请设计一个算法，给一个字符串进行二进制编码，使得编码后字符串的长度最短。

```
#include<iostream>
#include<queue>
#include<algorithm>
#include<string.h>
#define MAX 1000
using namespace std;
int main()
{
    char newString[MAX] = {0};
   while(cin>>newString)
    {
        int i, j;
       int countNum = 0;      //统计不同字符个数
       int sum = 0;           //记录编码后的长度
       int first = 0, second = 0;       //分别记录队列的最小两个值
        int len = strlen(newString);
        priority_queue <int, vector<int>, greater<int> > huffmanQueue;    //定义小值优先级高的队列
         sort(&newString[0], &newString[len]);

        for(i = 0; i < len; )
        {
              j = i;
              while((j < len)&&(newString[j] == newString[i]))
              {
                  j++;
              }
             huffmanQueue.push(j - i);    //将字符 newString[i]的个数压入队列
              i = j;
             countNum++;
       }
       for(i = 0; i < countNum - 1; i++) //霍夫曼编码步骤
       {
            first = huffmanQueue.top();
             huffmanQueue.pop();
           second = huffmanQueue.top();
           huffmanQueue.pop();
           huffmanQueue.push(first + second);
           sum += first + second;
```

```
        }
        cout<<sum<<endl;
    }//while
    return 0;
}
```

对于一个由 0..n 的所有数按升序组成的序列，我们要进行一些筛选，每次我们取当前所有数字中从小到大的第奇数位个的数，并将其丢弃。重复这一过程直到最后剩下一个数。请求出最后剩下的数字。

因为是从 0 开始，所以第一轮移走的是二进制下最右边为 0 的位置（从 0 开始的偶数位置）上的数，然后我们发现第二轮各个 number 的位置等于 number/2，即从 number 位置到 number>>1 位置，这时候我们依然移走二进制下最右边为 0 的位置（1(**0**1) 5(1**0**1) 9(10**0**1) .....它们第二轮对应的位置是 0， 2， 4），最后剩一个数肯定是 0 到 n 中二进制下 1 最多的那个数，因为它每次的位置都是奇数位置。代码如下

```
#include <cstdio>

int main()
{
        int n;
        while(scanf("%d", &n) != EOF){
                int b = 1;
                while(b < n){
                        b <<= 1;
                }
                printf("%d\n", (b >> 1) - 1);
        }
        return 0;
}
```

有一个二维数组(n*n),写程序实现从右上角到左下角沿主对角线方向打印。
给定一个二位数组 **arr** 及题目中的参数 **n**，请返回结果数组。
测试样例：

```
[[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]],4
```

返回：[4,3,8,2,7,12,1,6,11,16,5,10,15,9,14,13

```
class Printer {
```

```
public:
    vector<int> arrayPrint(vector<vector<int> > arr, int n) {
        // write code here
        vector<int>A;
        int i,j;
        int k1,k2,k;
        for( i=1;i<=n;i++)//打印右上角及对角线
            { k1=
            0;
            k2=n-i;
            k=i;            //标志个数
            while(k--)
                {
                A.push_back(arr[k1][k2]);//压入数据
                k1++;//下标同时增加
                k2++;
                }
        }
        for( i=1;i<=n-1;i++)//打印左下角
            {
            k1=i;
            k2=0;
            k=n-i;            //标志个数
            while(k--)
                {
                A.push_back(arr[k1][k2]);//压入数据
k1++;//下标同时增加
                k2++;
                }
```

```
        }
        return A;
    }
};
```

在股市的交易日中，假设最多可进行两次买卖(即买和卖的次数均小于等于 2)，规则是必须一笔成交后进行另一笔(即买-卖-买-卖的顺序进行)。给出一天中的股票变化序列，请写一个程序计算一天可以获得的最大收益。请采用实践复杂度低的方法实现。

给定价格序列 **prices** 及它的长度 **n**，请返回最大收益。保证长度小于等于 500。

测试样例：

```
[10,22,5,75,65,80],6
```

返回：87

```
1    class Stock {
2    public:
3            int maxProfit(vector<int> prices, int n) {
4                    // write code here
5                    if (n==0){
6                            return 0;
7                    }
8                    vector<int> pre_profit(n,0);
9                    vector<int> post_profit(n,0);
10
11                   int min_buy = prices[0];
12                   for(int i=1;i<n;i++){
13                           min_buy = min(prices[i], min_buy);
14                           pre_profit[i] = max(pre_profit[i-1], prices[i]-min_bu
15                   }
16
17                   int max_sell = prices[n-1];
18                   for(int j=n-2;j>=0;j--){
19                           max_sell = max(prices[j], max_sell);
20                           post_profit[j] = max(post_profit[j+1], max_sell-prices[
21                   }
22
23                   int max_profit = 0;
24                   for(int i=0; i<n;i++){
25                           max_profit = max(max_profit, pre_profit[i] + post_profi
26                   }
```

```
27                      return max_profit;
28              }
29      };
```

动态规划法 。以第i 天为分界线，计算第i 天之前进行一次交易的最大收益preProfit[i]，和第 i 天之后进行一次交易的最大收益 postProfit[i]，最后遍历一遍，max{preProfit[i] + postProfit[i]} (0≤i≤n-1)就是最大收益。