Homework #3
Solution Manual

1 Slime King

```
#include <stdio.h>
#include <stdlib.h>
#define N 100010
long long l[N],s[N],ans1;
int cmp( const void *a, const void *b ) {
        return *(long long*)b - *(long long*)a;
}
int main()
{
        int n,q,k;
        scanf("%d%d",&n,&q);
        for ( int i=1; i<=n; i++ ) scanf("%lld",l+i);
        qsort(l+1,n,sizeof(long long),cmp);
        for ( int i=1; i<=n; i++ ) s[i]=s[i-1]+l[i];
        for ( int i=1; i<n; i++ ) ans1+=i*l[i+1];
        while ( q-- ) {
                scanf("%d",&k);
                if ( k==1 ) {
                        printf("%lld\n",ans1);
                        continue;
                }
                long long ans=0,x=1,p=1;
                for ( int i=1; x<n; i++ ) {
                        p*=k;
                        long long y=x+p;
                        if ( y>n ) y=n;
                        ans+=i*(s[y]-s[x]);
                        x=y;
                }
                printf("%lld\n",ans);
        }
        return 0;
}
```

Note that the calculation of cost can be done separately, where each slime contribute the product of its level and how many times it be eaten to be the king. We can represent the eating relation in a k-ary tree, since every slime will be eaten exact once except the final slime king, which will be the root of tree, and u is a child of v iff u is eaten by v. If we define the depth of root is 0, the final cost will be the summation of each node's level times its depth. There must be an optimal solution, where the tree is complete and the level is sorted from root to leaf, otherwise we can swap any nodes conflict this rule and obtain a solution which would not be worser. If $k = 1$, the tree degenerate to a simple chain, where we can pre-calculate its cost in linear time, and just output the answer for any $k = 1$ query in $O(1)$. If $k > 1$, the depth of a complete k-ary tree will be $O(\lg N)$. Sort the slimes by its level, same depth will form an interval in this array. We can obtain the sum of this interval in $O(1)$ with a pre-calculated partial sum array, which just cost $O(N)$ time. Combine the above techniques, result a $((N + Q) \lg N)$ algorithm.

2 Gift

1) The previous step to (x, y) must be $(x-1, y)$ or $(x, y-1)$, thus there is a simple recursion $C(x, y) = C(x-1, y) + C(x, y-1)$. We can use a simple dynamic programming algorithm to solve this in $O(N^2)$ time.

2) Let $f(x, y, s)$, where $1 \leq x, y \leq N$, and $s \in \{0, 1\}^3$, represents the number of way to reach (x, y) and s denote that whether he had visited flower shop, bookstore, candy shop or not. This is the entire information we need when he is at position (x, y), since we do not care about which route was taken, where he buy the gift or anything else. Obviously, there is a recursive relation on $f(x, y, s)$ and $f(x-1, y, s_1), f(x, y-1, s_2)$, and what store is in (x, y). so we can build an $O(N^2)$ dynamic algorithm based on above recursion.

3 Go Go Go Greedy Girl!

1) Correct. If there is any cross pair in optimal solution such that a_x connected to b'_y, a'_x connected to b_y, and $x < y$, $x' < y'$. We can show that swap the connection to (a_x, b_y) and (a'_x, a'_y) would not obtain a worser solution by enumerate the possible six relative position of them. Since the number of cross pairs would be finite, this solution will as same as the greedy solution, and would not be worser.

2) Incorrect. Let $p = [1, 3, 4, 2]$, the greedy solution yield $3 \times 4 \times 2 + 1 \times 3 \times 2 = 30$, but the optimal solution should be $1 \times 3 \times 4 + 1 \times 4 \times 2 = 20$.

3) Correct. We can use mathematical induction on n to show that.
Basis: When $n = 1$, It is easy to see that just pay the most expensive item in that day by coupon is optimal. Induction: Assume that for any $n' < n$ the greedy solution will be optimal.
Consider the coupon issued on last day, we can always use it to pay for the most expensive item in last day. And after that, we can merge the last two days into one day, and apply the induction hypothesis.

4) Incorrect. Let the length of strings as $[3, 2, 2, 3]$, the greedy solution yield $(2+2)+(3+4)+(7+3) = 21$, but the optimal solution should be $(3+2)+(3+2)+(5+5) = 20$.

4 Coin Changing

1) We can use mathematical induction on n to prove the general case, thus the special case for $n = 4$ and $a = [1, 5, 10, 50]$ will be hold as well.
Basis: When $n = 1$, there is only one way to make changes for m dollors, so it is true. Induction: Assume the for any $n' < n$ the greedy solution will be optimal. Consider the denomination a_n, let the number of a_n used in the optimal solution be u. If $u = \lfloor m/a_n \rfloor$, the optimal solution use same number of a_n as greedy solution, and we can use greedy as a optiaml way to make changes for $m - a_n \times \lfloor m/a_n \rfloor$ with the the remaining $a_1, a_2, \ldots, a_{n-1}$ by induction htpothesis. If $u < \lfloor m/a_n \rfloor$, there is a optimal solution using u on a_n and $v = \lfloor (m - u \times a_n)/a_{n-1} \rfloor$ on a_{n-1} by induction hypothesis. But we have a better solution using $u+1$ on a_n, $v - a_n/a_{n-1}$ on a_{n-1}, a contradiction. Note that this solution is valid since $u < \lfloor m/a_n \rfloor \implies m - u \times a_n \geq a_n \implies v \geq a_n/a_{n-1}$.

2) When $a = [1, 3, 4]$ and $m = 6$, the greedy algorithm yields $6 = 4 + 1 + 1$ with 3 coins, but the optimal solution should be $6 = 3 + 3$ with only 2 coins.

3) Let $f(i,j)$ be the optimal way to make changes for j dollors with the denomination $a_1, a_2, \ldots, a_i$, consider whether a_i is used in optimal solution will obtain the recursion

$$f(i,j) = \begin{cases} 0 & \text{if } j = 0; \\ j & \text{if } i = 1; \\ f(i-1,j) & \text{if } a_i > j; \\ \min(f(i-1,j), f(i,j-a_i)+1) & \text{otherwise.} \end{cases}$$

So $f(n,m)$ is the answer we want. Use the dynamic programming algorithm can fill the table of f and store the decision in $O(nm)$ time.

4) **"if"**: Let o_1, o_2, o_3 be the amount of a_1, a_2, a_3 used in optimal solution which uses minimum a_2. If $o_2 \geq r$, we can replace r of a_2 by one a_3 and some a_1, since we know that greedy works for $M = r \times a_2$, but a less o_2 optimal solution, which contradicts to the minimum assumption. So $o_2 < r \implies o_2 \times a_2 < a_3$ since $r = \lceil a3/a2 \rceil$. If $o_2 \times a_2 + o_1 \geq a_3$, we can replace all a_2 and some a_1 by one a_3, but a less o_2 optimal solution, which contradicts to the minimum assumption again. So $o_2 \times a_2 + o_1 < a_3 \implies o_3 = \lfloor m/a_3 \rfloor$, since $m = o_3 \times a_3 + (o_2 \times a_2 + o_1)$ is the form as theorem of division. So this optimal solution uses same number of a_3 as the greedy solution, and the remaining part should also be made greedily, since $a_1 = 1$. So this optimal solution is totally equivalent to the greedy solution.

"only if": M is a possible value of "every m", so greedy dose not work for M implies it does not work for every m.