Candy Factory

Problem setter: Alex Poon

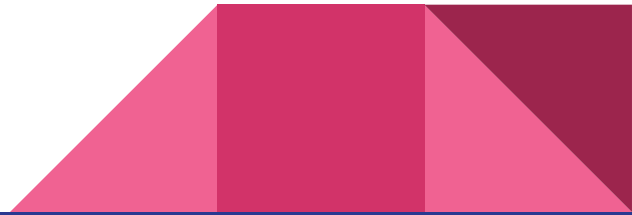
Problem statement

There are N robots and each robot has a power P[i] and stemina S[i]

The number of candy a robot can make in x hour = P[i] + S[i] * x for x > 0
The number of candy a robot can make in 0 hour = 0

You can hire some robots for different time
The sum of working hour of all robots must not exceed M

What is the maximum number of candy they can produce?

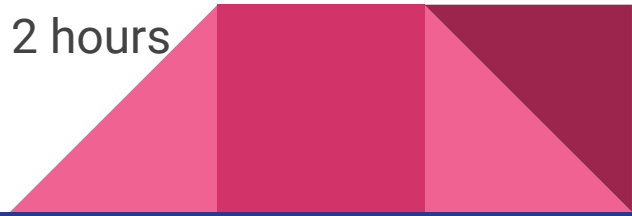


Sample

Input
3 6 (N, M)
2 2 (P[i], S[i])
3 1
4 2

Output
20

** hire robot 1 for 3 hours, robot 2 for 1 hours, robot 3 for 2 hours
sum = 2 + 3 * 2 + 3 + 1 * 1 + 4 + 2 * 2 = 8 + 4 + 8 = 20



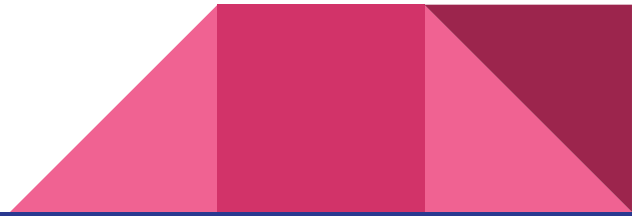
Subtask

Subtask	Points	N	M	P[i], S[i]
1	9	1 <= N <= 1000	M = 1	1 <= P[i], S[i] <= 1000
2	14	1 <= N <= 1000	M = 2	1 <= P[i], S[i] <= 1000
3	12	1 <= N <= 100000	M = 2	1 <= P[i], S[i] <= 100000
4	23	1 <= N <= 1000	1 <= M <= 1000	1 <= P[i], S[i] <= 1000
5	13	1 <= N <= 1000	1 <= M <= 10^9	1 <= P[i], S[i] <= 1000
6	29	1 <= N <= 100000	1 <= M <= 10^9	1 <= P[i], S[i] <= 100000

Statistic

Attempts: 63
Attempt Mean: 27.65
Attempt std dev: 31.393

100 Points: 8
58 Points: 4
35 Points: 9
23 Points: 7
9 Points: 26

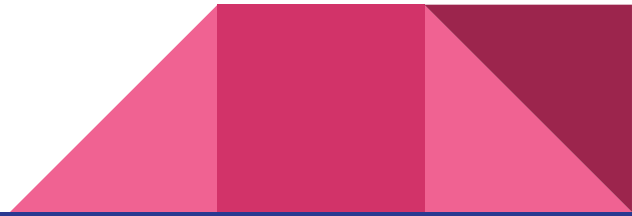


Subtask 1

Note that M = 1

Algorithm : Exhaustion and then find the maximum

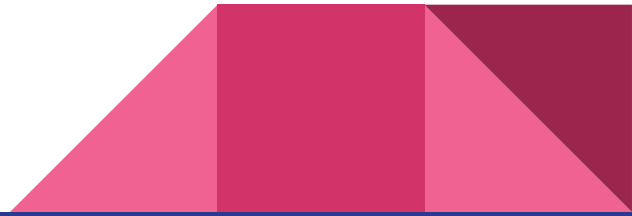
Time complexity : O(N)



Subtask 2 - 3

Note that M = 2

There are two cases
1. Hire one robot for 2 hours
2. Hire two robot for 1 hours



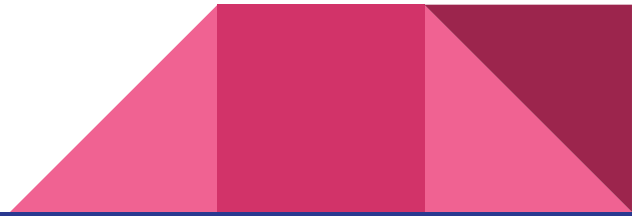
Subtask 2 - 3

For case 1, we exhaust which to buy and find the maximum just like subtask 1
Time complexity : O(N)

For case 2 :
If we exhaust all possible combination and find the maximum, we need O(N * N)

Actually, the only thing we need to do is to find the maximum and second maximum of P[i] + S[i] for 1 <= i <= n

Which can be done in O(N)



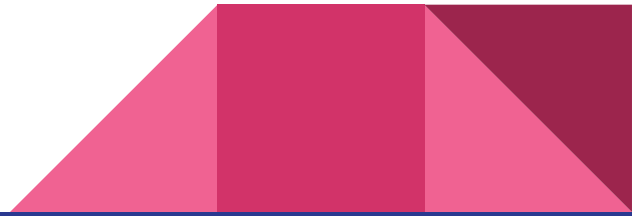
Problem Analysis

For subtask 4 onward, we need to analyze the problem again

** We can hire a robot for x hours (0 <= x <= M)
** The candy a robot can produce within x hours = P[i] + x * S[i]

These rules can be interpret as :

- We can hire a robot for 1 hour only, but we can hire it several time
- The first time we hire it, it can produce P[i] + S[i] candy
- The second time onward, it can produce S[i] candy



Problem Analysis

Sample Input
3 6 (N, M)
2 2 (P[i], S[i])
3 1
4 2

Samole Output
20

Hire robot 1 for 3 hours, robot 2 for 1 hours, robot 3 for 2 hours

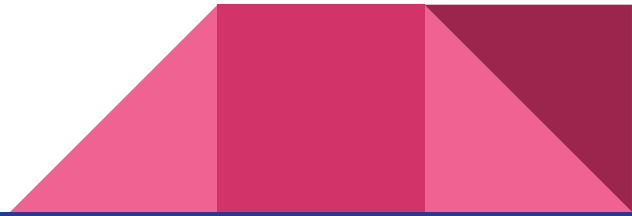
It can be interpret as: (* denote the first time)

Hire robot 1 for 1 hour* -> 2 + 2 = 4 candy it produces
Hire robot 1 for 1 hour -> 2 candy it produces
Hire robot 1 for 1 hour -> 2 candy it produces
Hire robot 2 for 1 hour* -> 3 + 1 = 4 candy it produces
Hire robot 3 for 1 hour* -> 4 + 2 = 6 candy it produces
Hire robot 3 for 1 hour -> 2 candy it produces

4 + 2 + 2 + 4 + 6 + 2 = 20

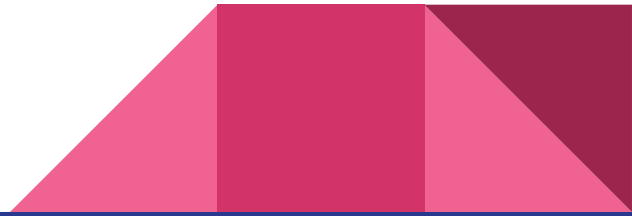
Problem Analysis

So, the question becomes:
For every dollars, we should hire which robot for 1 hour
Instead of we should hire each robot for how many hours



Observation 1

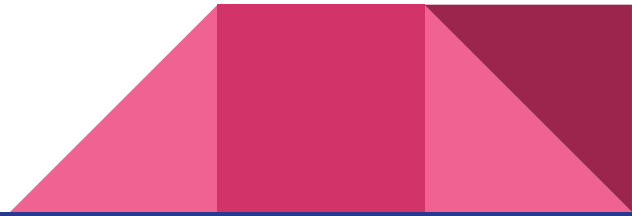
An optimal strategy:
Hire the robot which can produce the maximum number of candy every time



Subtask 1 - 4

Algorithm:
For every dollar, we hire the robot which can produce the max number of candy.
If it is the first time we hire the robot, update the number of candy it can produce

Time complexity:
We have M dollar and we need O(N) to find the maximum each time: O(M * N)

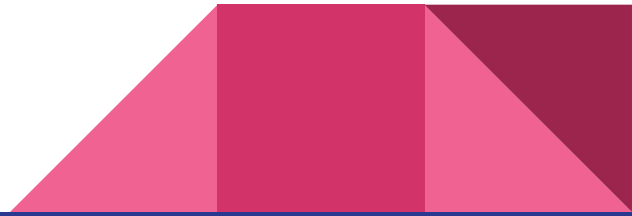


Observation 2

Base on subtask 4

If we hire a robot i for the second time in current round.
It represents the productivity of robot i (which is S[i]) is maximum among all
And we know the productivity of robot i remains S[i] in future rounds

Therefore, we will continue hire robot i for every future rounds



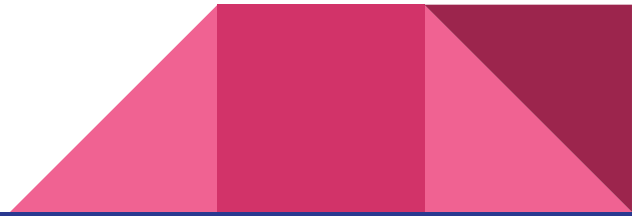
Subtask 1 - 5

So, using the algorithm in subtask 4

If we find that “the robot we hire this round” has been hired before

We can stop looping as we know for remaining dollars, we will hire the same robot

So time complexity becomes: O(N * N)



Full Solution

Base on observation 2: “We will continue hire robot i for every future rounds”

We can conclude that there is at most 1 robot we will hire for more than 1 hour, which is the robot having the highest ***stemina*** (let denote it as ***robot i***)

So we only need to find which robots we will hire (hire exactly 1 hours) before hiring ***robot i*** for the 2nd time

Full Solution

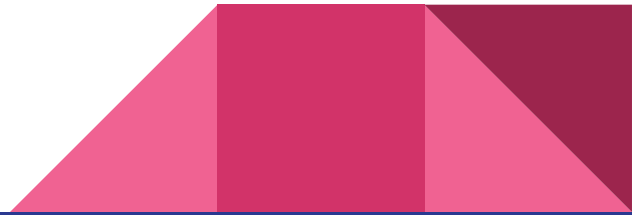
This time we only need to consider hire a robot for exactly 1 hour

So we can sort the robot by their productivity in the first hour (P[j] + S[j]) by counting sort

And hire the robot sequencially from the highest productivity to the lowest until

1. If we have comsume all our M dollars
2. If the productivity of the robot is smaller that S[i], which mean we will hire robot i using all of the the remaining dollars

Time Complexity: O(N)



Appendix - Counting Sort

We use an array to record the frequency of each value, and then sort it

e.g. The sequence we need to sort is {1, 8, 8, 3, 6, 2, 3, 7}

We have an array F, F[i] represent the frequency of i
i.e. F[1..8] = {1, 1, 2, 0, 0, 1, 1, 2}

We scan array F from 1 to 8, and insert i for F[i] time in another array

Then we sort it !!