Recursion

Page No.
Date

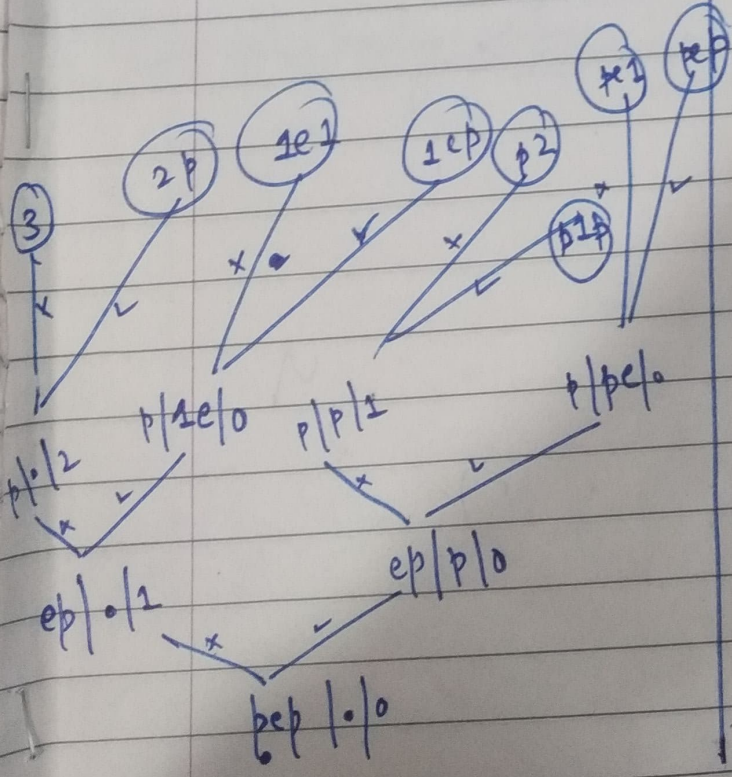
① Sub-Sequence

abc
a
b
c
ab
ac
bc
abc

```
void subsequence(string in, string out)
{
  if(in.length() == 0)
      cout << out; return;

  subsequence(in.substr(1), out + in[0]);
  subsequence(in.substr(1), out);
}
```

Level Up (1·1)

pep	① 000	pep
	② 001	pe1
	③ 010	p1p
	④ 011	p2
	⑤ 100	1ep
	⑥ 101	1e1
	⑦ 110	2p
	⑧ 111	3




```
void solve(string in, string out, int count)
{ if(str.length()==0)
  { if(count!=0)
      cout<< out + count
    else
      cout<<out;
    return;
  }
if(count==0)
  solve(in.substr(1), out+in[0], 0);
else
  solve(—, out+count+in[0], 0);
solve(—, count+1, out);
}
```

x → count + 1
✓ → 0

(2) Phone Keypad

```
vector<string> keypad = { "", "", "abc", "def", "ghi", "jkl",
                          "mno", "pqrs", "tuv", "wxyz" };

void phonekeypad(vector<int> in, string out, int
{
    if(i== in.size())
        cout<<out; return;
    if( in[i]==0 || in[i]==1)
        phonekeypad(in, out, i+1);

    for(int k=0; k< keypad[in[i]].length(); k++
    {
        phonekeypad(in, out+keypad[i][k], i
    }
}
```

(3)

Page No.
Date

(3) P n C

GB
q1, q2

arr(4,0)

```
void permute(vector<int> arr, tp, qpst, string ans)
{ if( tp = qpst )
      cout << ans; return;

  for(int box=0; i<arr.size(); i++)
     { if(arr[box] != 1) arr[box] = 1;
permute( arr, tp, qpst+1, ans + "qpst+1" -> "b
        arr[box] = 0;
     }
}
```

```
void combination
    int box = index+1
```

-1

(box)

→ coin change < P / C > → can use same coin multiple time

(int) 0

Page No. _____
Date

(4) Board Ladder

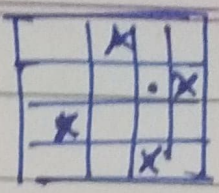
```
void BoardGame (int src, int dest, ladder, string ans)
  {
    if(src==dest) cout<<ans; return;
    if (src>dest) return;

    if (src==0)
      {boardGame (1, dest, ladder, ans+"1");
       boardGame (1, dest, ladder, ans+"6");
      }

    if (ladder [src] !=0)
      boardGame (ladder [src], dest, lader, ans+(
    else
      { for (dice=1; dice<=6; dice++)
        { int inter = src+dice;
          boardgame (inter, dest, ladder, ans+"dice");
        }
      }
  }
```

// 1, 6 opening

// Ladder

Page No.
Date

(5) N-Queen



```
void nqueen(vector<vector<int>>chess, int tpq,
                string ans, int index)
{ if (tpq = chess.size())
        cout << ans ; return;

for (int i= index+1; i<chess.size()*chess.size();
  { int row = i / chess.size();
    int col = i % chess.size();

Permutation
& not
Combination
    if(chess[row][col]==0)
     { if ( isQueenSafe(chess, row, col)
          { chess[row][col] = 1;
      nqueen (chess, tpq+1, ans+(i), i)
            chess[row][col] = 0;
     } }
}
```

Page No.
Date

→ bool isQueenSafe (chess, row, col)

{

vector <vector <int>> dir =

dir

{ {-1, 0}, {1, 0}, {1, 1}, {-1, -1}, {0, 1}, {0, -1}

{-1, 1}, {1, -1} };

for (int i=0; dir.size(); i++)

{ for (int dist=1, true; dist ++)

{ int eqcol = col + dist * dir[i][1];

int eqrow = row + dist * dir[i][0];

if (eqcol < 0 || eqrow < 0 || eqcol >= chess.size -

break;

if (chess [eqrow][eqcol] == 1)

return false;

}

return true;

}

→ bool isKnightSafe

8 dir

{-1, 2}, {1, -2}, {-2, 1}, {2, 1} {-1, -2}, {1, 2}

{-2, -1} {2, -1}

for (int i=0; i<dir.size(); i++)

int eqcol = col + dir[i][1];

int eqrow = row + dir[i][0];

Page No.
Date

(6) Permutation

abc
acb
bac
bca
cab
cba

(7) Ge

```
void permutation (string in, string ans)
{ if (in.length()==0)
    cout << ans << return;

  for (int i=0; i<in.length(); i++)
  { char ch = in[i];
    string left = in.substr(0,i);
    string right = in.substr(i+1);
    permutation (left+right, ans+ch
  }
```

2
(())
()()

Page No.
Date

(1) Generate Brackets

```
void generateBrackets( int i, int n, int closing,
                       int opening
                       string ans )
{
    if (i == 2*n)
        cout << ans; return;

    if ( opening < n)
        generate (i+1, n, closing, opening+1, ans+"{");

    if ( closing < opening)
        generate (i+1, n, closing+1, opening, ans+"}");

}
```

2^n

$(n * m)$ = possible

DP ques

⑧ 0/1 Knapsack

```
int knapsack (vector<int> weights (w), vector<int> pric
              int i, int cap)
{ if (i==w.size || cap == 0)
      return 0;

  int ans = 0;
  int inc, exc = 0;

  if (cap >= w[i])
    { inc = p[i] + knapsack (w, p, i+1, c
  exc = knapsack (w, p, i+1, cap);

  ans = max (inc, exc);
  return ans;
}
```

dP [i] [cap]

→ Fractional Knapsack

	1	2	3	
Price	280	100	120	120
weight	40	10	20	24
ratio	7	10	6	5

cap = 60

50-10 · 50-40	sort	10	7	6 5

(10) Solve Sudoku — Backtracking

```
bool solve (int mat[][9], int i, int j, int n
  {
    if ( i==n) printMat. return true;

// next line  if (j==n)
                return solve (mat, i+1, 0, n);

// prefilled    if (mat[i][j] !=0)
                return solve ( mat, i, j+1, n);

for ( int num=1 ; num<=9; num++)
  {
    if (canPlace(mat, i, j, num))
      { mat[i][j] = num;
        bool canweSolve = solve(mat, i, j+1, n);
        if (canweSolve)
          return true;
      }
  }
mat[i][j]=0;
return false;
}
```

```
bool canPlace ( int mat[][], int i, int j, int num)
{
    // row & col
    for(int x=0; x<n ; x++)
    { if(mat[i][x] == num || mat[x][j] == num)
        return false;
    }

    // small box
    int rn = sqrt(n);
    int sx = (i/rn) * rn;
    int sy = (j/rn) * rn;

for (int x=sx ; x<sx+rn ; x++)
    for ( y=sy; y<sy+rn ; y++)
        if ( mat[x][y] == num)
            return false;

return true;

}
```

(ii) Reverse Stack

```
void reverse (stack<int> s)
{   if (s.empty()) return;
    int top = s.top();
    s.pop();
    reverse stack (s)
    insertAtBottom (s, x);
}

void insertAtBottom (stack<int> s, int x)
{   if (s.empty())
        { s.push(x); return; }

    int top = s.top();
    insertAtBottom (s,x);
    s.push (top);
}
```

"123" → ABC
AW
LC

Page No.
Date

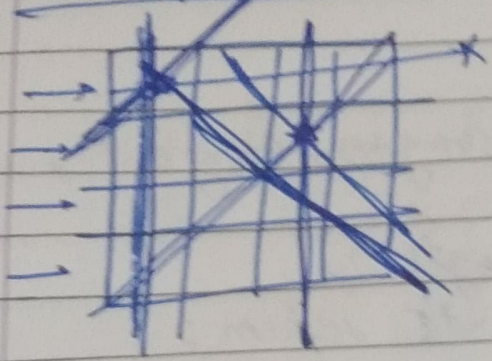
(12) Num To Chars

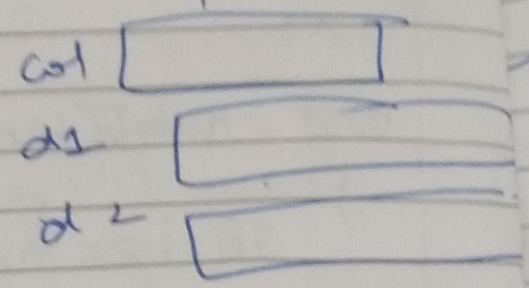
```
void numToChars( string in, string out, int i)
{
    if( i >= in.length())
        cout << out; return;

int fdigit = in[i] - '0';
char ch    = fdigit + 'A' - 1;

numToChars ( in, out+ch , i+1);

    if( i+1 != in.length())
    { int sdigit = in[i+1] - '0';
      int no = fdigit*10 + sdigit;
      if (no <= 26)
          char ch = no + 'A' - 1;
          numToChars( in, out+ch, i+2);
      }

}
```

Level Up

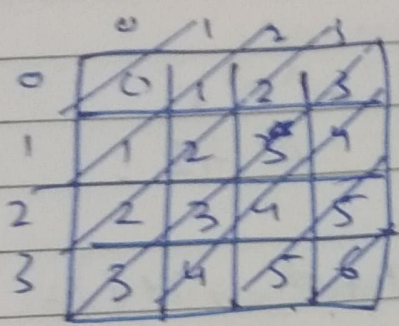
Page No.
Date

5.2 N-Queen

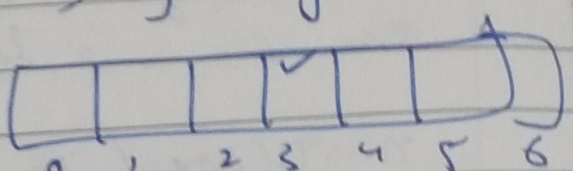




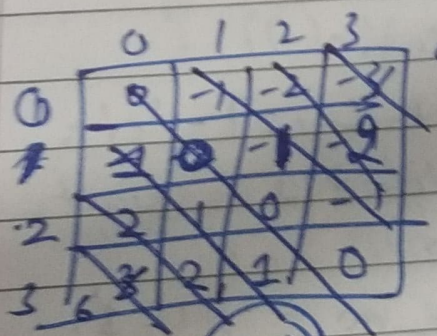
r+c

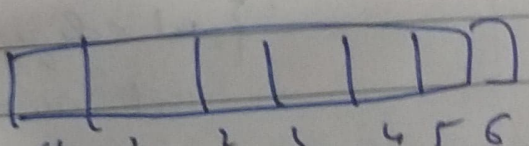


(2×n)-1 diagonals



r-c





4-1 +3

diag1[ ]
diag2[2n-1]
col [n]

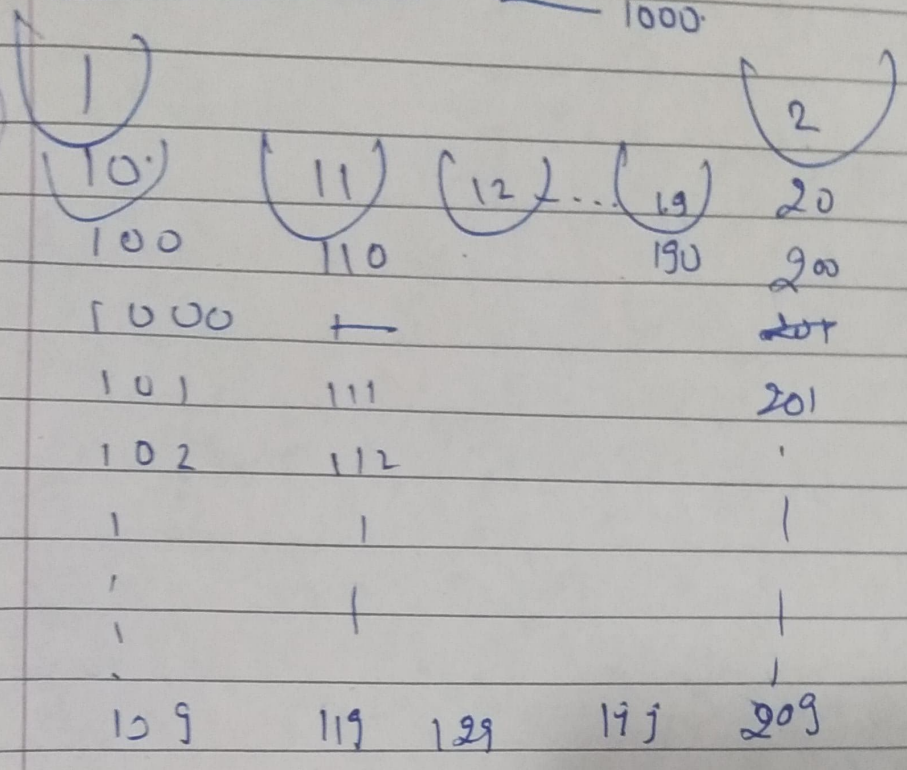
solve(0, n, diag1, diag2, ans);

```
void Solve (row, n, dig1, dig2, col, a
{if (row == n)
    cout<<ans; return;
for(c=0; c<n; c++)
 {if(col[c]==false && dig[r+c]==
 && dig[r-c+n-1]==false)
 { [ ] = true.
   solve(row+1, n, d1, d2, col,
                 ans + "row"+"
 [ ] : false
```

Page No.
Date

Lexical Order (dictionary type)

— 1000.




```
int n = 1000;

for (int i=1; i<=9; i++)
    dfs (i, n);

void dfs (i, n)
{ if (i>n) return;
  cout << i;
  for (int j=0; j<10; j++)
    { dfs(i*10 + j, n);
  }
}
```