Lab7 Solutions

Note: ในเฉลยของแลปนี้จะใช้ upper_bound หรือ lower_bound ของ Stand Template Library(STL) แทนการ implement binary search เอง จะถือว่าเข้าใจหลักการเขียน binary search โดยใช้ loop หรือ recursion มาแล้วนิดนึง และโค้ดตัวอย่างจะได้ไม่ยาวเกินไป ทั้งสองคำสั่งจะทำงานกับ array หรือ container ที่ sort เรียบร้อยแล้ว ลักษณะคำสั่งคร่าวๆ เป็นประมาณนี้

- upper_bound(arr,arr+size,n) หาตำแหน่งแรกใน arr ที่มีค่ามากกว่า n
- lower_bound(arr,arr+size,n) หาตำแหน่งแรกใน arr ที่มีค่าไม่น้อยกว่า n

ศึกษาเพิ่มเติมได้ที่:

- https://cplusplus.com/reference/algorithm/upper_bound/
- https://cplusplus.com/reference/algorithm/lower_bound/

Count

เนื่องจากข้อมูลมีจำนวนไม่มาก ดังนั้นสามารถวน loop ทั้ง array เพื่อหาว่ามี k กี่ตัวแบบตรงไปตรงมาได้เลย(Linear Search)

Example Code

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n, k, count = 0;
    cin >> n;
    int arr[n];
    for(int i=0;i<n;i++) cin >> arr[i];
    cin >> k;
    for(int i=0;i<n;i++) if(arr[i]==k) count++;
    cout << count;
}
```

$time\ complexity : O(n)$

Dungeon

ในข้อนี้จำนวนมอนสเตอร์และจำนวนอาวุธค่อนข้างเยอะ การคิดแบบตรงไปตรงมาเหมือนข้อก่อนหน้าอาจจะทำให้ใช้เวลามากเกินไป สามารถลดเวลาได้โดยการ sort array ของเลือดมอนสเตอร์จากน้อยไปมากแล้ว binary search เพื่อหาว่าตำแหน่งแรกที่เลือดมอนสเตอร์มากกว่าดาเมจของอาวุธชิ้นหนึ่งอยู่ที่ตำแหน่งไหน แล้วทางซ้ายทั้งหมดคือมอนสเตอร์ที่สามารถสังหารได้ในเพียงการโจมตีเดียวของอาวุธชิ้นนั้น

ปล. เนื่องจาก *function upper_bound* ถ้าทำงานกับ *array* จะ *return* ค่าเป็น *address* เลยสามารถนำไปลบกับ *arr* ซึ่งมองว่าเป็น *address* ได้เหมือนกันเพื่อหา *index* ของจำนวนที่ต้องการได้

Example Code

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n,q,x;
    cin >> n;
    int arr[n];
    for(int i=0;i<n;i++) cin >> arr[i];
    sort(arr,arr+n);
    cin >> q;
    while(q--)
    {
        cin >> x;
        cout << upper_bound(arr,arr+n,x)-arr << endl;
    }
}
```

$time\ complexity : O(q \cdot log(n))$

Satorbarer

เริ่มจากการหาว่าสามารถใช้กล่องขนาดใดได้บ้าง โดยหา ห.ร.ม. ของผลผลิตในแต่ละฟาร์ม ขนาดกล่องที่เป็นไปได้ทั้งหมดคือตัวประกอบทั้งหมดของ ห.ร.ม. เนื่องจาก ห.ร.ม. หารทุกจำนวนลงตัวเลยใช้เป็นขนาดกล่องหนึ่งได้ ดังนั้นตัวประกอบของของ ห.ร.ม. ก็ต้องหารทุกจำนวนลงตัวเช่นกัน

ในพิจารณาว่าแต่ละข้อเสนอควรใช้กล่องขนาดใด ให้หากล่องที่มีขนาดมากที่สุดที่ไม่เกิน h_j โดยใช้ binary search และขนาดกล่องนั้นต้องไม่น้อยกว่า l_j ถ้าหากน้อยกว่า l_j หมายความว่าไม่สามารถใช้กล่องขนาดใดได้เลย และต้องปฏิเสธข้อเสนอ

ปล. upper_bound() จะ return ตำแหน่งแรกที่มีค่ามากกว่า ต้องถอยมา 1 ตำแหน่ง(--upper_bound()) ถึงจะได้ตำแหน่งสุดท้ายที่มีค่าไม่เกิน แล้วค่อย dereference เพื่อเข้าถึงค่าในตำแหน่งนั้น

ปล2. ในโค้ดใช้ set<int> เพื่อเก็บตัวประกอบ ลักษณะคล้ายๆ array แต่จะไม่มีค่าซ้ำใน set นั้น เพื่อป้องกันกรณีที่ ห.ร.ม เป็นจำนวนกำลังสอง(เช่น 9 เวลาวนลูปจะได้ 3*3 เป็นตัวประกอบซ้ำกันสองตัว)

Example Code

```cpp
#include <bits/stdc++.h>
using namespace std;

int gcd(int a,int b) { return (b==0)? a:gcd(b,a%b); }

int main()
{
    int n,k;
    long long int ans=0;
    cin >> n;
    long long int a[n];
    for(int i=0;i<n;i++) cin >> a[i];

    int x = a[0];
    for(int i=0;i<n;i++) x = gcd(x,a[i]);
    set<int> factors;
    for(int i=1;i<=sqrt(x);i++)
    {
        if(x%i==0)
        {
            factors.insert(i);
            factors.insert(x/i);
        }
    }
```

```
    cin >> k;
    while(k--)
    {
        int l,r,p;
        long long int sum=0;
        cin >> l >> r >> p;

        // Maximum number not exceed r -> prev of upper bound
        int size = *(--factors.upper_bound(r));

        // size requirement not satisfied -> skip
        if(size<l) continue;

        for(int i=0;i<n;i++) sum += a[i]/size*p;
        ans = max(ans,sum);
    }

    cout << ans;
}
```

$time\ complexity : O(\mathrm{k})$