```
Class Node { Node next, down; int data; }

Class LinkedListApp {

    Public Node newNode(int i) {
        Node a = new Node();
        a.data = i;
        a.next = null;
        a.down = null;
        return a;
    }

    Node Prev;

    Public Node flattenEasy(Node node) {
        if(node == null)
            return null;
        prev = node;
        Node next = node.next;

        if(node.down != null)
            node.next = flattenEasy(node.down);

        if(next != null)
            Prev.next = flattenEasy(next);

        return node;

    }

}
```

```
Public class flattendepthwise {
    public static void main(String args[]) {

        LinkedListAPP a = new LinkedListAPP();

        Node head = a.newNode(1);
        head.next = a.newNode(2);
        head.next.next = a.newNode(3);

        head.next.down = a.newNode(7);
        head.next.down.down = a.newNode(11);
        head.next.down.down.down = a.newNode(13);

        head.next.down.next = a.newNode(8);
        head.next.down.next.down = a.newNode(12);

        a.flattenEasy(head);
    }
}
```

Stack

Next Greater Element

arr[] = [1 3 2 4] o/p 3 4 4 -1

```
Public static long[] nextLargerElement(long[] arr, int n){

    Stack<Integer> stack = new Stack<Integer>();
    long res[] = new long[n];
    for(int x=0; x< arr.length; x++) {
        if(stack.isEmpty())
            stack.push(x);
        else if(arr[x] <= arr[stack.peek()])
            stack.push(x);
        else if(arr[x] > arr[stack.peek()]) {
            while(!stack.isEmpty() && arr[x]>arr[stack.peek()]
                res[stack.peek()] = arr[x];
                stack.pop();
            }
            stack.push(x);
        }
    }
    for(Iterator<Integer> iterator = stack.iterator(); iterator.hasNext
        ();) {
            Integer index = (Integer) iterator.next();
            res[index] = -1;
    }

    return res;

}
```

Important Question on stack

1. Nearest Greater to left.
2. Nearest Greater to Right.
3. Nearest Smaller to left.
4. Nearest Smaller to Right.

5. Stock Span problem.
6. Maximum Area of Histogram.
7. Maximum Area Rectangle in binary matrix

8 Rain water trapping
9 Implementing a min stack. → Extra space / → Extra space
10 Implementing stack using heap
11 The celebrity problem.
12 longest valid Parenthesis.
13 Iterative TOH

---

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

for
- j → 0 to i j++
- j → i to 0 j--
- j → i to n j++
- j → n to i j--

100% → Stack

Second loop → J loop.
J loop if dependent on i
↓↓
Stack

Stock Spain Problem

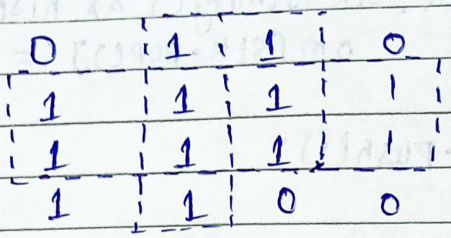
```
Public static int[] calculateSpan(int arr[], int n) {

    Stack<int[]> stack = new Stack<>();
    int result[] = new int[arr.length];

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

        if(stack.isEmpty())
            result[i] = -1;
        else if(stack.peek()[0] > arr[i])
            result[i] = stack.peek()[1];

        else {
            while(!stack.isEmpty() && stack.peek()[0] <=arr[i]){
                stack.pop();

            if(stack.isEmpty())
                result[i] = -1;
            else
                result[i] = stack.peek()[1];

        }
        stack.push(new int[]{arr[i], i});
    }
    for(int i = 0; i< n; i++)
        result[i] = i - result[i];

    return result;

}
```

// Maximum Area of Histogram

```
public static long[] leftmin(long hist[], long n) {

    long lm[] = new long[(int) n];
    Stack<Integer> stk = new Stack<>();
    lm[0] = -1;
    stk.push(0);
    for(int i = 1; i < n; i++) {

        long el = hist[i];
        while(!stk.isEmpty() && hist[stk.peek()] >= el)
            stk.pop();

        if(stk.isEmpty())
            lm[i] = -1;
        else
            lm[i] = stk.peek();

        stk.push(i);
    }

    return lm;
}


Public static long[] rightmin(long hist[], long n) {

    long rm[] = new long[(int) n];
    Stack<Integer> stk = new Stack<>();

    for(int i = 0; i < n; i++) {
        long el = hist[i];
```

```
        while(!stk.isEmpty() && hist[stk.peek()] > el)
            rm[stk.pop()] = i;

        stk.push(i);
    }
    while(!stk.isEmpty())
        rm[stk.pop()] = n;

    return rm;
}

public static long getMaxArea(long hist[], long n){

    long maxArea = 0;

    long lm[] = leftmin(hist, n);
    long rm[] = rightmin(hist, n);

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

        long el = hist[i];
        long w = rm[i] - lm[i] - 1;

        long area = el * w;

        if(area > maxArea)
            maxArea = area;
    }
    return maxArea;

}
```

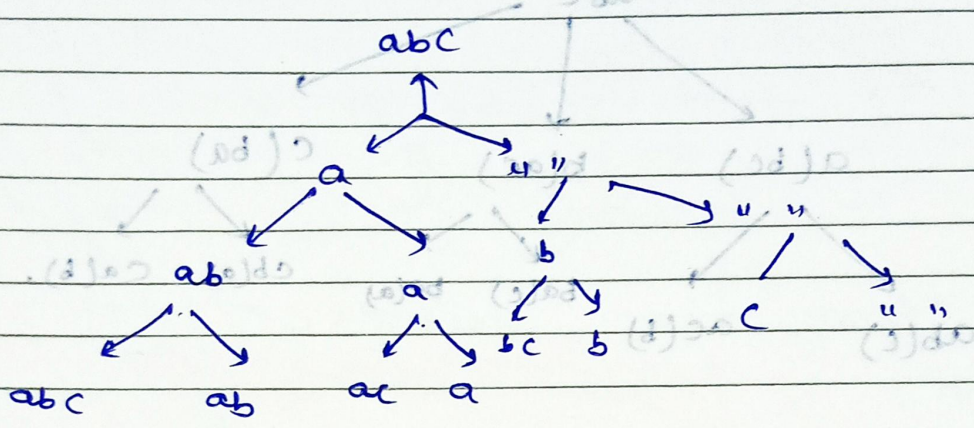
Maximum Area Rectangle in binary matrix.



Ans 8.

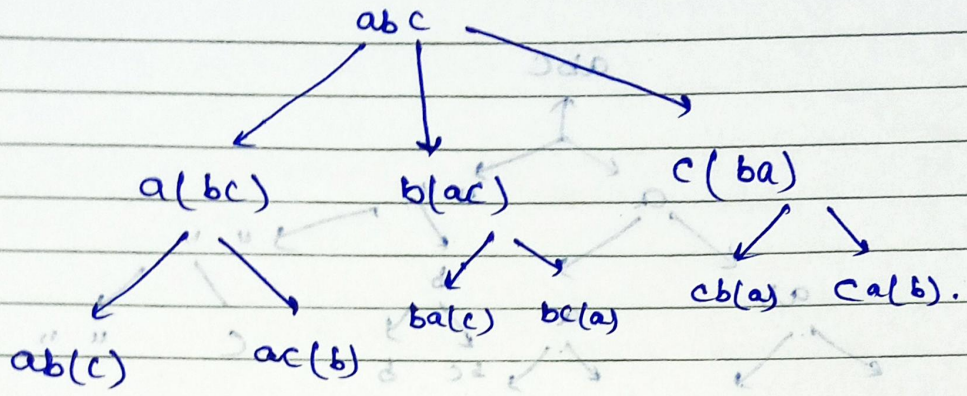
Array Recursion

Powerset of String) abc = { "", a, b, c, ab, bc, ac, abc }




```
void powerset(String s, int i, String cur) {
    if (i == s.length()) {
        print(cur);
        return;
    }
    powerset(s, i+1, cur + s[i]);
    powerset(s, i+1, cur);
}
```

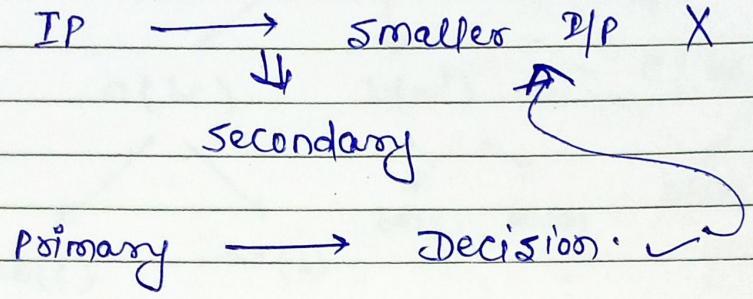
Print all permutation of string



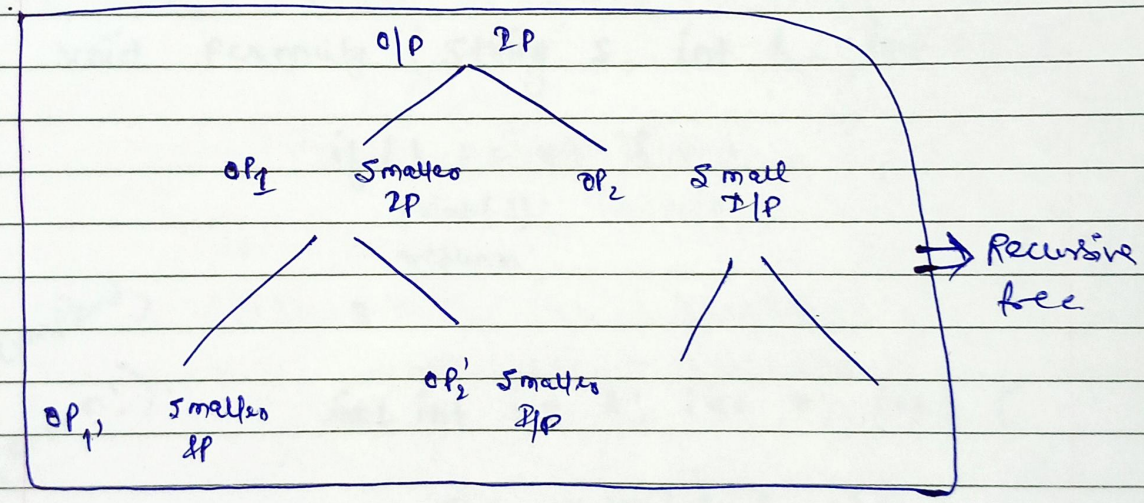

```
void Permute (String s, int l, int r) {
    if (l == r) {
        Print(s);
        return;
    }
    for (int i = l; i <= r; i++) {
        s = swap(s, l, i);
        Permute(s, l+1, r);
        s = swap(s, l, i);
    }
}
```

Time complexity
$O(n * n!)$.

Recursion ⟶ Choices + Decisions.

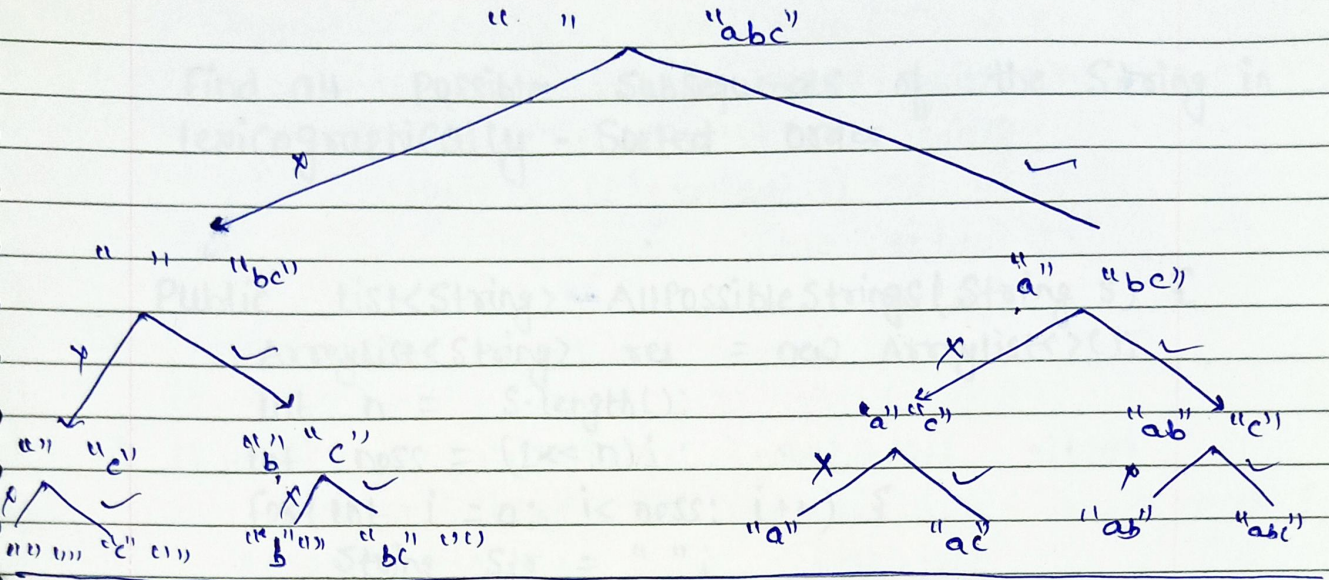


O/P ← " "



# of choices
⇓
# of Branches

Decision
↓
Smaller I/P



# of operation <= # of second (default 1sec.) * 10^8

Constraint (max).	Time Complexity.
10^{18}	logN
10^8	O(N)
10^4	$O(N^2)$
10^6	N.logN ($6 * 10^6 < 10^8$)
500	$O(N^3)$
85 - 90	$O(N^4)$
15 — 20	$O(2^N)$ — recursion or $(O(2^N * N^2))$
10 — 11	O(N!), $O(n^6)$

Recursion.

Find all possible subsequences of the String in lexicographically - Sorted order.

```
Public List<String> AllPossibleStrings(String s) {
    ArrayList<String> res = new ArrayList<>();
    int n = s.length();
    int noss = (1<<n);
    for(int i = 0; i< noss; i++) {
        String str = "";
        for(int j = 0; j< n; j++)
            if(checkbit(i, j) == 1)
                str += s.charAt(j);
        if(str.equals(""))
            {}
        else
            res.add(str);
    }
    Collections.sort(res);
    return res;
}
Public int checkbit(int n, int i) {
    return (n>> i) &1;
}
```

OR

```
Public List<String> S2 = new ArrayList<>();
Public void fun(String ip, String op) {
    if(ip.length() == 0) {
        if(op == "") return; S2.add(op); return;
    }
```

```
String op1 = op;
String op2 = op + ip.charAt(0);
ip = ip.substring(1);
fun(ip, op1);
fun(ip, op2);
}
```

```
Public List<String> AllPossibleStrings(String s) {

    List<String> a = new ArrayList<>();
    fun(s, "");
    a = s2;
    Collections.sort(a);
    return a;
}
```

Permutation with Spaces.

i/p ABC o/p A,B C, A BC AB C ABC

```
Public void spaces(String s, String str, int idx,
        ArrayList<String> a) {

    if(idx == s.length()) {
        a.add(str);
        return;
    }
    char currChar = s.charAt(idx);
    if(idx != s.length()-1)
        space(s, str + currChar + "", idx+1, a);
    space(s, str + currChar, idx+1, a);
}
```

```
Arraylist<String> permutation(String s) {
    ArrayList<String> ans = new ArrayList<>();
    space(S, "", 0, ans);
    return ans;
}
```

784 Letter case permutation.

ip a1b2 o/p a1B2, a1b2, A1b2, A1B2

```
void recurse(Char[] str, int POS, List<String> result) {

    if(pos == str.length) {
        result.add(new String(str)); return }
    if(Character.isLetter(str[POS])) {
        if(Character.isUPPERCase(str[Pos])) {
            str[POS] = Character.toLowerCase(str[Pos]);
            recurse(Str, pos+1, result);
            Str[Pos] = Character.toUppercase(Str[Pos]);
        }
        else {
            Str[pos] = Character.toUPPERCase(str[Pos]);
            recurse(str, pos+1, result);
            Str[Pos] = Character.toLowerCase(Str[Pos]);
        }
    }
    recurse(Str, Pos+1, result);
}
Public List<String> letterCasePermutation(String s) {
    List<String> result = new Arraylist<>();
    recurse(S.toCharArray(), 0, result);
    return result;
}
```

* Print N-bit binary numbers having more 1s than 0s

i/p 2 o/p 11 10

```
void Solve(int one, int zero, int N, String OP, ArrayList<String> res){
    if(N == 0) {
        res.add(OP);
        return;
    }
    if(one == zero) {
        String OP1 = OP + "1";
        Solve(one+1, zero, N-1, OP1, res);
    }
    else {
        String OP1 = OP + "1";
        Solve(one+1, zero, N-1, OP1, res);
        String OP2 = OP + "0";
        Solve(one, zero+1, N-1, OP2, res);
    }
}
ArrayList<String> NBitBinary(int N) {
    if(N == 0)
        return null;
    ArrayList<String> res = new ArrayList<>();
    String OP = "1";
    N -= 1;
    int one = 1, zero = 0;
    Solve(one, zero, N, OP, res);
    return res;
}
```

// Tower of Hanoi.

```
Static int count = 0;
Static ArrayList<Integer> arr = new ArrayList<>();

static void hanoi(int N, int s, int t, int d, int n) {
    if(N == 0)
        return;
    hanoi(N-1, s, d, t, n);
    count++;
    if(n == count) {
        arr.add(s);
        arr.add(d);
    }
    hanoi(N-1, t, s, d, n);
}

static List<Integer> shiftPile(int N, int n) {
    count = 0;
    arr.clear();
    hanoi(N, 1, 2, 3, n);
    return arr;
}
```

Important Question.

8) Print all subsequences of a string

Ans) Print all Substring

```
Static void substrings(int start, int end) {
    if(start == in.length() && end == in.length())
        return;
    else {
        if(end == in.length() + 1)
            substrings(start+1, start+1);
        else {
            s.o.p(in.substrings(start, end));
            substrings(start, end+1);
        }
    }
}
main() {
    substring(0, 1);
}
```

Sliding window problem.

fixed.

1) max / min sum sub array of size k
2) 1st -ve in every window size of k.
3) count occurrence of anagram.
4) max of all sub array of size k.
5) max of min for every window size

variable (window size variable)

1) largest / smallest sub array of sum k.
2) Longest substring with k unique character.
3)
4) pick toy
5) minimum window substring

// Max Sum Subarray of size K.

```
static int maximumSumSubarray(int K, ArrayList<Integer>
                Arr, int N) {

    int i=0, j=0, sum=0, max = min_value;

    while(j< N) {
        sum = sum + Arr.get(j);
        if(j-i+1< K)
            j++;
        else if(j-i+1 == K) {
            max = Math.max(max, sum);
            i++;
            j++;
            sum -= Arr.get(i-1);
        }
    }
    return max;
}
```

// 1st -ve in every window size of K.

```
Public long[] printFirstNegativeInteger(long arr[], int n, int K) {

    long arr[] = new long[n-k+1];
    int count = 0;
    LinkedList<Long> queue = new LinkedList<>();
    int i=0, j=0;
```

```
while(j < arr.length) {

    if(arr[j] < 0)
        queue.add(arr[j]);

    if(j-i+1 < K)
        j++;
    else if(j-i+1 == K) {

        if(!queue.isEmpty()) {
            arr[count] = queue.peek();
            count++;
            if(arr[i] == queue.peek())
                queue.remove();
        }
        else {
            arr[count] = 0;
            count++;
        }
        i++;
        j++;
    }
}
return arr;
}

// Count Occurrence of anagram.

int search(String pat, String txt) {

    HashMap<Character, Integer> hP = new HashMap<>();
    for(int i = 0; i<pat.length(); i++)
        hP.put(pat.charAt(i), hP.getOrDefault(pat.charAt(i), 0) + 1);
```

```
int count = hp.size(), i=0, j=0, res=0;

while(j< txt.length()) {
    if(hp.containsKey(txt.charAt(j))) {
        hp.put(txt.charAt(j), hp.get(txt.charAt(j))-1);
        if(hp.get(txt.charAt(j)) == 0)
            count--;
    }
    if(j-i+1 == pat.length()) {
        if(count==0)
            res++;
        if(hp.containsKey(txt.charAt(i))) {
            hp.put(txt.charAt(i), hp.get(txt.charAt(i))+1);
            if(hp.get(txt.charAt(i)) == 1)
                count++;
        }
        i++;
    }
    j++;
}
return res;
}

// Max of all sub array of size k.

Public int[] maxSlidingWindow(int [] a, int k) {
    if( a == null || k <= 0)
        return new int[0];
    int n = a.length, ri=0, r[] = new int[n-k+1];
    Deque<Integer> q = new ArrayDeque<>();
```

```
for(int i =0; i< a.length; i++ ) {
        while( !q.isEmpty() && q.peek()< i-K+1)
            q.poll();
        while( !q.isEmpty() && a[q.peekLast()] < a[i])
            q.pollLast();
        q.offer(i);
        if( i >= K-1)
            r[ri++] = a[q.peek()];
    }
    return r;
}
```

// Largest Subarray Of Sum K.

```
Public Static int lenOfLongSubarr(int A[], int N, int k){

    HashMap<Integer, Integer> hm = new HashMap<>();
    int ans = 0, Sum =0;
    hm.put(0, -1);
    for(int i =0; i< N; i++) {
        Sum += A[i];
        if( !hm.containsKey(sum))
            hm.Put(Sum, i);
        if( hm.containsKey( sum - K))
            ans = Math.max(ans, i - hm.get(Sum-K));
    }
    return ans;

}
```

// Longest Substring with K unique Character.

```
Public int longestKSubstr(String s, int K) {
  int ans = -1, i = -1, j = -1;
  HashMap<Character, Integer> map = new HashMap<>();
  While(true) {
    boolean f1 = false, f2 = false;
    While(i < s.length() - 1) {
      i++;
      f1 = true;
      Char ch = s.charAt(i);
      map.put(s.charAt(i), map.getOrDefault(s.charAt(i), 0) + 1);
      if(map.size() < K)
          continue;
      else if(map.size() == K) {
          int len = i-j;
          ans = Math.max(len, ans);
    }
    else
          break;
  }
  While(j<i) {
    j++; f2 = true; Char temp = s.charAt(j);
    if(map.get(temp) == 1)
          map.remove(temp);
    else
          map.put(temp, map.get(temp) - 1);
    if(map.size() > K)
          continue;
    else if(map.Size() == K) {
      int len = i-j; ans = Math.max(len, ans); break;
    }
  } if(f1 == false && f2 == false) break;
} return ans;
}
```

Quick Sort On Linked List

```
Public Static Node quickSort(Node node) {
    int size = 0;
    Node temp = node;
    while(temp != null) {
        size++;
        temp = temp.next;
    }
    quickSort(node, size);
    return node;
}

Public static void quickSort(Node node, int size) {

    if(size == 0 || size == 1)
        return;
    int len = 0, small = 0;
    Node least = node.next, high = node.next, prev =
        least;
    while(len != size - 1) {
        if(high.data < node.data) {
            int k = least.data;
            least.data = high.data;
            high.data = k;
            prev = least;
            least = least.next;
            small++;
        }
        high = high.next;
        len++;
    }
```

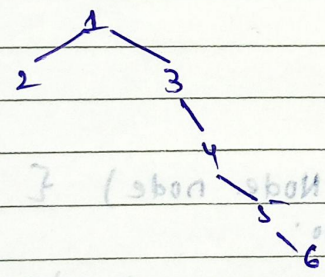
```
if(small != 0) {
        int k = prev.data;
        prev.data = node.data;
        node.data = k;
}
quicksort(node, small);

if(small != 0)
        quicksort(prev.next, size-small-1);
else
        quicksort(node.next, size-small-1);

}
```

Qu). Geek has developed an effective vaccine for Corona virus and he wants each of the N houses in Geekland to have access to it. Given a binary tree where each node represents a house in Geek Land. find the minimum no. of houses that should be Suppli -ed with the vaccine kit if one vaccine kit is sufficient for that house, its parent house and it's immediate child node.

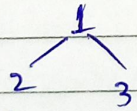
```
Static int ans = 0;

Public Static int SupplyVaccine (Node root) {
    int result = Post(root);
    if(result == 0)
        return ans+1;
    return ans;
}

public static int post(Node node) {
    boolean zero, two;
    if(node.left == null && node.right == null)
        return 0;

    if(node.left != null) {
        int get = Post(node.left);
        if(get==0)
            zero=true;
        if(get==2)
            two=true;
    }
    if(node.right != null) {
        int get = Post(node.right);
        if(get==0) zero=true;
```

```
if(get == 2)
        two = true;
}
if( zero == true) {
    ans ++;
    return 2;
}

if(two) {
    return 1;
}
return 0;
}
```

ex



O/P = 2



O/P 1

Q. You are given arrays A and B of length N each. Determine the number of good pairs.

A pair (i, j) $(1 \le i, j \le N)$ is said to be good if all of the following conditions are satisfied.

- $i < j$
- $A_i = B_j$ and $A_j = B_i$

```
Map<String, Integer> m = new HashMap<>();
long count = 0;

for(int i = 0; i < N; i++) {
    count += m.getOrDefault(B[i]+"."+a[i], 0);
    m.put(a[i]+"."+B[i], m.getOrDefault(a[i]+"."+B[i]
        ,0)+1);

}

return count;
```

Chef has a set S containing N distinct integers

Chef wants to gift Chefina an array A of any finite length such that the following conditions hold true:

- $A_i \in S \; \forall i$ In other words, each element of the array A should belong to the set S.
- Mean value of all the elements in A is exactly X.

Find whether there exists an array A of finite length satisfying the above conditions.

```
int max = A[0], min = A[0];

for(int i=0; i<N; i++) {
    if(A[i] > max)
        max = A[i];

    if(A[i] < min)
        min = A[i];
}

if(a > max || a < min)
    S.O.P("No");
else
    S.O.P("Yes");
```