1、给定一个函数：

F(0)=0;

F(1)=1;

F(n)=F(n-1)+F(n-2),n>1

程序输入一个整数 n，0 < n<100000,输入 F(n)的值，结果对 854562545 取余。

解决思路

可以用递归的方法进行迭代相加，但由于数据比较大，用递归调用函数栈会消耗大量时间，所以用循环迭代相加的方式代替递归。

源代码：

```
#include<stdio.h>
int main()
{
    long i,n,f1,f,f0;
    f0=0;
    f1=1;
    scanf("%ld",&n);
    if(n==0)
    {
        printf("0\n");
        return 0;
    }
    if(n==1)
    {
        printf("1\n");
        return 0;
    }
    for(i=2;i<=n;i++)
    {
        f=(f0+f1)%854562545;
        f0=f1;
        f1=f;
    }
    printf("%ld\n",f);
    return 0;
}
```


2、问题描述

定义一个单调栈：每次整数 n 入栈时，如果栈顶元素大于 n，则栈顶元素出 栈，并且继续判断栈顶元素是否大于 n，大于则出栈，重复操作，直到栈顶 元素不大于 n，n 入栈。入栈完毕。例如:栈中元素为 2 3 7，如栈元素为 6， 则 7 出栈，6 入栈，最后结果为 2 3 6；

输入输出

第一行输入一个整数 0<n<100000，表示待入栈的元素序列

第二行输入 n 个待入栈的数
输出所有元素入栈后，栈的元素
如：
输入：
3
5 1 2
输出：
1 2
解决思路
类似于插入排序，边输入边入栈，用数组维护栈。
源代码：

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
	int n,*a,i,m;
	int l;
	scanf("%d",&n);
	a=(int *)malloc(sizeof(int)*n);
	l=0;
	while(n--)
	{

		scanf("%d",&m);
		while(1)
		{

			if(l==0)
			{
				a[l]=m;
				l++;
				break;
			}
			if(a[l-1]>m)
				l--;
			if(a[l-1]<=m)
			{
				a[l]=m;
				l++;
				break;
			}

		}
```

```
    }
    for(i=0;i<l-1;i++)
        printf("%d ",a[i]);
    printf("%d\n",a[l-1]);
    return 0;
}
```

3、问题描述

输入两个整数 n，m，(0 < n,m<100000)每一次，第一个整数可以执行乘 2,、减 1、加 1 三种操 作的任意一种，求 n 到 m 至少要多少次这样的操作。

解决思路

每次可以进行三个操作的其中一个，问题可以转化成单源最短路径问题：

每次可以走三个方向，求最短出口。

用队列实现 bfs，能比较完善的解决此类问题，但是由于队列需要储存出口之前的所以路径，空间冗余度较大。

源代码：

```
#include<stdlib.h>
#include<stdio.h>
struct list//用结构体表示每一个节点，用队列来实施 bfs
{
    long n;
    int step;//表示步数
    struct list *next;
};
int main()
{
    long n,m,k;
    struct list *p,*p1,*p2,*p3,*pm;
    scanf("%ld %ld",&n,&m);
    p=(struct list *)malloc(sizeof(struct list ));
    p->n=n;
    p->step=1;
    p->next=NULL;
    pm=p;
    while(p!=NULL)//每一步-1，+1，*2 都保存在链表维护的队列中
    {
        k=p->n;//每一次去队列的队头
        if(k==m)
            break;
        p1=(struct list *)malloc(sizeof(struct list ));
        p1->n=k-1;
        p1->step=p->step+1;
        p1->next=NULL;
```

```
            pm->next=p1;
            p2=(struct list *)malloc(sizeof(struct list ));
            p2->n=k+1;
            p2->step=p->step+1;
            p2->next=NULL;
            p1->next=p2;
            p3=(struct list *)malloc(sizeof(struct list ));
            p3->n=k*2;
            p3->step=p->step+1;
            p3->next=NULL;
            p2->next=p3;
            pm=p3;
            p1=p;
            p=p->next;

            free(p1);

        }
        printf("%d\n",p->step-1);
        return 0;
}
```