*static* *void* **Nto1**(*int* *n*){

        if(n==0)return;

        System.out.println(n);

        Nto1(n-1);

    }

*static* *void* **Nto1backtrack**(*int* *i*,*int* *n*){

        if(i>n)return;

        Nto1backtrack(i+1, n);

        System.out.println(i);

    }

*static* *void* **\_1toN** (*int* *n*){

        if(n==0)return;

        \_1toN(n-1);

        System.out.println(n);

    }

*static* *void* **NameNtime**(*String* *str*,*int* *n*){

        if(n==0)return;

        System.out.println(str);

        NameNtime(str, n-1);

    }

*static* *int* **sumofN(***int* *n*){

        if(n==1)return 1;

        return n + sumofN(n-1);

    }

*static* *int* **Factorial**(*int* *n*){

        if(n==0) return 1;

        return n \* Factorial(n-1);

    }

*static* *void* **reverseArray**(*int* *arr*[],*int* *l* ,*int* *r*){

        if(l>r)return;

*int* temp=arr[l];

        arr[l]=arr[r];

        arr[r]=temp;

        reverseArray(arr, l+1, r-1);

    }

*static* *boolean* **pallendrome**(*String* *str* , *int* *i*){

        if(i>=str.length()/2)return true;

        else if(str.charAt(i)!=str.charAt(str.length()-i-1))return false;

        return pallendrome(str, ++i);

    }

*static* *int* **fibo(***int* *n*){

        if(n==1|| n==0)return n;

        return fibo(n-2)+fibo(n-1);

    }

***//PowerDet appoach***

*static* *void* Powerset(*String* *str*){

*int* n =str.length();

        for(*int* i =0 ;i<=(1<<n)-1;i++){

*String* sub="";

            for(*int* j=0;j<n;j++){

                if((i&(1<<j))!=0){

                    sub += str.charAt(j);

                }

            }

            System.out.println(sub);

        }

    }

***//recursive approach (string input )***

***//1.printing all sub sequences***

*static* *void* PrintS(*int* *index* , *int* *arr*[], *List*<*Integer*> *result*,*List*<*List*<*Integer*>> *store*,*int* *n*){

        if(index==n){

            store.add(new *ArrayList*<>(result));

            return;

        }

        result.add(arr[index]);

        PrintS(index+1,arr,result,store,n);

        result.remove(result.size()-1);

        PrintS(index+1, arr, result, store,n);

    }

***//2.printing all sub sequneces wiht sum K***

*static* *void* PrintSumK(*int* *index* , *int* *currsum*,*int* *sum*,*int* *arr*[], *List*<*Integer*> *result*,*List*<*List*<*Integer*>> *store*,*int* *n*){

        if(index==n){

            if(currsum==sum)

            store.add(new *ArrayList*<>(result));

            return;

        }

        currsum+=arr[index];

        result.add(arr[index]);

        PrintSumK(index+1,currsum,sum,arr,result,store,n);

        result.remove(result.size()-1);

        currsum-=arr[index];

        PrintSumK(index+1, currsum,sum,arr, result, store,n);

    }

***//3.printing only 1 sub seuences with sum K***

*static* *boolean* PrintSumKOnly1(*int* *index* , *int* *currsum*,*int* *sum*,*int* *arr*[], *List*<*Integer*> *result*,*int* *n*){

        if(index==n){

            if(currsum==sum){

                for(*Integer* i : result){

                    System.out.print(i+" ");

                }

                return true;

            }

            else{

                return false;

            }

        }

        currsum+=arr[index];

        result.add(arr[index]);

        if(PrintSumKOnly1(index+1,currsum,sum,arr,result,n)==true)return true;

        result.remove(result.size()-1);

        currsum-=arr[index];

        if(PrintSumKOnly1(index+1, currsum,sum,arr, result,n)==true)return true;

        return false;

    }

***//4.counting all sub sequences wiht sum K***

*static* *int* PrintSumKCount(*int* *index* , *int* *currsum*,*int* *sum*,*int* *arr*[], *List*<*Integer*> *result*,*int* *n*){

        if(index==n){

            if(currsum==sum){

                return 1;

            }

            else{

                return 0;

            }

        }

        currsum+=arr[index];

        result.add(arr[index]);

*int* l = PrintSumKCount(index+1,currsum,sum,arr,result,n);

        result.remove(result.size()-1);

        currsum-=arr[index];

*int* r= PrintSumKCount(index+1, currsum,sum,arr, result,n);

        return l+r;

    }

***//all posible subset with distinct elements input***

*static* *void* subset1(*int* *index*,*int* *arr*[],*List*<*Integer*> *result*,*List*<*List*<*Integer*>>*store*,*int* *n*){

        if(index==n){

            store.add(new *ArrayList*<>(result));

            return;

        }

        result.add(arr[index]);

        subset1(index+1, arr, result, store, n);

        result.remove(result.size()-1);

        subset1(index+1, arr, result, store, n);

    }

***//all possible subset with duplicates in input***

*static* *void* subset2(*int* *index*,*int* *arr*[],*List*<*Integer*> *result*,*List*<*List*<*Integer*>> *store*,*int* *n*){

        store.add(new *ArrayList*<>(result));

        for(*int* i =index ; i <n ;i++){

            if(i>index && arr[i]==arr[i-1])continue;

            result.add(arr[i]);

            subset2(i+1, arr, result, store, n);

            result.remove(result.size()-1);

        }

    }

***// the differennce is basically in the input tbh duplicate inputs use the for loop else use the pick not pick approach ..........for sub sequene1 where we can repeat values dont move index forward thats it***

***//Permutation question 1 -> array + ds to store***

*static* *void* perm1(*int* *index*,*List*<*Boolean*> *visited*,*int* *arr*[],*List*<*Integer*> *result*,*List*<*List*<*Integer*>>*store*,*int* *n*){

        if(n==index){

            store.add(new *ArrayList*<>(result));

            return;

        }

        for(*int* i = 0 ;i < n;i++){

            if(visited.get(i)==false){

                visited.set(i, true);

                result.add(arr[i]);

                perm1(index+1,visited,arr,result,store,n);

                result.remove(result.size()-1);

                visited.set(i, false);

            }

        }

    }

***//swap approach space optimised***

*static* *void* swap(*int* *i* , *int* *j* ,*int* *arr*[]){

*int* temp = arr[i];

        arr[i]=arr[j];

        arr[j]=temp;

    }

*static* *void* perm2(*int* *index*,*int* *arr*[],*List*<*List*<*Integer*>> *store*,*int* *n*){

        if(index==n){

*List*<*Integer*> permutation = new *ArrayList*<>();

            for(*int* i : arr){

                permutation.add(i);

            }

            store.add(new *ArrayList*<>(permutation));

            return;

        }

        for(*int* i= index;i<n;i++){

            swap(index,i,arr);

            perm2(index+1,arr,store,n);

            swap(index, i, arr);

        }

    }

**//NQUEEN**

*static* *Boolean* isSafe(*int* *col* ,*int* *row* , *char* *arr*[][],*int* *n*){

*int* coltemp =col;

*int* rowtemp = row;

        while(col>=0&&row>=0){

            if(arr[row][col]=='Q')return false;

            col--;

            row--;

        }

        col=coltemp;

        row=rowtemp;

        while(col>=0){

            if(arr[row][col]=='Q')return false;

            col--;

        }

        col=coltemp;

        row=rowtemp;

        while(col>=0&&row<n){

            if(arr[row][col]=='Q')return false;

            col--;

            row++;;

        }

        return true;

    }

*static* *void* **Nqueens**(*int* *column*, *int* *n* ,*char* *arr*[][],*List*<*List*<*String*>> *boards*){

            if(column==n){

*List*<*String*> board = new *ArrayList*<>();

                for(*int* i=0 ;i<n;i++){

                    board.add(new String(arr[i]));

                }

                boards.add(new *ArrayList*<>(board));

                return;

            }

            for(*int* i =0 ;i<n ;i++){

                if(isSafe(column,i,arr,n)){

                    arr[i][column]='Q';

                    Nqueens(column+1,n,arr,boards);

                    arr[i][column]='0';

                }

            }

    }

***//Optimised code left daigonal upper diagonal(n-1 col-row) ,lower diagonal(col + row) while fillingin value matain this hash and make entries according to check if possible condition we can check if its 0 for give row and column***

**//Counting inversion in Array (meger sort approach)**

*public* *class* inversionsinarray {

*static* *int* merge(*int* *arr*[],*int* *low* ,*int* *mid*,*int* *high*){

*int* i =low ;

*int* j= mid+1;

*List*<*Integer*> temp = new *ArrayList*<>();

*int* count=0;

        while(i<=mid && j<=high){

            if(arr[i]<=arr[j]){

                temp.add(arr[i]);

                i++;

            }

            else{

                temp.add(arr[j]);

**count+=mid-i+1;**

                j++;

            }

        }

        while(i<=mid){

            temp.add(arr[i]);

            i++;

        }

        while(j<=high){

            temp.add(arr[j]);

            j++;

        }

        for(*int* k = 0;k<temp.size();k++){

            arr[low+k]=temp.get(k);

        }

        return count;

    }

*static* *int* mergerSort(*int* *arr*[],*int* *low* ,*int* *high*){

*int* mid;

*int* count =0;

        if(low>=high)return count;

        mid=(low+high)/2;

        count += mergerSort(arr, low, mid);

        count += mergerSort(arr, mid+1, high);

        count += merge(arr,low,mid,high);

        return **count;**

    }

*public* *static* *void* main(*String*[] *args*) {

*int* arr[]={4,3,2,1};

        System.out.println(mergerSort(arr,0 , 3));

    }

}

**Suddoku solver**

public class suddokusolver {

static boolean isSafe(int row,int col ,char value,List<List<Character>> board){

for(int i=0;i<9;i++){

if((board.get(row).get(i)==value)||

(board.get(i).get(col)==value)||

board.get((3\*(row/3) + i/3)).get(3\*(col/3)+i%3)==value)

return false;

}

return true;

}

static boolean solve(List<List<Character>>board){

for(int i =0 ;i<9;i++){

for(int j=0 ;j<9;j++){

if(board.get(i).get(j)=='.'){

for(char c = '1'; c <= '9'; c++){

if(isSafe(i,j,c,board)){

board.get(i).set(j,c);

if(solve(board)==true)return true;

board.get(i).set(j,'.');

}

}

return false;

}

}

}

return true;

}

static void solveSuddoku(List<List<Character>> board){

if(solve(board)==true){

for(List<Character> row : board){

System.out.println(row);

}

return;

}

System.out.println("No valid input");

}

**Kth permutation**

*public* *class* kthpermutation {

*static* *void* kthperm(*String* *str*,*int* *n*,*int* *k*){

*int* blocksize=1;

*List*<*Character*> chars = new *ArrayList*<>();

       for(*int* i = 1; i <= n; i++) {

            blocksize = blocksize \* i;

            chars.add(str.charAt(i - 1));  ***// now includes all n characters***

        }

       blocksize = blocksize / n;

*//intial blocksize acchieved (n-1)!*

       k=k-1 **; *//zero based index for kth permutation***

*String* ans="";

       while(true){

*int* blocknumber =k/blocksize;

        ans+=chars.get(blocknumber);

        chars.remove(blocknumber);

        if(chars.size()==0){

            System.out.println(ans);

            return;

        }

***//new k vlaue inside the block***

        k=k%blocksize;

***//size of new blocks***

        blocksize= blocksize/chars.size();

       }

    }

*public* *static* *void* main(*String*[] *args*) {

*String* str ="1234";

        kthperm(str,str.length(), 17);

    }

}