GARLAND

```
int dp[105][105][105][2];

int a[105];

int n;

/*

  5 36 7 3 2 2 5 4 6 2 5 7 4  7  8 4

*/

int f(int i,int odd,int even,int prev){

  if(i==n)
    return 0LL;

  else if(dp[i][odd][even][prev]!=-1)
    return dp[i][odd][even][prev];

  //two cases
  //attached or removed bulb

  if(a[i]){

    int parity=(a[i]&1LL);

    return dp[i][odd][even][prev]=f(i+1,odd,even,parity)+(prev!=parity);

  }else{
```

```
    int op1=INF,op2=INF;

    if(odd){

      op1=f(i+1,odd-1,even,1)+(1-prev);

    }

    if(even){

      op2=f(i+1,odd,even-1,0)+prev;

    }

    return dp[i][odd][even][prev]=min(op1,op2);
  }
}







inline void solve(){

  // int n;
  cin>>n;

  // vector<int>a(n);
```

```
int odd=0,even=0;
for(int i=0;i<n;++i){
  cin>>a[i];
  if(a[i]==0)
    continue;
  if(a[i]&1LL)
    odd++;
  else
    even++;
}

//odd,even represents number of odds and evens in given array
odd=(n+1)/2-odd;
even=n/2-even;

memset(dp,-1LL,sizeof(dp));
//now odd,even represents number of odds /evens we can take
//f(0,odd,even,prev)

//attached
if(a[0]){

  int parity=(a[0]&1ll);
  out(f(1,odd,even,parity));

}else{

  int op1=INF,op2=INF;

  if(odd){
    op1=f(1,odd-1,even,1);
  }
```

```
    if(even)
      op2=f(1,odd,even-1,0);

    out(min(op1,op2));//answer

  }


}
```

Bit Manipulation

Decimal System -> 10 digits 0-9
Binary System -> 2 digits 0 and 1 -> bits

3 -> 11 -> 2^1 + 2^0

Decimal -> 97 -> 9*(10^1) + 7*(10^0)

Set -> Bit is 1
Unset -> Bit is 0

10110 -> 2^1 + 2^2 + 2^4 = 2+4+16 = 22

Operators -> Bitwise AND, OR, NOT, XOR

1 AND 1 -> 1 (Both the operands should be 1 to give 1 as output)

10110
01101
00100 -> Bitwise AND

1 OR 0 -> 1 (Resut will be 1 if either of the operands are 1)

10010
11000
11010 -> Bitwise OR

NOT 1 -> 0 (Inverts the operands)
NOT 0 -> 1

11101
00010 -> Bitwise NOT

1 XOR 1 -> 0 (Result will be 1 if both operands are different, and 0 if both are same)
1 XOR 0 -> 1
0 XOR 1 -> 1
0 XOR 0 -> 0

10010
11000
01010 -> Bitwise XOR

Shifts -> Left Shift, Right Shift

000110 left shift 2 -> 00011000
110101 right shift 2 -> 001101

Bitwise AND -> &

Bitwise OR -> |
Bitwise XOR -> ^
Bitwise NOT -> ~

Left shift -> <<
Right Shift -> >>

2 & 1 -> 10 & 01 = 0
2 | 1 -> 10 | 01 = 11 = 3
2 ^ 1 -> 10 ^ 01 = 11 = 3
~2 -> ~(10) = 01 = 1

int -> 32 bits -> 2^0 + 2^1 + .. 2^9 + 2^11 + .. 2^30 (2^33 not possible)
111000011010101011010101 -> maximum allowed width is 32
long long -> 64 bits
1010100101100011101000010101001101 -> maximum width is 64
2^40 + 2^41

int x = 3; // 000000...0011 -> 30 zeroes and 2 ones
int y = ~x; //1111111...1100 -> 30 ones and 2 zeroes

int z = 3<<1; // 00000..0011 << 1 -> 000..00110 -> 6 = 3*2
to multiply by 2^i -> perform x<<i
Similarly
z >> 5 -> equivalent to z / 2^5

int z -> 1110000000..0000 width 32
z << 5 -> 11100 000000000..00 width 32 -> overflow

int w = 000...1110 width 32 = 2+4+8 = 14
w >> 6 -> 00...0000 001110
//14/64 -> answer is not incorrect but set bits are being lost

long long z = 1110000..00 width 32 -> 0000..00 11000..0000 width 64
z<<5 -> 000.. 00 11100 0000000...000 -> no loss

1<<3 = 8 equivalent to 1 * (2^3)
1<<i = 2^i -> 1 is being treated as int
1LL<<i -> 1 is being treated as long long, safe to use i = 30, 40, 50 etc.

checking whether a particular bit is set or not
long long x = 111000011 -> find out whether 3rd bit is set or not, with expression involving bitwise operators. (bits start from 0)

(x >> 3) & 1
111000011 -> 11100001 -> 1110000 -> 111000
111000 & 1 = 111000 & 000001 -> 0

x & (1<<3) -> 111000011 & 000001000 -> 0

for checking ith bit is set or not -> use x & (1LL<<i) == 0

setting the ith bit -> x | (1LL<<i)
unsetting the ith bit -> x & (~(1LL<<i))

11100011 unset the 1st bit
1<<1 -> 00000010
~(1<<1) -> 11111101

  11100011
&11111101
=11100001

toggling the ith bit (0 to 1 and vice versa) -> x ^ (1LL<<i)

toggle 2nd bit of 11100011
  11100011
^00000100
=11100111 -> toggled