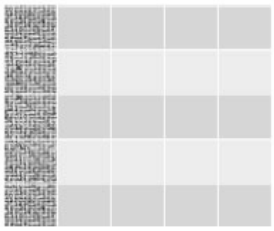
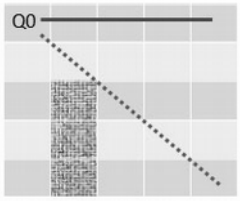
// 8 Queen (Uva-11195)
// Backtracking with Bitmask

```
//row, column, leftDiagonal, rightDiagonal
//placing queen in row major order
//we check if it is possible to place the queen in that row(of a fix column)
//if so, move forward
int cnt = 0, ALL = (1 << 5) – 1;                    // testing for n = 5 queens
void backtrack(int r, int c, int ld, int rd) {
   if(r == ALL) {                                   //ALL = n number of 1 in bitset (starting from least)
      cnt++;                                        //if all rows are taken
      return; }
//pos = (those bits which we want to work with(to avoid overflow)) & (~(row where queen placed | if left diagonal attacked || if right diagonal attacked))

   int pos = ALL & (~(r | ld | rd));                //negate the or, so the set bits are accessable

//there are also set bits, whose position is greater than n, so to turn them off, we used ALL with it (&)

   while(pos) {                 //pos is the suitable places for placing queen, place queen in each position, and move forward
      int place = pos & -pos;                       //get right most set bit
      pos -= place;                                 //turn off the right most set bit from place
      if(mp[make_pair(place, 1 << c)])              //checking if the place can be used
         continue;
  //for each (left to right)column left diagonal moves left from the queens place(s), right diagonal moves right from the queens place(s)
      backtrack(r | place, c+1, (ld | place) << 1, (rd | place) >> 1);
} }
```



```
pos = 11111 & ~00000 = 11111 (p = 1)
```




```
rw                = 00001 (1)
ld = 00001 << 1 = 00010 (2)
rd = 00001 >> 1 = 00000|1 (0, the LSB is removed)
----------------------- OR
                  00011 -> NEGATE -> 11100
pos = 11111 & 11100 = 11100 (p = 4)
```

Fig : 8 Queens Simulation

// Subset Sum
// Bitmask Trick
// Complexity : O(2^n)

```
void SubsetSum(int val[], n) {                      // val[] contains element value, n is the length of val array
   int maxVal = 0, bitPos;
   for (i = 0; i < (1 << n); i++) {                 // The main routine, variable ‘i’ (the bitmask) has been declared earlier
      int sum = 0, cnt;                             // For each subset, O(2^n)
      for (int j = 0; j < n; j++) {                 // Check membership, O(n)
         if (i & (1 << j))                          // Test if bit ‘j’ is turned on in subset ‘i’?
            sum += val[j],  cnt++;                  // If yes, process ‘j’
      }
```

```
      if(sum > maxVal) {
         maxVal = sum;
         bitPos = I;                          // The answer is found: bitmask 'i'
      }  }
   for(register int i = 0; i < l; i++)        // Prints the taken values
      if(pos & (1 << i))
         printf("%d ", val[i]);
}
```

// Iteration with Bitwise (Complete Search)
// Uva – 725 Division

/*Abridged problem statement: Find and display all pairs of 5-digit numbers that collectively use the digits 0 through 9 once each, such that the first number divided by the second is equal to an integer N, where 2 ≤ N ≤ 79. That is, abcde / fghij = N, where each letter represents a different digit */

```
//Main part of code
for (int fghij = 1234; fghij <= 98765 / N; fghij++) {
   int abcde = fghij * N;                     // This way, abcde and fghij are at most 5 digits
   int tmp, used = (fghij < 10000);           // If digit f=0, then we have to flag it as used
   tmp = abcde;
   while (tmp) {
      used |= 1 << (tmp % 10); tmp /= 10; }   // Marking all digits as used
   tmp = fghij;
   while (tmp) {
      used |= 1 << (tmp % 10); tmp /= 10; }   // Marking all digits as used
      if (used == (1<<10) – 1)                // If all digits are used (111111111)
         printf("%0.5d / %0.5d = %d\n", abcde, fghij, N);  // If all digits are used, print it
}
```

// Iteration (Complete Search)
// Uva – 441 Lotto

//Given 6 < k < 13 integers, enumerate all possible subsets of size 6 of these integers in sorted order. (12 C 6 = 924 outputs)

```
// Main chunk code
for (int i = 0; i < k; i++)                   // input: k sorted integers
   scanf("%d", &S[i]);
for (int a = 0 ; a < k - 5; a++)              // six nested loops!
   for (int b = a + 1; b < k - 4; b++)
      for (int c = b + 1; c < k - 3; c++)
         for (int d = c + 1; d < k - 2; d++)
            for (int e = d + 1; e < k - 1; e++)
               for (int f = e + 1; f < k; f++)
                  printf("%d %d %d %d %d %d\n",S[a],S[b],S[c],S[d],S[e],S[f]);
```

// Bisection Method
// Complexity : O(log 2 ((max − min) / e)) e : A small threshold 1e-9

```
void bisection() {
   double lo = 0.0, hi = 10000.0, mid = 0.0, ans = 0.0;
   for (int i = 0; i < 50; i++) {          // Looping 50 times should be precise enough as log_2 ((10000.0 - 0.0) / 1e-9) ~= 43
```

```
    mid = (lo + hi) / 2.0;                  // Try the middle value
  if (can(mid)) {                           // 'can' function is a simulator that tests if this mid value can be the answer
    ans = mid;                              // Save the value, then continue
    hi = mid; }       // Note : This version of code gives floating values this can also be modified to find integer value as well
  else                // if (lo + hi) & 1== 1 {mid1 = (lo+hi+1)/2, mid2 = (lo+hi-1)/2;}   |use the best option according to problem
    lo = mid;         // else mid1 = (lo+hi)/2;
}
```

// Recursive and Dynamic Programming
// Uva 10003 Cutting Sticks

/*Given a stick of length $1 \le l \le 1000$ and $1 \le n \le 50$ cuts to be made to the stick (the cut coordinates, lying in the range [0..l], are given). The cost of a cut is determined by the length of the stick to be cut. Your task is to find a cutting sequence so that the overall cost is minimized. */

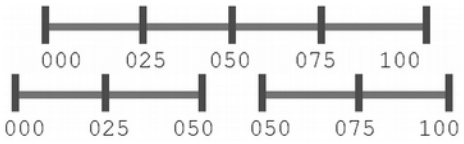


Fig: Cutting Sticks Illustration (optimal 200)

```
int l, n, A[55], memo[55][55];

int cut(int left, int right) {
  if (left + 1 == right)                    // If left + 1 == right, there is no space to cut!
    return 0;
  if (memo[left][right] != -1)              // Memorization
    return memo[left][right];
  int ans = 2000000000;                     // An INF value
  for (int i = left + 1; i < right; i++)
    ans = min(ans, cut(left, i) + cut(i, right) + (A[right]-A[left]));
  return memo[left][right] = ans;
}
```

// Some Bitwise Operations

```
#define isOn(S, j) (S & (1 << j))
#define setBit(S, j) (S |= (1 << j))
#define clearBit(S, j) (S &= ~(1 << j))
#define toggleBit(S, j) (S ^= (1 << j))
#define lowBit(S) (S & (-S))
#define setAll(S, n) (S = (1 << n) - 1)

#define modulo(S, N) ((S) & (N - 1))                    // returns S % N, where N is a power of 2
#define isPowerOfTwo(S) (!(S & (S - 1)))
#define nearestPowerOfTwo(S) ((int)pow(2.0, (int)((log((double)S) / log(2.0)) + 0.5)))
#define turnOffLastBit(S) ((S) & (S - 1))
#define turnOnLastZero(S) ((S) | (S + 1))
#define turnOffLastConsecutiveBits(S) ((S) & (S + 1))
#define turnOnLastConsecutiveZeroes(S) ((S) | (S - 1))

void printSet(int vS) {                                 // Integer to Binary
  stack<int> st;
  while (vS)
    st.push(vS % 2), vS /= 2;
  while (!st.empty())                                   // To reverse the print order
    printf("%d", st.top()), st.pop(); printf("\n"); }
```