, 2}}      // Length of this subset is 2

Don’t print the subsets, just save them in output.  // Bottom Up approach
***/
#include <iostream>
using namespace std;

int subsetSumToK(int input[], int n, int output[][50], int k) {
    if (n==0){                 // 2 conditions at base case (n==0)
        if (k==0){             // if k==0 then the sum to k achieved, return 1
            output[0][0]=0;
            return 1;
        }
        else return 0;         // if k<0 or k>0 then sum to k not achieved,
return 0
    }
    int out1[10000][50];       // 1st branch - it includes the first element
of the input array
    int rec1 = subsetSumToK(input+1, n-1, out1, k-input[0]); // thus k-
input[0] is passed as next k
```

```
    for(int i=0; i<rec1; i++){
        for (int j=out1[i][0]; j>=0; j--){
            out1[i][j+1]=out1[i][j];         // need to shift all col from
2nd col of all rows by one
        }
        out1[i][1]=input[0];                 // 2nd column is given the value
of input[0];
        out1[i][0]++; //
    }
    int out2[10000][50];
    int rec2 = subsetSumToK(input+1, n-1, out2, k); // 2nd branch - it does
not include the first element of the input array
    int size = rec1+rec2;                           // total size is the sum
of the sizes of both out1 and out2 2D arrays
    for (int i=0; i<size; i++){                     // adding both output1
and output2 to give us the final output array
        if (i<rec1){
            for (int j=0; j<=out1[i][0]; j++){
                output[i][j]=out1[i][j];
            }
        }
        if (i>=rec1){
            for (int j=0; j<=out2[i-rec1][0]; j++){ // i-rec is used to
properly add elements after output1 has been added
                output[i][j]=out2[i-rec1][j];
                }
        }
    }
        return size;
}

int main() {
  int input[20],length, output[10000][50], k;
  cin >> length;
  for(int i=0; i < length; i++)
    cin >> input[i];

  cin >> k;

  int size = subsetSumToK(input, length, output, k);
  for( int i = 0; i < size; i++) {
        for( int j = 1; j <= output[i][0]; j++) {
                cout << output[i][j] << " ";
    }
    cout << endl;
  }
}
```

- Eg 36. Print subset sum to k. Given an array A and an integer K, print all subsets of A which sum to K.
  - Only print hence top down approach
  - Each level has two branches (with or without including the first element of the input array)
  - Shortcut used here - a. Using print all the subset of a given array (Eg. 34)

                              b. Simply modified the above program that only if
sum == k print output else dont print

```
#include <iostream>
using namespace std;

void printSStoK(int a[], int as, int k, int b[], int bs){
    if (as==0){
        int sum=0;
        for (int i=0; i<bs; i++) sum+=b[i];
        if (sum==k){
                for(int i=0; i<bs; i++){
                cout<<b[i]<<" ";
                }
                cout<<"\n";
        }
        return;
    }
    printSStoK(a+1, as-1, k, b, bs); // Same as for the Eg. 34
    b[bs++]=a[0];
    printSStoK (a+1, as-1, k, b, bs);
}

void printSubsetSumToK(int input[], int size, int k) { // Redundant function
    int b[1000];
    int bs=0;
```

```
        printSStoK(input, size, k, b, bs); // Main function as we have to
include an output array in function
}

int main() {
  int input[1000],length,k;
  cin >> length;
  for(int i=0; i < length; i++)
    cin >> input[i];
  cin>>k;
  printSubsetSumToK(input, length,k);
}
```

- `Eg. 37 - Return all codes - String.  (Return => Bottom up approach)`
  Assume that the value of a = 1, b = 2, c = 3, ... , z = 26. You are given a numeric string S. Write a program to return the list of all possible codes that can be generated from the given string.
  - Note : The order of codes are not important. And input string does not contain 0s.
  - Input format :

```
A numeric string
```

Constraints :
1 <= Length of String S <= 10

Sample Input:

```
1123
```

Sample Output:

```
aabc
kbc
alc
aaw
kw
```

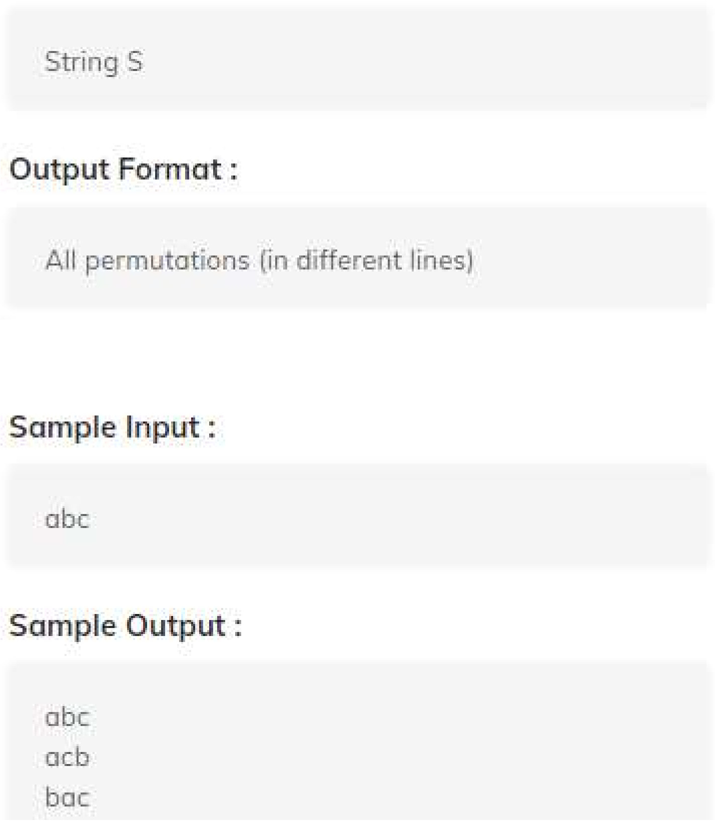
```
#include <iostream>
#include<string>
using namespace std;
```

```
int getCodes(string s, string output[10000]) {
    if (s.size()==0) {
        output[0]="";                               // Base case for bottom
up approach
        return 1;
    }
    int rec1=getCodes(s.substr(1), output);          // 1st branch - decoding
only the first element i.e. as a single digit
    char x1=s[0]-'0'+'a'-1;                          // Decoding the sigle
digit character to its corresg ASCII character
    for (int i=0; i<rec1; i++){                      // IMP - char digit to
ASCII char alphabet => char-'0' + 'a' - 1;
        output[i]=x1+output[i];                      // IMP - int digit to
ASCII char alphabet => int + 'a' -1;
    }
    if(s.size()>=2){
        int rec2=getCodes(s.substr(2), output+rec1);  // 2nd - decoding both
the first and second element i.e as a double digit
        int y, z;
        y=(s[0]-'0')*10;                              // Creating a double
digit integer
        z=y+(s[1]-'0');                               // Converting the
double digit integer to its corresg alphabet character
        if (z>=10&&z<=26){                            // making sure to only
convert 1 to 26 to their ASCII char equivalent
            char x2=z+'a'-1;
            for (int i=rec1; i<rec1+rec2; i++){       // Putting the results
of both the branches in the final result output
                output[i]=x2+output[i];
            }
            return rec1+rec2;                         // Returning the size
of the resultant output array of strings
        }
    }
    return rec1;                                      // If second branch was
did not give any result i.e. >26
}

int main(){
    string input;
    cin >> input;
    string output[10000];
    int count = getCodes(input, output);
    for(int i = 0; i < count && i < 10000; i++)
        cout << output[i] << endl;
    return 0;
}
```

- Eg. 38 - Print all codes - String. (Only Print - Top Down approach)
  Assume that the value of a = 1, b = 2, c = 3, ... , z = 26. You are given a numeric string S. Write a program to print the list of all possible codes that can be generated from the given string.
  - Note : The order of codes are not important. Just print them in different lines.
  - Input format : Same as Eg. 37.

```
#include <iostream>
#include<string>
using namespace std;

void print(string s, string output){
    if (s.size()==0) {
        cout<<output<<"\n";                    // Base case of top down
approach i.e. print the output
        return;
    }
    string output2=output;
    char x1=s[0]-'0'+'a'-1;                    // Converting a char single
digit to char ASCII char
        output=output+x1;
    print(s.substr(1), output);                // 1st branch - char single
digit recursive call, shifts the first pos by 1
    if (s.size()>=2){                          // Only if size is >= 2 i.e.
double digit recursive call requires atleast size=2
        int y, z;
        y=(s[0]-'0')*10;
        z=y+(s[1]-'0');                        // Converting a char double
digit to char ASCII char
        if (z>=10&&z<=26){
            char x2=z+'a'-1;
            output2=output2+x2;
                print(s.substr(2), output2);   // Shifts the first position
by 2 not 1 as char double digit already considered
        }
    }
}

void printAllPossibleCodes(string input) {     // Redundant function
        string output;
    output="";
    print(input, output);                      // Main function uses two
strings as an argument
}

int main(){
    string input;    cin >> input;
    printAllPossibleCodes(input);
    return 0;
}
```

- Eg. 39 - Return Permutations - String (Only Return - Bottom Up approach)
    - Given a string S, find and return all the possible permutations of the input string.
    - Note 1 : The order of permutations is not important.
    - Note 2 : If original string contains duplicate characters, permutations will also be duplicates.
    - Using an array of strings - string output[10000] to store the different permutations
    - Input Format :



```
String S
```

Output Format :

```
All permutations (in different lines)
```

Sample Input :

```
abc
```

Sample Output :

```
abc
acb
bac
```

```
bca
cab
cba
```

```
#include <iostream>
#include<string>
using namespace std;

//Method A - The character in the first position changes and appends at the
beginning to recursion for n-1 elements

int returnPermutations(string s, string output[]){
        if(s.size()==0){                                 // Base case of
bottom up approach
        output[0]="";
        return 1;
    }
        int rec_Call=0;
    for (int i=0; i<s.size(); i++){
        string recString = s.substr(0,i) + s.substr(i+1); // Important step
as it helps in creating different substrings of // the same size i.e. size-1
& changing the first element of the input string
        rec_Call = returnPermutations(recString, output); // Retrieve the
output of recursion on n-1 elements
        for (int j=0; j<rec_Call; j++){                   // Loop used as the
first element changes the same no. of times
            output[j] = s[i] + output[j];                 // as the size of
the input string
        }
        output = output+rec_Call;                         // The first posn
of the output array is shifted by rec_Call
    }
    return s.size()*rec_Call;                             // The size of
output thus becomes s.size() times rec_Call
}
int main(){
```