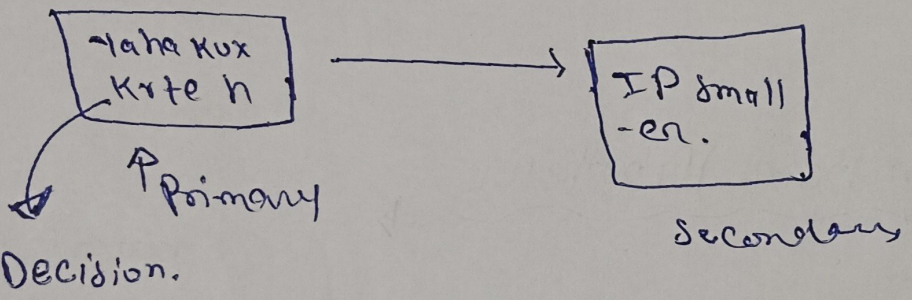
Recursion.

flow.

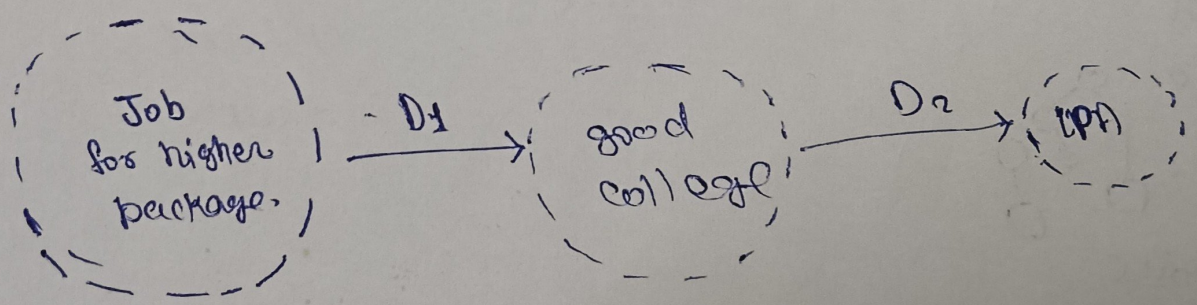
1.) make I/P smaller ? But Why
2.) Recursion - Decision tree
3.) Recursion - soul of Recursion
4.) 2 steps to solve any Recursion Problem

1.) make I/P smaller ? But Why

Recursion में I/P small करते rhi n ho jate.

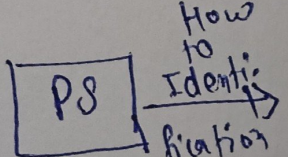


example:



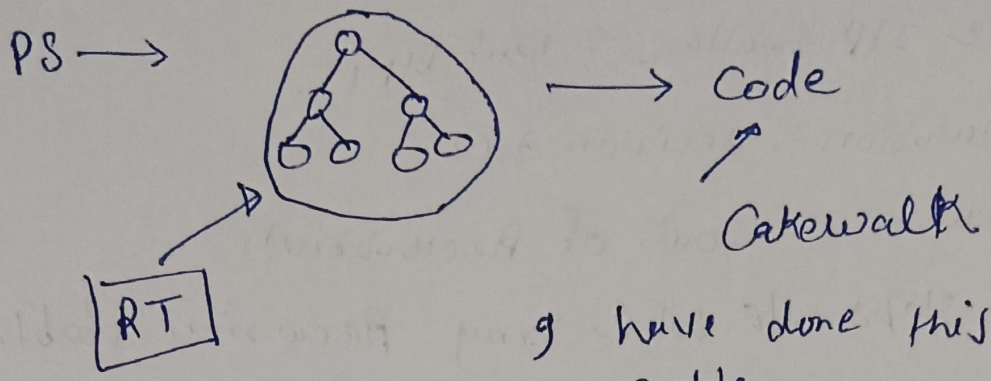
2) Recursion - Decision space

identification



↳ Choice + Decision

3.) Recursive tree



I have done this problem.

example.

subset problem

'abc' → "" a ab abc
b bc
c ca

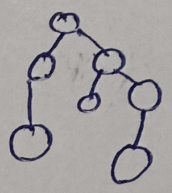
IBH

IBH : Induction Base Hypothesis.

Recursion.

1.) IBH

2.) Tree



3.) choice

Beauty of Hypothesis and Induction.
print n to 1.

```
void Print(int n)
{  if (n==1)
   {  cout<<n<<" ";
      return;
   }
```

```
    Print(n-1);
    cout << n << " ";
  }
```

Print(n-1);

now print 1 to n;

```
void print(int n) {
    int
    if (n==1) {
        cout << n << " ";
        return;
    }
    print(n-1);
    cout << n << " "; } induction Base | step;
}
int main() {
    int n; cin>>n;
    print(n);
    return 0;
}
```

Base condition

Sort an array.

(Recursion)

decision

decision I — Smaller II

I/P arr = 2 3 7 6 4 5 9

O/P sort = 2 3 4 5 6 7 9

sort

merge sort } nlogn.

Smaller IP

BC

Hypo

Induction

Hypo

```
Sortfun(arr)
  {
  }
```

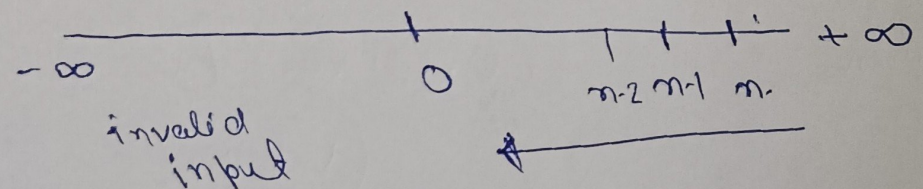
Sortfun(2 3 7 6 4 5 2) → 2 3 4 5 6 7 9

Sortfun(2 3 7 6 4 5) → 2 3 4 5 6,7

Base Condition

reduce

I/P → array(size of array)

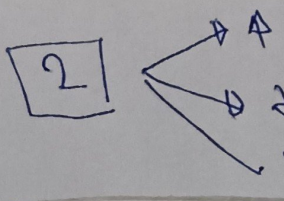


Sortfun(2,5,4) → 2 4 5

Sortfun(2,5) → 2 5

Sortfun(2) → 2

but



so, BC => `if (v.size() == 1) return`

(i) BC

(ii) Hypothesis (काम क्या करना है)

(iii) Induction (काम कैसे करना है)

Induction Step

sortfun(1 5 0 2) → 0 1 2 5

sortfun(1 5 0) → 0 1 5 ; 2

put in suitable place.

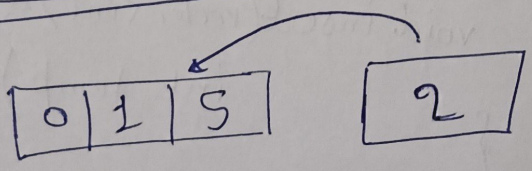
0 1 5 ; 2

↓

0 1 2 5

this do by loop / for loop. but

Recursion.



Decision

smaller I/P

2 in 0 1 5

2 in 0 1 x

Recursion

BC

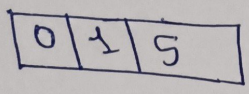
Hypo

Induction.

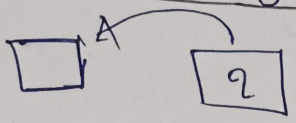
BaseCondition.

→ insert(vector, temp)

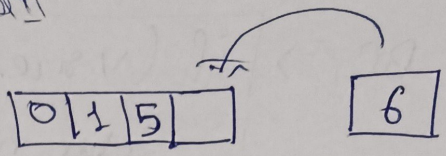
Cond I.



if (size of array == 0)



Cond II



यहाँ no need to compare because largest element

v[v.size-1] <= temp

overall BaseCondition.

if(v.size() == 0 || v[v.size()-1] <= temp)

v.push_back(temp)

Code

```
void sort(vector<int> &v)
{
    if (v.size() == 1)
    {
        return;
    }
    int temp = v[v.size()-1]
    v.pop_back();
    sort(v)
    insert(v, temp)
}
```

```
int main()
{
    ...
    ...
    sort(v)
}
```

```
void insert(vector<int> &v,
            int temp)
{
    if(v.size() == 0 || v[v.size()-1]
        <= temp)
    {
        v.push_back(temp);
        return;
    }
    int val = v[v.size()-1]
    v.pop_back()
```

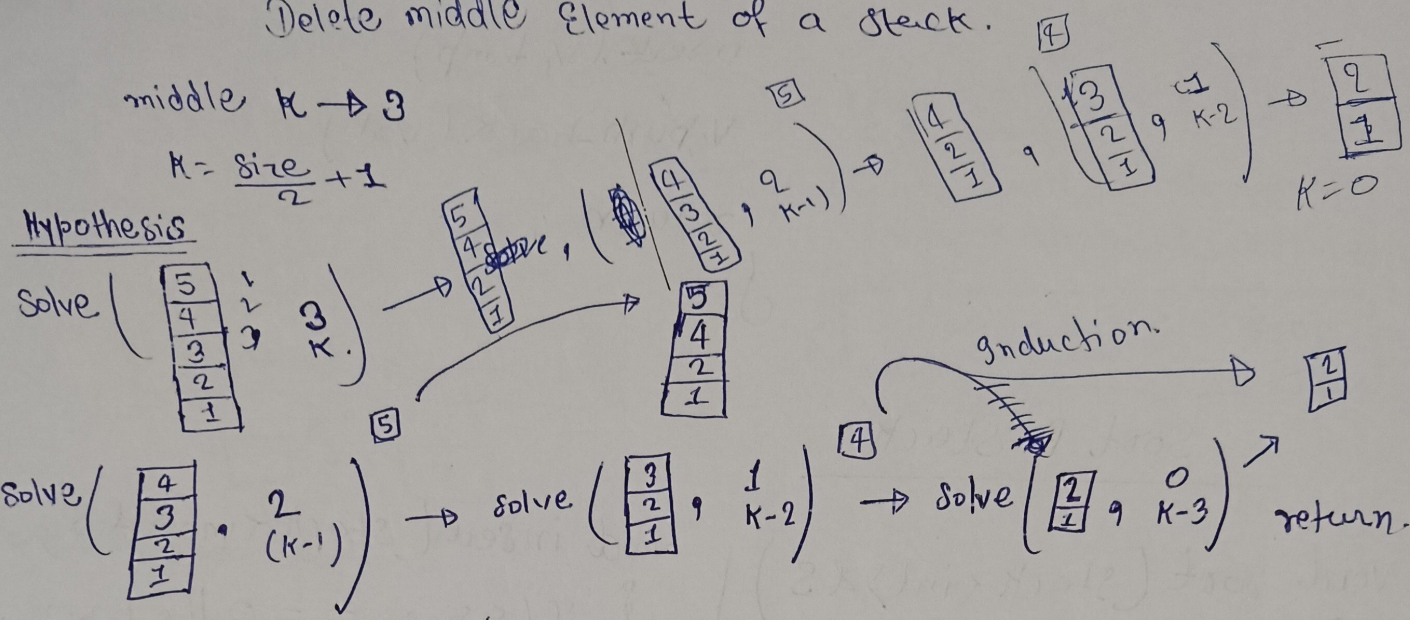
```
    insert (V, temp)
    V.push_back (val)
    return ;
}.
```

Sort A Stack

```
void sort (stack<int>&S)
{
  if (S.size() == 1)
  {
    return;
  }
  int temp = S.top()
    S.pop()
  Sort (S)
  insert (S, temp)
  return ;
}
```

```
void insert (stack<int>&S)
{ if (S.size == = 0 || S.top() = temp)
  {
    S.push (temp)
    return
  }
  int val = S.top()
  S.pop()
  int sort (S, temp)
    S.push(n)
  }. return
```

Delete middle Element of a stack.



code

```
stack<int> middel (stack<int> s, int size){
    if (s.size() == 0){
        return s;
    }
    int K = s.size()/2 + 1;
    solve(s, K)
    return s;
}
```

```
void solve(stack<int> &s, int K){
    if (K == 1){
        s.pop();
        return;
    }
    int temp = s.top()
    s.pop();
    solve(s, K-1);
    s.push(temp);
    return;
}
```

kth symbol Grammer.

$0 \rightarrow 01$
$1 \rightarrow 10$

N = 1 K = 1 return 0;

n = no. of row
k = no. of colum.

2^0 N = 1 : 0
2^1 N = 2 : 0 1
2^2 N = 3 : 0 1 1 0
2^3 N = 4 : 0 1 1 0 1 0 0 1

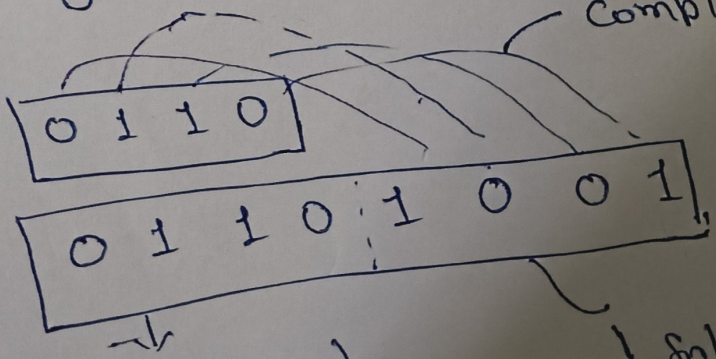
N-1 : 0 1 1 0
N : 0 1 1 0 1 0 0 1

length / 2 : $\frac{2^{N-1}}{2}$ = mid.

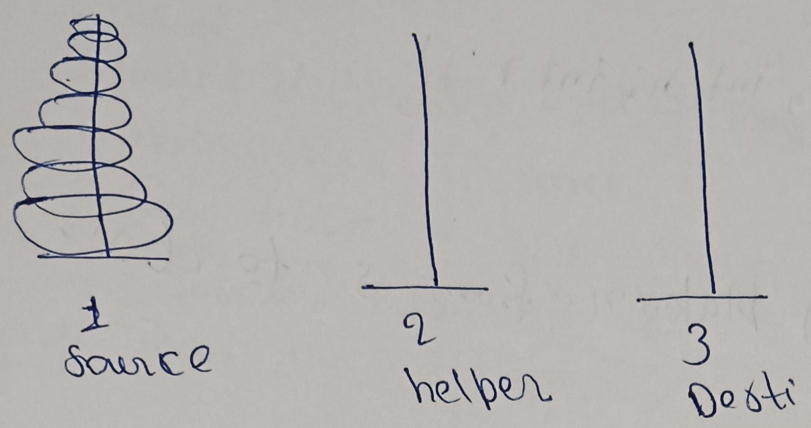
Code.

```
int solve(int N, int K) {
  if(N==1 && K==1) {
    return 0;
  }
  int mid = pow(2, N-1) / 2;
  if (K<=mid)
  {
    return solve(N-1, K);
  }
  else {
    return !solve(N-1, K-mid);
  }
```

e.g

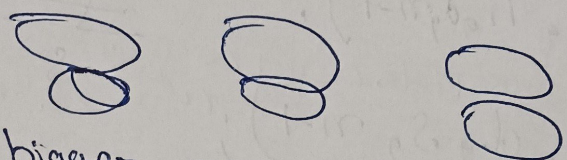


Tower of Hanoi Recursion.



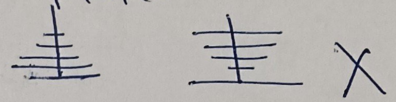
Condition: 1.) Pick 1 Plate at a time.

2.)



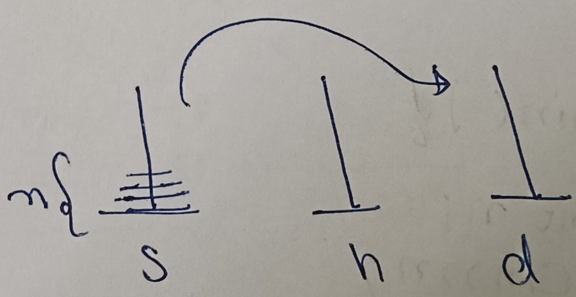
not allow

→ bigger are not above the smaller plate.



i) Hypothesis

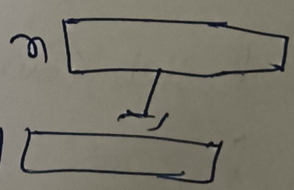
→ Solve(n, s→d, h)



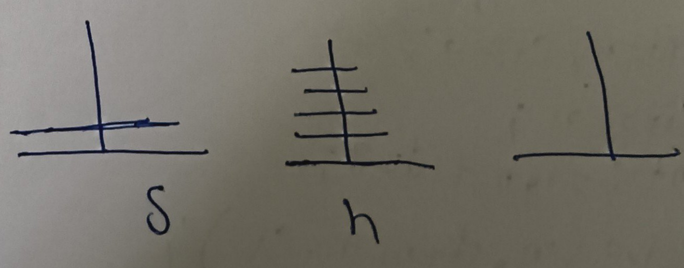
ii) for smaller input.
का काम

Solve (n-1, s→d, h)

IBH method bus एक input chota करके sochte h. उसके आगे नहीं sochte.



Solve (n-1, s→h, d) →

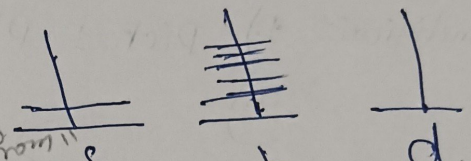


Solve (n-1, h

Code

```
void solve(int s, int d, int h, int n){
    if(n==1)
    {
     cout<<"moving plate"<<n<<"from"<<s<<"to"<<d;
     return;
    }
    solve(s, h, d, n-1);
    cout<<"move plate"<<n-1<<"from"<<s<<"to"<<d<<"n";
    solve(h, d, s, n-1);
    return;
}

int main(){
    int n;
    cin>>n;
    int s=1, h=2, d=3;
    solve(n, s, d, h);
}
```



जब no. of step count करना होगा

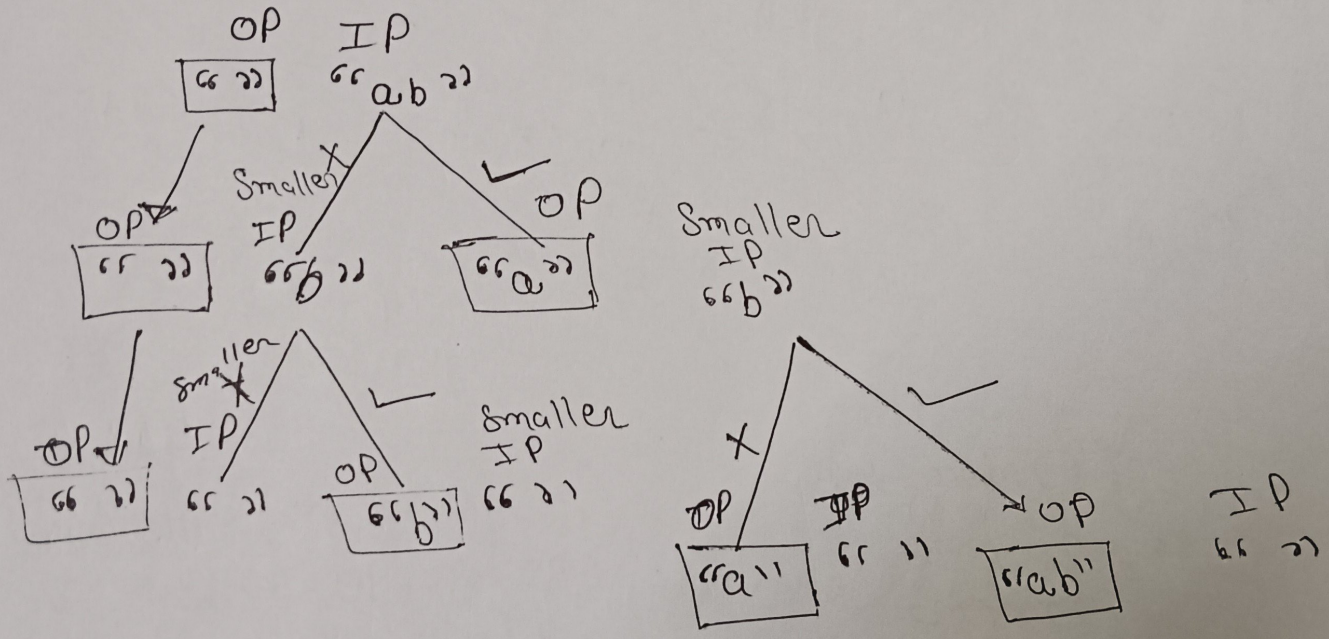
```
int main(){
  int n, ct=0;
  cin>>n;
  int s=1, h=2, d=3;
  solve(n, d, s, h, ct);
}
```

```
void (int s, int d, int h, int n,
    int ct)
{  ct++;
 if(n==1)
  { return; }
 solve(s, h, d, n-1)
 solve(h, d, s, n-1)
}return
```

IP - OP method.

→ subset
→ subset duplicates and other variation
→ Permutation ~~C~~ space
→ Permutation ~~c~~ case change.
→ Josephus problem.
→ Print space String.

IP-OP method.



Code

```
void solve(String IP, String OP)
{
    if(IP.size()==0)
    {
        cout<<OP<<endl;
        return;
    }
    String OP1=OP;
    String OP2=OP;
```

```
int main(){
    String IP;
    cin>>IP;
    String OP;
    solve(IP,OP);
}
```

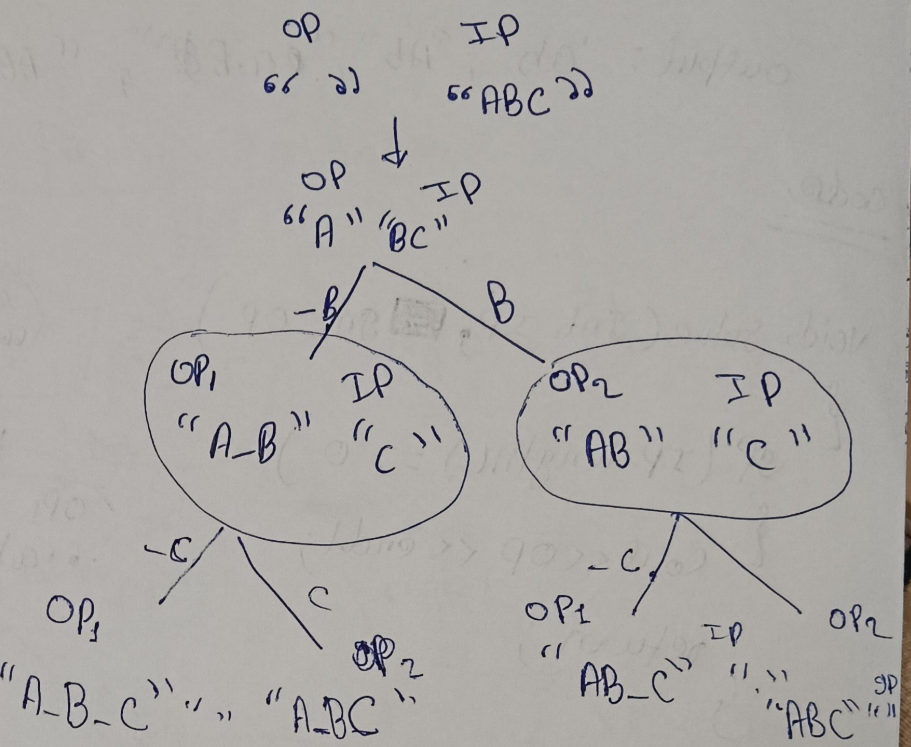
```
OP2.push_back(IP[0]);
IP.erase(IP.begin()+0);
solve(IP,OP1);
solve(IP,OP2);
}
```

Print unique subsets And variations.

Permutation with Space

input ABC

output: A_B_C
A_BC
AB_C
ABC



code

```
void solve(int IP, int OP)
{   string OP1 = OP, OP2 = OP;
    if(IP.size() == 0) {
        cout << OP << " ";
        return;
    }
    OP1.push_back("_");
    OP1.push_back(IP.begin() + 0)
    OP2.push_back(IP.begin()+0);
    IP.erase(IP.begin() + 0);
    solve(IP, OP1);
    solve(IP, OP2);
}
```

```
int main() {
    string IP;
    cin >> IP;
    String OP = "";
    OP.push_back(IP[0]);
    IP.erase(IP.begin() + 0);
    Solve(IP, OP);
}
```

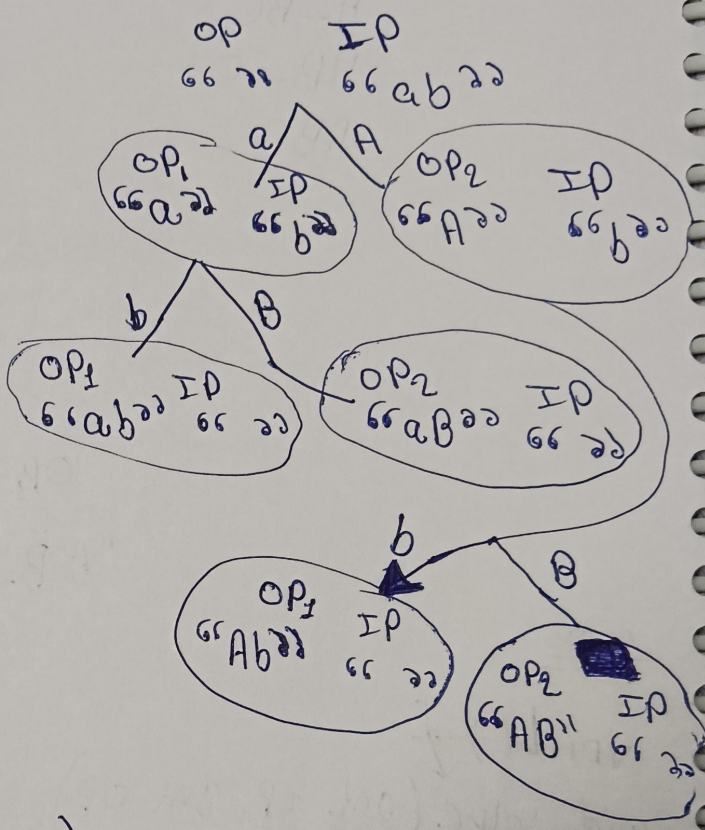
Permutation with case change.

Input: "AB"

Output: "ab", "Ab", "aB", "AB"

* IP - OP method.

Code




```
void solve(int IP, int OP)
{
    if (IP.length() == 0)
    {
        cout<<OP<<endl;
        return;
    }
    string OP1 = OP;
    string OP2 = OP;
    OP1.push_back(IP[0]);
    OP2.push_back(toupper(IP[0]));
    IP.erase(IP.begin()+0);
    solve(IP, OP1);
    solve(IP, OP2);
    return;
}
```

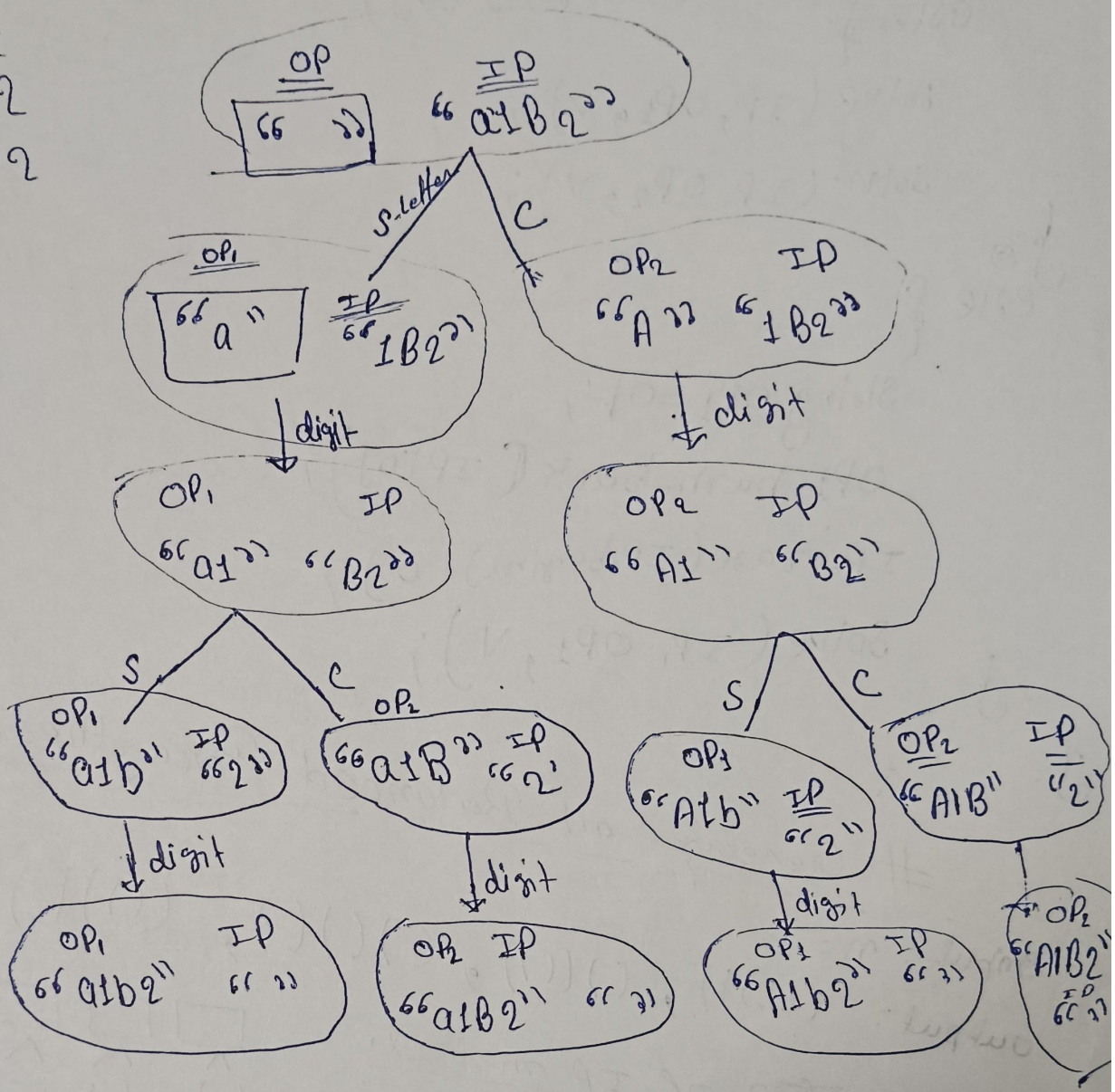
```
int main() {
    string IP;
    cin>>IP;
    string OP = "";
    solve(IP, OP);
    return;
}
```

Letter Case Permutation

input: a1B2

output: a1B2
a1b2
A1B2
A1b2

IP - OP method.



Code.

```
vector<string> fun(string s){
{ string IP = s;
  String OP = "";
  vector<String>V;
  Solve (IP, OP, V);
} return V;
```

~~void~~

```
void solve(string IP, string OP,
           vector<string>& V)
{
  if (IP.length() == 0) {
     V.push_back (OP)
  } return;
if (isalpha(IP[0]))
{  String OP1 = OP;
   String OP2 = OP;
```

```
op1.push_back(tolower(ip[0]));
op2.push_back(toupper(ip[0]));
ip.erase(ip.begin()+0);
}
solve(ip, op1, v);
solve(ip, op2, v);
}
else {
string op1 = op;
op1.push_back(ip[0]);
ip.erase(ip.begin()+0);
solve(ip, op1, v);
}
```

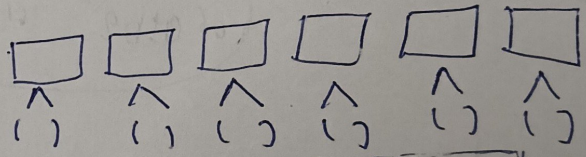
3.

Generate all Balanced Parenthesis.

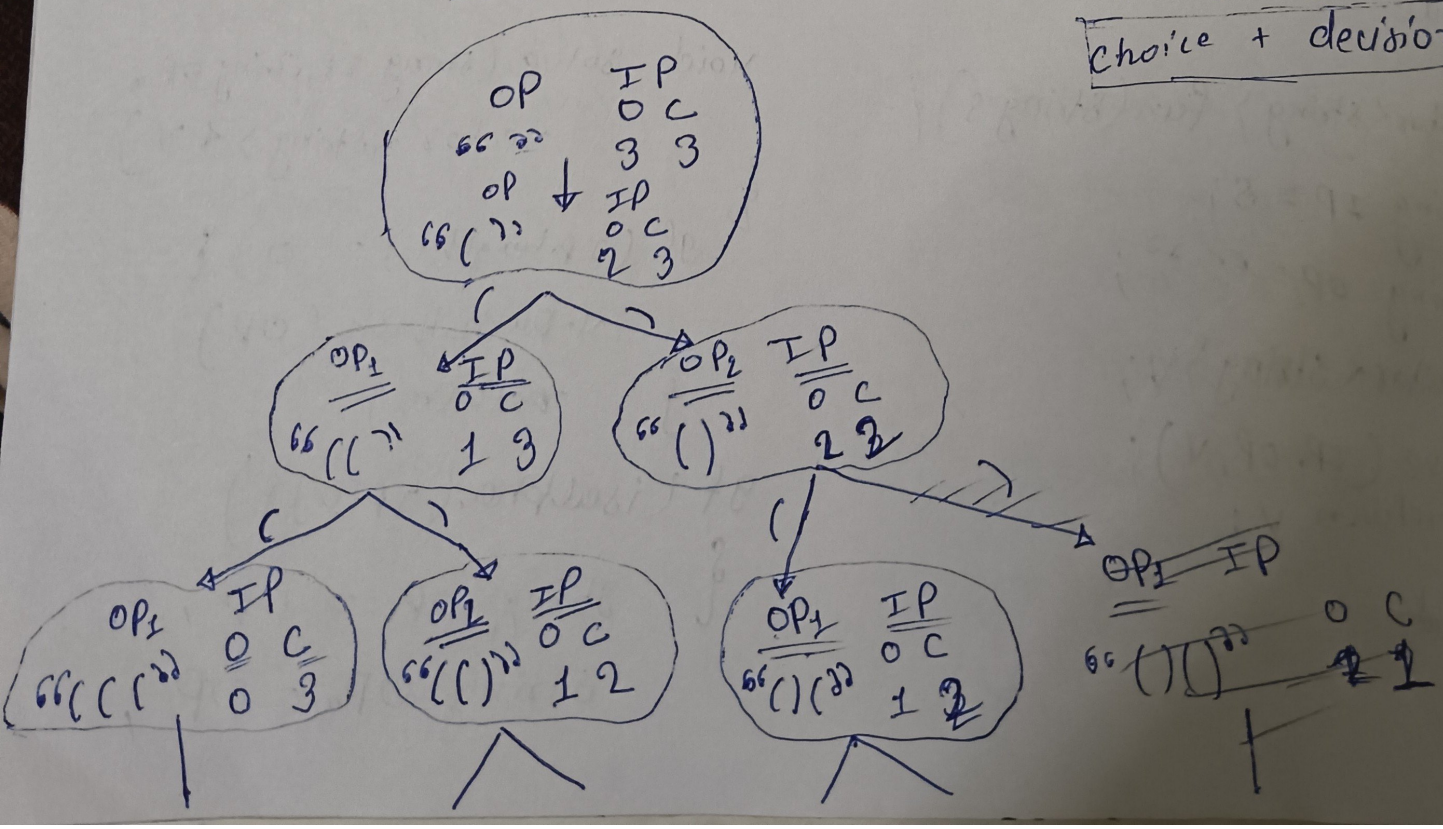
input: n = 3

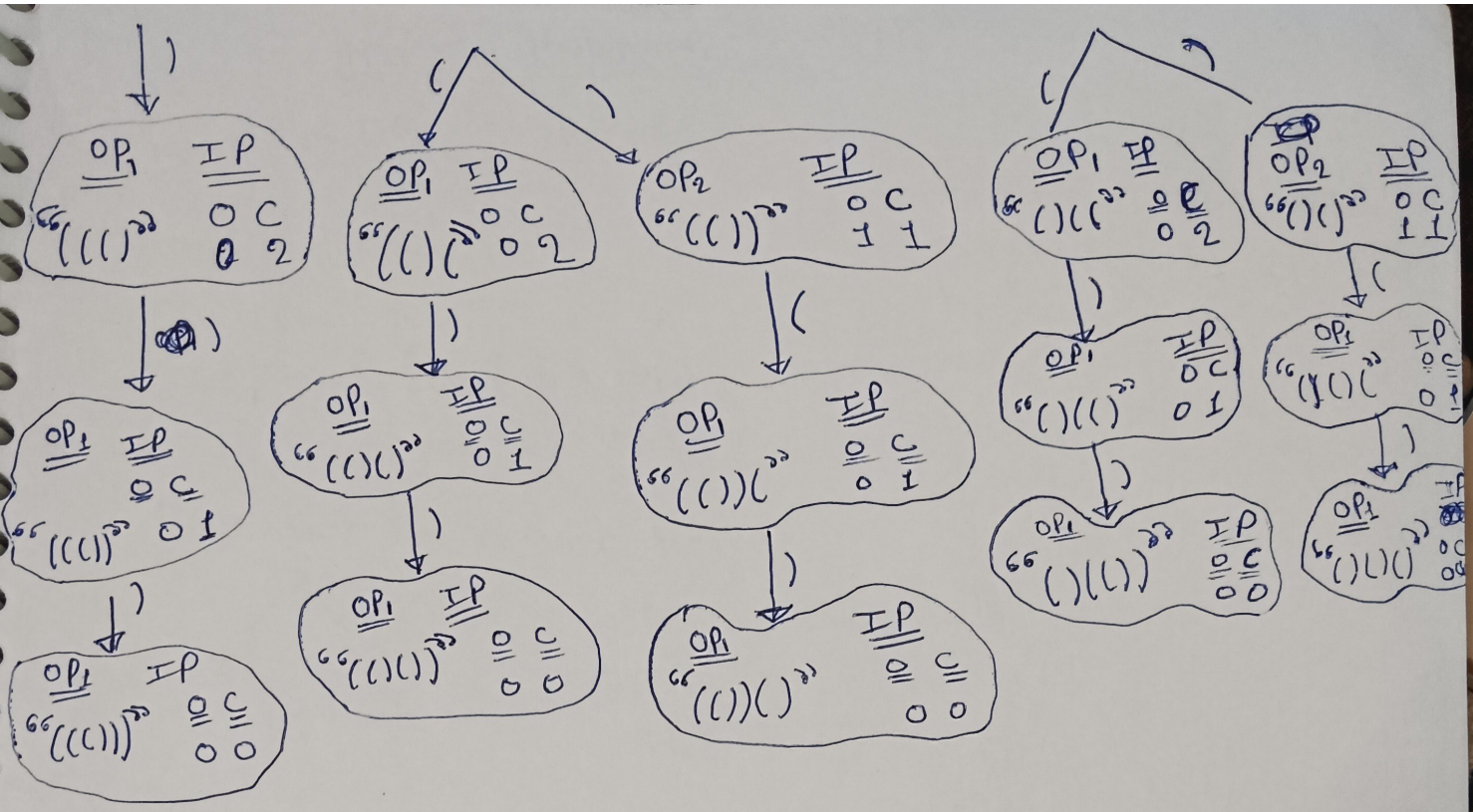
output: ((())), ()(()), ()()(), (())(), (()())

op-ip method:



choice + decision.





```
void solve(int o, int c, string op,
    vector<string> &v) {
  if(o==0 && c==0){
     v.push_back(op);
     return;
  }
  if(o!=0){
     string op1=op;
     op1.push_back('(');
     solve(o-1, c, op1, v);
  }
  if(c>o){
     string op2=op;
     op2.push_back(')');
     solve(o, c-1, op2, v);
  }
}
```

```
vector<string> fun(int n){
   int o=n;
   int c=n;
   vector<string>v;
   string op="";
   solve(o,c,op,v);
   return v;
}
```