算法基础 - HW5

牛庆源 PB21111733

T1

```
#define N 100000    // x数组的大小上限
int x[N];   // x数组
int delete[N];  // 某个位置是否可以删除，可以删除置1
int maxn;   // 在main中初始化为数组大小
int del_able_num = 0;   // 可以删除元素的个数
int ans = 0;    // ans

int check(int nowpos)
{   // 判断当前位置是否可以删除(nowpos是当前位置)
    if((nowpos - 1 >= 0 && x[nowpos] - x[nowpos - 1] == 1) ||
    (nowpos + 1 < maxn && x[nowpos] - x[nowpos + 1] == 1))
    {
        delete[nowpos] = 1;
        del_able_num ++;
    }
}

int del(int pos)
{   // 删除元素
    for(int i = pos; i < maxn; i ++)
        x[pos] = x[pos + 1];

    maxn --;
}

int main()
{
    scanf("%d", &maxn);
    for(int i = 0; i < maxn; i ++)
        scanf("%d", &x[i]);

    for(int i = 0; i < maxn; i ++)
        check(i);   // 初始化delete数组和可以删除元素个数

    int max_num = 0;    // 最大的可删除元素
    int max_num_i = 0;  // 最大可删除元素的位置
```

```
    while(del_able_num != 0)
    {
        for(int i = 0; i < maxn; i ++)
        {
            if(delete[i] == 1 && max_num < x[i])
            {
                max_num = x[i];
                max_num_i = i;
            }
        }

        del(max_num_i);
        ans ++;

        for(int i = 0; i < maxn; i ++)
            check(i);
    }

    printf("%d", ans);


    del_able_num O(n)
    {
        O(n)
    }
}
```

$O(n^2)$

T2

(1) (2)

- 显然每天去到新的宿舍会使得收缴的电脑更多，即不重复去同一个宿舍，一共k天,去k个宿舍。
  (如果重复的话就会导致少收缴一个宿舍初始的电脑数量)
- 在这k天里，一共收缴的电脑分为两部分：
  一部分为本来这k间宿舍的初始电脑总数；
  另一部分为这k天这k间宿舍一共增加的电脑总数$\frac{(0+k-1)*k}{2} = \frac{(k-1)k}{2}$
- 即先从宿舍0到宿舍k - 1，计算出从左到右查，这k间宿舍的总电脑数。
  然后查宿舍的起点从宿舍1遍历到宿舍n-k，即查宿舍的终点从宿舍k遍历到宿舍n - 1（一共n个宿舍，数组下标为从0到n-1），如果有某一次的总电脑数大于先前查到的最大总电脑数，则更新。
  注意这一步不是从[1 , k]到[n - k , n - 1]这n - k个区间都遍历一次，而是每一次都减去上一次的首项并且加上这一次的末项，这样避免了遍历(O(n-k))导致的时间复杂度增加，在这些区间上的时间复杂度为O(1)

- 最后结果即为所求。

```
// 假设宿舍数n，查寝天数k，宿舍i的初始电脑数量为x[i]

int main()
{
    int temp_tot = 0;
    for(int i = 0; i < k; i ++)
        temp_tot += x[i];

    int max_tot = temp_tot;
    for(int i = 1; i < n - k + 1; i ++)
    {
        temp_tot -= x[i - 1];
        temp_tot += x[i + k - 1];
        if(max_tot < temp_tot)
            max_tot = temp_tot;
    }
    int ans = max_tot + ((k - 1) * k / 2);

    printf("%d", ans);
}
```