An Introduction to Greedy Algorithms

Ruhan Habib, Mubasshir Chowdhury

April 20, 2025

BRAC University

Be Greedy?

- Don’t be greedy?
- Sometimes, greed is good (in Computer Science, at least).
- What is a greedy algorithm?
- An algorithm where we always make “greedy” choices.
- Plenty of problems have simple (easy-to-code) greedy solutions.

Be Greedy?

- Don't be greedy?
- Sometimes, greed is good (in Computer Science, at least).
- What is a greedy algorithm?
- An algorithm where we always make "greedy" choices.
- Plenty of problems have simple (easy-to-code) greedy solutions.

Be Greedy?

- Don't be greedy?
- Sometimes, greed is good (in Computer Science, at least).
- What is a greedy algorithm?
- An algorithm where we always make "greedy" choices.
- Plenty of problems have simple (easy-to-code) greedy solutions.

Be Greedy?

- Don't be greedy?
- Sometimes, greed is good (in Computer Science, at least).
- What is a greedy algorithm?
- An algorithm where we always make "greedy" choices.
- Plenty of problems have simple (easy-to-code) greedy solutions.

Be Greedy?

- Don't be greedy?
- Sometimes, greed is good (in Computer Science, at least).
- What is a greedy algorithm?
- An algorithm where we always make "greedy" choices.
- Plenty of problems have simple (easy-to-code) greedy solutions.

Resource

This lecture is based on/copied from the YouTube video "Episode 14 - Exchange Arguments" by AlgorithmsLive: https://www.youtube.com/live/Oq1seKJvfQU?si=iRfuVzIP87o7mcgv

Maximize Product

Input

We are given a positive integer n ($2 \leq n \leq 10^{18}$).

Output

Find the maximum value of ab over all positive integers a and b such that $a + b = n$.

Sample Input

4

Sample Output

4

Maximize Product

Input

We are given a positive integer n $(2 \leq n \leq 10^{18})$.

Output

Find the maximum value of ab over all positive integers a and b such that $a + b = n$.

Sample Input

4

Sample Output

4

Maximize Product

Input

We are given a positive integer n ($2 \leq n \leq 10^{18}$).

Output

Find the maximum value of ab over all positive integers a and b such that $a + b = n$.

Sample Input

4

Sample Output

4

Maximize Product

Input

We are given a positive integer n ($2 \leq n \leq 10^{18}$).

Output

Find the maximum value of ab over all positive integers a and b such that $a + b = n$.

Sample Input

4

Sample Output

4

Maximize Product

Maximize Product

Algorithm 1 MaximizeProduct(n)

Set $a \leftarrow \left\lfloor \frac{n}{2} \right\rfloor$.
Set $b \leftarrow \left\lceil \frac{n}{2} \right\rceil$.
return ab.

Theorem

Assume that n is an even positive integer. Then $a = b = \frac{n}{2}$ is optimal.

Proof.

Let $x = \frac{n}{2}$. Suppose, on the contrary, that $a = b = x = \frac{n}{2}$ is not optimal. Let a_*, b_* be an optimal choice for a and b. Then $a_* + b_* = n$ and $a_* b_* > x^2$. Let $\Delta = x - a_*$. So $a_* = x - \Delta$. Since $a_* + b_* = n = 2x$, we have $b_* = x + \Delta$. Then, $a_* b_* = (x - \Delta)(x + \Delta) = x^2 - \Delta^2 \leq x^2$. But we said that $a_* b_* > x^2$. So our assumption that $a = b = x$ is unoptimal, is wrong. Indeed, $a = b = x$ is optimal. $\square$

Maximize Product

Theorem

Assume that n is an even positive integer. Then $a = b = \frac{n}{2}$ is optimal.

Proof.

Let $x = \frac{n}{2}$. Suppose, on the contrary, that $a = b = x = \frac{n}{2}$ is not optimal. Let a_*, b_* be an optimal choice for a and b. Then $a_* + b_* = n$ and $a_* b_* > x^2$. Let $\Delta = x - a_*$. So $a_* = x - \Delta$. Since $a_* + b_* = n = 2x$, we have $b_* = x + \Delta$. Then, $a_* b_* = (x - \Delta)(x + \Delta) = x^2 - \Delta^2 \leq x^2$. But we said that $a_* b_* > x^2$. So our assumption that $a = b = x$ is unoptimal, is wrong. Indeed, $a = b = x$ is optimal. □

Maximize Product

Theorem

Assume that n is an even positive integer. Then $a = b = \frac{n}{2}$ is optimal.

Proof.

Let $x = \frac{n}{2}$. Suppose, on the contrary, that $a = b = x = \frac{n}{2}$ is not optimal. Let a_*, b_* be an optimal choice for a and b. Then $a_* + b_* = n$ and $a_* b_* > x^2$. Let $\Delta = x - a_*$. So $a_* = x - \Delta$. Since $a_* + b_* = n = 2x$, we have $b_* = x + \Delta$. Then, $a_* b_* = (x - \Delta)(x + \Delta) = x^2 - \Delta^2 \leq x^2$. But we said that $a_* b_* > x^2$. So our assumption that $a = b = x$ is unoptimal, is wrong. Indeed, $a = b = x$ is optimal. □

Maximize Product

Theorem

Assume that n is an even positive integer. Then $a = b = \frac{n}{2}$ is optimal.

Proof.

Let $x = \frac{n}{2}$. Suppose, on the contrary, that $a = b = x = \frac{n}{2}$ is not optimal. Let a_*, b_* be an optimal choice for a and b. Then $a_* + b_* = n$ and $a_* b_* > x^2$. Let $\Delta = x - a_*$. So $a_* = x - \Delta$. Since $a_* + b_* = n = 2x$, we have $b_* = x + \Delta$. Then, $a_* b_* = (x - \Delta)(x + \Delta) = x^2 - \Delta^2 \leq x^2$. But we said that $a_* b_* > x^2$. So our assumption that $a = b = x$ is unoptimal, is wrong. Indeed, $a = b = x$ is optimal. $\square$

Maximize Product

Theorem

Assume that n is an even positive integer. Then $a = b = \frac{n}{2}$ is optimal.

Proof.

Let $x = \frac{n}{2}$. Suppose, on the contrary, that $a = b = x = \frac{n}{2}$ is not optimal. Let a_*, b_* be an optimal choice for a and b. Then $a_* + b_* = n$ and $a_* b_* > x^2$. Let $\Delta = x - a_*$. So $a_* = x - \Delta$. Since $a_* + b_* = n = 2x$, we have $b_* = x + \Delta$. Then, $a_* b_* = (x - \Delta)(x + \Delta) = x^2 - \Delta^2 \leq x^2$. But we said that $a_* b_* > x^2$. So our assumption that $a = b = x$ is unoptimal, is wrong. Indeed, $a = b = x$ is optimal. □

Maximize Product

Theorem

Assume that n is an even positive integer. Then $a = b = \frac{n}{2}$ is optimal.

Proof.

Let $x = \frac{n}{2}$. Suppose, on the contrary, that $a = b = x = \frac{n}{2}$ is not optimal. Let a_*, b_* be an optimal choice for a and b. Then $a_* + b_* = n$ and $a_* b_* > x^2$. Let $\Delta = x - a_*$. So $a_* = x - \Delta$. Since $a_* + b_* = n = 2x$, we have $b_* = x + \Delta$. Then, $a_* b_* = (x - \Delta)(x + \Delta) = x^2 - \Delta^2 \leq x^2$. But we said that $a_* b_* > x^2$. So our assumption that $a = b = x$ is unoptimal, is wrong. Indeed, $a = b = x$ is optimal. $\square$

Maximize Product

Theorem

Assume that n is an even positive integer. Then $a = b = \frac{n}{2}$ is optimal.

Proof.

Let $x = \frac{n}{2}$. Suppose, on the contrary, that $a = b = x = \frac{n}{2}$ is not optimal. Let a_*, b_* be an optimal choice for a and b. Then $a_* + b_* = n$ and $a_* b_* > x^2$. Let $\Delta = x - a_*$. So $a_* = x - \Delta$. Since $a_* + b_* = n = 2x$, we have $b_* = x + \Delta$. Then, $a_* b_* = (x - \Delta)(x + \Delta) = x^2 - \Delta^2 \leq x^2$. But we said that $a_* b_* > x^2$. So our assumption that $a = b = x$ is unoptimal, is wrong. Indeed, $a = b = x$ is optimal. $\square$

Maximize Product

Theorem

Assume that n is an even positive integer. Then $a = b = \frac{n}{2}$ is optimal.

Proof.

Let $x = \frac{n}{2}$. Suppose, on the contrary, that $a = b = x = \frac{n}{2}$ is not optimal. Let a_*, b_* be an optimal choice for a and b. Then $a_* + b_* = n$ and $a_* b_* > x^2$. Let $\Delta = x - a_*$. So $a_* = x - \Delta$. Since $a_* + b_* = n = 2x$, we have $b_* = x + \Delta$. Then, $a_* b_* = (x - \Delta)(x + \Delta) = x^2 - \Delta^2 \leq x^2$. But we said that $a_* b_* > x^2$. So our assumption that $a = b = x$ is unoptimal, is wrong. Indeed, $a = b = x$ is optimal. $\square$

Maximize Product

Theorem

Assume that n is an even positive integer. Then $a = b = \frac{n}{2}$ is optimal.

Proof.

Let $x = \frac{n}{2}$. Suppose, on the contrary, that $a = b = x = \frac{n}{2}$ is not optimal. Let a_*, b_* be an optimal choice for a and b. Then $a_* + b_* = n$ and $a_* b_* > x^2$. Let $\Delta = x - a_*$. So $a_* = x - \Delta$. Since $a_* + b_* = n = 2x$, we have $b_* = x + \Delta$. Then, $a_* b_* = (x - \Delta)(x + \Delta) = x^2 - \Delta^2 \leq x^2$. But we said that $a_* b_* > x^2$. So our assumption that $a = b = x$ is unoptimal, is wrong. Indeed, $a = b = x$ is optimal. $\square$

Maximize Product

This is the core idea behind proving greedy strategies: we show that any other choice is worse (or not better) compared to our greedy choice.

Coin Change

Input

A positive integer n ($1 \leq n \leq 10^{18}$).

Output

Find the minimum number of coins needed such that their value sums to n.

Sample Input

1000

Sample Output

40

Coin Change

Input

A positive integer n ($1 \leq n \leq 10^{18}$).

Output

Find the minimum number of coins needed such that their value sums to n.

Sample Input

1000

Sample Output

40

Coin Change

Input

A positive integer n ($1 \leq n \leq 10^{18}$).

Output

Find the minimum number of coins needed such that their value sums to n.

Sample Input

1000

Sample Output

40

Coin Change

Input

A positive integer n ($1 \leq n \leq 10^{18}$).

Output

Find the minimum number of coins needed such that their value sums to n.

Sample Input

1000

Sample Output

40

Coin Change

- The problem is ill-defined.
- What type of coins? Different countries have different denominations.
- The specific denomations matter!
- Bad for greedy: 1, 499, and 502.
- Good for greedy: 1, 5, 10, and 25.
- That is the denomination that we shall assume for this problem.
- How do we know that a greedy solution works?
- What is a correct greedy solution for this problem?

Coin Change

- The problem is ill-defined.
- What type of coins? Different countries have different denominations.
- The specific denomations matter!
- Bad for greedy: 1, 499, and 502.
- Good for greedy: 1, 5, 10, and 25.
- That is the denomination that we shall assume for this problem.
- How do we know that a greedy solution works?
- What is a correct greedy solution for this problem?

Coin Change

- The problem is ill-defined.
- What type of coins? Different countries have different denominations.
- The specific denomations matter!
- Bad for greedy: 1, 499, and 502.
- Good for greedy: 1, 5, 10, and 25.
- That is the denomination that we shall assume for this problem.
- How do we know that a greedy solution works?
- What is a correct greedy solution for this problem?

Coin Change

- The problem is ill-defined.
- What type of coins? Different countries have different denominations.
- The specific denomations matter!
- Bad for greedy: 1, 499, and 502.
- Good for greedy: 1, 5, 10, and 25.
- That is the denomination that we shall assume for this problem.
- How do we know that a greedy solution works?
- What is a correct greedy solution for this problem?

Coin Change

- The problem is ill-defined.
- What type of coins? Different countries have different denominations.
- The specific denomations matter!
- Bad for greedy: 1, 499, and 502.
- Good for greedy: 1, 5, 10, and 25.
- That is the denomination that we shall assume for this problem.
- How do we know that a greedy solution works?
- What is a correct greedy solution for this problem?

Coin Change

- The problem is ill-defined.
- What type of coins? Different countries have different denominations.
- The specific denomations matter!
- Bad for greedy: 1, 499, and 502.
- Good for greedy: 1, 5, 10, and 25.
- That is the denomination that we shall assume for this problem.
- How do we know that a greedy solution works?
- What is a correct greedy solution for this problem?

Coin Change

- The problem is ill-defined.
- What type of coins? Different countries have different denominations.
- The specific denomations matter!
- Bad for greedy: 1, 499, and 502.
- Good for greedy: 1, 5, 10, and 25.
- That is the denomination that we shall assume for this problem.
- How do we know that a greedy solution works?
- What is a correct greedy solution for this problem?

Coin Change

- The problem is ill-defined.
- What type of coins? Different countries have different denominations.
- The specific denomations matter!
- Bad for greedy: 1, 499, and 502.
- Good for greedy: 1, 5, 10, and 25.
- That is the denomination that we shall assume for this problem.
- How do we know that a greedy solution works?
- What is a correct greedy solution for this problem?

Coin Change

Coin Change

Algorithm 2 CoinChange(n)

```
Set Coins ← [25, 10, 5, 1].
Set counter ← 0.
Set unpayed ← n.
while unpayed ≥ 1 do
    Set j ← 1.                                  ▷ 1-based indexing.
    while Coins[j] > unpayed do
        Set j ← j + 1.
    end while
    Set counter ← counter + 1.
    Set unpayed ← unpayed − Coins[j].
end while
return counter.
```

Theorem

Let n be a positive integer. Let $\mathcal{O}$ denote any optimal multiset of coins that sum to n. Let $\mathcal{G}$ denote the multiset of coins that we get by the greedy solution for n. Then $|O| = |G|$. That is, the greedy solution uses the exact same number of coins as any optimal solution.

Coin Change

Theorem

Let n be a positive integer. Let $\mathcal{O}$ denote any optimal multiset of coins that sum to n. Let $\mathcal{G}$ denote the multiset of coins that we get by the greedy solution for n. Then $|O| = |G|$. That is, the greedy solution uses the exact same number of coins as any optimal solution.

Coin Change

Theorem

Let n be a positive integer. Let $\mathcal{O}$ denote any optimal multiset of coins that sum to n. Let $\mathcal{G}$ denote the multiset of coins that we get by the greedy solution for n. Then $|O| = |G|$. That is, the greedy solution uses the exact same number of coins as any optimal solution.

Coin Change

Theorem

Let n be a positive integer. Let $\mathcal{O}$ denote any optimal multiset of coins that sum to n. Let $\mathcal{G}$ denote the multiset of coins that we get by the greedy solution for n. Then $|O| = |G|$. That is, the greedy solution uses the exact same number of coins as any optimal solution.

Coin Change

Theorem

Let n be a positive integer. Let $\mathcal{O}$ denote any optimal multiset of coins that sum to n. Let $\mathcal{G}$ denote the multiset of coins that we get by the greedy solution for n. Then $|O| = |G|$. That is, the greedy solution uses the exact same number of coins as any optimal solution.

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than two 10s.

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least three 10s. But then we can get another solution $\mathcal{O}'$ by replacing three 10 with a 25 and a 5. Since we are removing three elements and adding two to get $\mathcal{O}'$, this means that $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than two 10s. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than two 10*s.*

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least three 10s. But then we can get another solution $\mathcal{O}'$ by replacing three 10 with a 25 and a 5. Since we are removing three elements and adding two to get $\mathcal{O}'$, this means that $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than two 10s. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than two 10s.

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least three 10s. But then we can get another solution $\mathcal{O}'$ by replacing three 10 with a 25 and a 5. Since we are removing three elements and adding two to get $\mathcal{O}'$, this means that $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than two 10s. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than two 10s.

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least three 10s. But then we can get another solution $\mathcal{O}'$ by replacing three 10 with a 25 and a 5. Since we are removing three elements and adding two to get $\mathcal{O}'$, this means that $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than two 10s. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than two 10s.

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least three 10s. But then we can get another solution $\mathcal{O}'$ by replacing three 10 with a 25 and a 5. Since we are removing three elements and adding two to get $\mathcal{O}'$, this means that $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than two 10s. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than two 10*s.*

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least three 10s. But then we can get another solution $\mathcal{O}'$ by replacing three 10 with a 25 and a 5. Since we are removing three elements and adding two to get $\mathcal{O}'$, this means that $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than two 10s. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than two 10*s.*

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least three 10s. But then we can get another solution $\mathcal{O}'$ by replacing three 10 with a 25 and a 5. Since we are removing three elements and adding two to get $\mathcal{O}'$, this means that $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than two 10s. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than one 5s.

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least two 5s.But then we can get another solution $\mathcal{O}'$ by replacing two 5 with a 10. Again, since we are removing two elements and adding one to get $\mathcal{O}'$, we have $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than one 5s. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than one 5s.

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least two 5s.But then we can get another solution $\mathcal{O}'$ by replacing two 5 with a 10. Again, since we are removing two elements and adding one to get $\mathcal{O}'$, we have $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than one 5s. $\square$

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than one 5s.

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least two 5s. But then we can get another solution $\mathcal{O}'$ by replacing two 5 with a 10. Again, since we are removing two elements and adding one to get $\mathcal{O}'$, we have $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than one 5s. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than one 5s.

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least two 5s.But then we can get another solution $\mathcal{O}'$ by replacing two 5 with a 10. Again, since we are removing two elements and adding one to get $\mathcal{O}'$, we have $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than one 5s. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than one 5s.

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least two 5s.But then we can get another solution $\mathcal{O}'$ by replacing two 5 with a 10. Again, since we are removing two elements and adding one to get $\mathcal{O}'$, we have $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than one 5s. □

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than one 5s.

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least two 5s.But then we can get another solution $\mathcal{O}'$ by replacing two 5 with a 10. Again, since we are removing two elements and adding one to get $\mathcal{O}'$, we have $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than one 5s. □

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than one 5s.

Proof.

Suppose, on the contrary, that $\mathcal{O}$ has at least two 5s.But then we can get another solution $\mathcal{O}'$ by replacing two 5 with a 10. Again, since we are removing two elements and adding one to get $\mathcal{O}'$, we have $|\mathcal{O}'| = |\mathcal{O}| - 1$. But $\mathcal{O}$ is suppose to be an optimal solution! So $\mathcal{O}$ does not more than one 5s. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ does not have more than four 1*s.*

Proof.

Exercise! $\square$

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ can not have both two 10s and one 5.

Proof.

Exercise! □

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then $\mathcal{O}$ can not have both two 10s and one 5.

Proof.

Exercise! $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then, excluding 25s, $\mathcal{O}$ sums to at most 24.

Proof.

The optimal multiset $\mathcal{O}$ can have at most two 10s, one 5, and four 1s. If it has two 10s, then their sum can be at most $2 \times 10 + 0 \times 5 + 4 \times 1 = 24$. If it does not have two 10s, then their sum can be at most $1 \times 10 + 1 \times 5 + 4 \times 1 = 19$. So if we exclude 25, the rest of $\mathcal{O}$ can sum to at most 24. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then, excluding 25s, $\mathcal{O}$ sums to at most 24.

Proof.

The optimal multiset $\mathcal{O}$ can have at most two 10s, one 5, and four 1s. If it has two 10s, then their sum can be at most $2 \times 10 + 0 \times 5 + 4 \times 1 = 24$. If it does not have two 10s, then their sum can be at most $1 \times 10 + 1 \times 5 + 4 \times 1 = 19$. So if we exclude 25, the rest of $\mathcal{O}$ can sum to at most 24. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then, excluding 25s, $\mathcal{O}$ sums to at most 24.

Proof.

The optimal multiset $\mathcal{O}$ can have at most two 10s, one 5, and four 1s. If it has two 10s, then their sum can be at most $2 \times 10 + 0 \times 5 + 4 \times 1 = 24$. If it does not have two 10s, then their sum can be at most $1 \times 10 + 1 \times 5 + 4 \times 1 = 19$. So if we exclude 25, the rest of $\mathcal{O}$ can sum to at most 24. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then, excluding 25s, $\mathcal{O}$ sums to at most 24.

Proof.

The optimal multiset $\mathcal{O}$ can have at most two 10s, one 5, and four 1s. If it has two 10s, then their sum can be at most $2 \times 10 + 0 \times 5 + 4 \times 1 = 24$. If it does not have two 10s, then their sum can be at most $1 \times 10 + 1 \times 5 + 4 \times 1 = 19$. So if we exclude 25, the rest of $\mathcal{O}$ can sum to at most 24. $\square$

Coin Change

Lemma

Let $\mathcal{O}$ denote any optimal multiset that sum to the positive integer n. Then, excluding 25s, $\mathcal{O}$ sums to at most 24.

Proof.

The optimal multiset $\mathcal{O}$ can have at most two 10s, one 5, and four 1s. If it has two 10s, then their sum can be at most $2 \times 10 + 0 \times 5 + 4 \times 1 = 24$. If it does not have two 10s, then their sum can be at most $1 \times 10 + 1 \times 5 + 4 \times 1 = 19$. So if we exclude 25, the rest of $\mathcal{O}$ can sum to at most 24. $\square$

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 25s.

Proof.

Let q equal the frequency/count of 25 in $\mathcal{O}$; let q' equal the frequency of 25 in $\mathcal{G}$. Since the greedy algorithm takes as many 25 as it can, we must have $q \leq q'$. Suppose that $q < q'$. But the rest of $\mathcal{O}$ can sum to at most 24. So the sum of $\mathcal{O}$ is at most $25q + 24$, whereas the sum of $\mathcal{G}$ is at least $25q'$. Thus, we must have $q = q'$. □

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 25s.

Proof.

Let q equal the frequency/count of 25 in $\mathcal{O}$; let q' equal the frequency of 25 in $\mathcal{G}$. Since the greedy algorithm takes as many 25 as it can, we must have $q \leq q'$. Suppose that $q < q'$. But the rest of $\mathcal{O}$ can sum to at most 24. So the sum of $\mathcal{O}$ is at most $25q + 24$, whereas the sum of $\mathcal{G}$ is at least $25q'$. Thus, we must have $q = q'$. $\square$

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 25s.

Proof.

Let q equal the frequency/count of 25 in $\mathcal{O}$; let q' equal the frequency of 25 in $\mathcal{G}$. Since the greedy algorithm takes as many 25 as it can, we must have $q \leq q'$. Suppose that $q < q'$. But the rest of $\mathcal{O}$ can sum to at most 24. So the sum of $\mathcal{O}$ is at most $25q + 24$, whereas the sum of $\mathcal{G}$ is at least $25q'$. Thus, we must have $q = q'$. □

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 25s.

Proof.

Let q equal the frequency/count of 25 in $\mathcal{O}$; let q' equal the frequency of 25 in $\mathcal{G}$. Since the greedy algorithm takes as many 25 as it can, we must have $q \leq q'$. Suppose that $q < q'$. But the rest of $\mathcal{O}$ can sum to at most 24. So the sum of $\mathcal{O}$ is at most $25q + 24$, whereas the sum of $\mathcal{G}$ is at least $25q'$. Thus, we must have $q = q'$. $\square$

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 25s.

Proof.

Let q equal the frequency/count of 25 in $\mathcal{O}$; let q' equal the frequency of 25 in $\mathcal{G}$. Since the greedy algorithm takes as many 25 as it can, we must have $q \leq q'$. Suppose that $q < q'$. But the rest of $\mathcal{O}$ can sum to at most 24. So the sum of $\mathcal{O}$ is at most $25q + 24$, whereas the sum of $\mathcal{G}$ is at least $25q'$. Thus, we must have $q = q'$. □

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 25s.

Proof.

Let q equal the frequency/count of 25 in $\mathcal{O}$; let q' equal the frequency of 25 in $\mathcal{G}$. Since the greedy algorithm takes as many 25 as it can, we must have $q \leq q'$. Suppose that $q < q'$. But the rest of $\mathcal{O}$ can sum to at most 24. So the sum of $\mathcal{O}$ is at most $25q + 24$, whereas the sum of $\mathcal{G}$ is at least $25q'$. Thus, we must have $q = q'$. □

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 25s.

Proof.

Let q equal the frequency/count of 25 in $\mathcal{O}$; let q' equal the frequency of 25 in $\mathcal{G}$. Since the greedy algorithm takes as many 25 as it can, we must have $q \leq q'$. Suppose that $q < q'$. But the rest of $\mathcal{O}$ can sum to at most 24. So the sum of $\mathcal{O}$ is at most $25q + 24$, whereas the sum of $\mathcal{G}$ is at least $25q'$. Thus, we must have $q = q'$. □

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 25s.

Proof.

Let q equal the frequency/count of 25 in $\mathcal{O}$; let q' equal the frequency of 25 in $\mathcal{G}$. Since the greedy algorithm takes as many 25 as it can, we must have $q \leq q'$. Suppose that $q < q'$. But the rest of $\mathcal{O}$ can sum to at most 24. So the sum of $\mathcal{O}$ is at most $25q + 24$, whereas the sum of $\mathcal{G}$ is at least $25q'$. Thus, we must have $q = q'$. $\square$

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 10*s.*

Proof.

Let d equal the frequency of 10 in $\mathcal{O}$; let d' equal the frequency of 10 in $\mathcal{G}$. Since the greedy algorithm takes as many 10 as it can, we must have $d \leq d'$. Suppose that $d < d'$. But the rest of $\mathcal{O}$ can sum to at most $1 \times 5 + 4 \times 1 = 9$. So the sum of $\mathcal{O}$ is at most $25q + 10d + 9$, whereas the sum of $\mathcal{O}'$ is at least $25q + 10d'$. Thus, we must have $d = d'$. □

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 10s.

Proof.

Let d equal the frequency of 10 in $\mathcal{O}$; let d' equal the frequency of 10 in $\mathcal{G}$. Since the greedy algorithm takes as many 10 as it can, we must have $d \leq d'$. Suppose that $d < d'$. But the rest of $\mathcal{O}$ can sum to at most $1 \times 5 + 4 \times 1 = 9$. So the sum of $\mathcal{O}$ is at most $25q + 10d + 9$, whereas the sum of $\mathcal{O}'$ is at least $25q + 10d'$. Thus, we must have $d = d'$. □

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 10s.

Proof.

Let d equal the frequency of 10 in $\mathcal{O}$; let d' equal the frequency of 10 in $\mathcal{G}$. Since the greedy algorithm takes as many 10 as it can, we must have $d \leq d'$. Suppose that $d < d'$. But the rest of $\mathcal{O}$ can sum to at most $1 \times 5 + 4 \times 1 = 9$. So the sum of $\mathcal{O}$ is at most $25q + 10d + 9$, whereas the sum of $\mathcal{O}'$ is at least $25q + 10d'$. Thus, we must have $d = d'$. $\square$

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 10s.

Proof.

Let d equal the frequency of 10 in $\mathcal{O}$; let d' equal the frequency of 10 in $\mathcal{G}$. Since the greedy algorithm takes as many 10 as it can, we must have $d \leq d'$. Suppose that $d < d'$. But the rest of $\mathcal{O}$ can sum to at most $1 \times 5 + 4 \times 1 = 9$. So the sum of $\mathcal{O}$ is at most $25q + 10d + 9$, whereas the sum of $\mathcal{O}'$ is at least $25q + 10d'$. Thus, we must have $d = d'$. $\square$

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 10s.

Proof.

Let d equal the frequency of 10 in $\mathcal{O}$; let d' equal the frequency of 10 in $\mathcal{G}$. Since the greedy algorithm takes as many 10 as it can, we must have $d \leq d'$. Suppose that $d < d'$. But the rest of $\mathcal{O}$ can sum to at most $1 \times 5 + 4 \times 1 = 9$. So the sum of $\mathcal{O}$ is at most $25q + 10d + 9$, whereas the sum of $\mathcal{O}'$ is at least $25q + 10d'$. Thus, we must have $d = d'$. $\square$

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 10s.

Proof.

Let d equal the frequency of 10 in $\mathcal{O}$; let d' equal the frequency of 10 in $\mathcal{G}$. Since the greedy algorithm takes as many 10 as it can, we must have $d \leq d'$. Suppose that $d < d'$. But the rest of $\mathcal{O}$ can sum to at most $1 \times 5 + 4 \times 1 = 9$. So the sum of $\mathcal{O}$ is at most $25q + 10d + 9$, whereas the sum of $\mathcal{O}'$ is at least $25q + 10d'$. Thus, we must have $d = d'$. □

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 10s.

Proof.

Let d equal the frequency of 10 in $\mathcal{O}$; let d' equal the frequency of 10 in $\mathcal{G}$. Since the greedy algorithm takes as many 10 as it can, we must have $d \leq d'$. Suppose that $d < d'$. But the rest of $\mathcal{O}$ can sum to at most $1 \times 5 + 4 \times 1 = 9$. So the sum of $\mathcal{O}$ is at most $25q + 10d + 9$, whereas the sum of $\mathcal{O}'$ is at least $25q + 10d'$. Thus, we must have $d = d'$. $\square$

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 10*s.*

Proof.

Let d equal the frequency of 10 in $\mathcal{O}$; let d' equal the frequency of 10 in $\mathcal{G}$. Since the greedy algorithm takes as many 10 as it can, we must have $d \leq d'$. Suppose that $d < d'$. But the rest of $\mathcal{O}$ can sum to at most $1 \times 5 + 4 \times 1 = 9$. So the sum of $\mathcal{O}$ is at most $25q + 10d + 9$, whereas the sum of $\mathcal{O}'$ is at least $25q + 10d'$. Thus, we must have $d = d'$. $\square$

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n, $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 5s.

Proof.

Exercise! $\square$

Coin Change

Lemma

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n; let $\mathcal{G}$ be our greedy solution. Then $\mathcal{O}$ and $\mathcal{G}$ has the same number of 1s.

Proof.

Exercise! $\square$

Coin Change

Theorem

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n and $\mathcal{G}$ denote the greedy solution. Then $|\mathcal{O}| = |\mathcal{G}|$.

Proof.

By the preceding lemmas, we know that $\mathcal{O}$ and $\mathcal{G}$ have the exact same number of 1, 5, 10, and 25. Thus, $\mathcal{O} = \mathcal{G}$. Obviously, $|\mathcal{O}| = |\mathcal{G}|$. $\square$

Coin Change

Theorem

Let n be a positive integer. Let $\mathcal{O}$ denote an optimal solution for n and $\mathcal{G}$ denote the greedy solution. Then $|\mathcal{O}| = |\mathcal{G}|$.

Proof.

By the preceding lemmas, we know that $\mathcal{O}$ and $\mathcal{G}$ have the exact same number of 1, 5, 10, and 25. Thus, $\mathcal{O} = \mathcal{G}$. Obviously, $|\mathcal{O}| = |\mathcal{G}|$. $\square$

Minimizing Dot Product

Input

We are given a positive integer n ($1 \leq n \leq 10^5$) and two lists $x = [x_1, \ldots, x_n]$ and $y = [y_1, \ldots, y_n]$ of size n. For all integers i such that $1 \leq i \leq n$, we have $-10^6 \leq x_i, y_i \leq 10^6$.

Output

Find the minimum value of $\sum_{i=1}^{n} x'_i y'_i$ over all permutations x' and y' of x and y respectively.

Sample Input

3

4 5 6

3 1 2

Sample Output

28

Minimizing Dot Product

Input

We are given a positive integer n ($1 \leq n \leq 10^5$) and two lists $x = [x_1, \ldots, x_n]$ and $y = [y_1, \ldots, y_n]$ of size n. For all integers i such that $1 \leq i \leq n$, we have $-10^6 \leq x_i, y_i \leq 10^6$.

Output

Find the minimum value of $\sum_{i=1}^{n} x'_i y'_i$ over all permutations x' and y' of x and y respectively.

Sample Input

3

4 5 6

3 1 2

Sample Output

28

Minimizing Dot Product

Input

We are given a positive integer n $(1 \leq n \leq 10^5)$ and two lists $x = [x_1, \ldots, x_n]$ and $y = [y_1, \ldots, y_n]$ of size n. For all integers i such that $1 \leq i \leq n$, we have $-10^6 \leq x_i, y_i \leq 10^6$.

Output

Find the minimum value of $\sum_{i=1}^{n} x_i' y_i'$ over all permutations x' and y' of x and y respectively.

Sample Input

3

4 5 6

3 1 2

Sample Output

28

Minimizing Dot Product

Input

We are given a positive integer n ($1 \leq n \leq 10^5$) and two lists $x = [x_1, \ldots, x_n]$ and $y = [y_1, \ldots, y_n]$ of size n. For all integers i such that $1 \leq i \leq n$, we have $-10^6 \leq x_i, y_i \leq 10^6$.

Output

Find the minimum value of $\sum_{i=1}^{n} x'_i y'_i$ over all permutations x' and y' of x and y respectively.

Sample Input

3

4 5 6

3 1 2

Sample Output

28

Minimizing Dot Product

Minimizing Dot Product

Algorithm 3 MinimizingDotProduct(n, x, y)

Sort x in ascending (non-decreasing) order.
Sort y in descending (non-increasing) order.
Set $sum \leftarrow 0$.
for $i = 1, \ldots, n$ **do** ▷ x and y have 1-based.
    Set $sum \leftarrow sum + x_i \cdot y_i$.
end for
return sum.

Minimizing Dot Product

Theorem

Let $x' = [x'_1, \ldots, x'_n]$ equal x sorted in ascending (more precisely, non-decreasing) order. Let $\mathcal{O} = [o_1, \ldots, o_n]$ be any optimal permutation of y. Let $\mathcal{G} = [g_1, \ldots, g_n]$ be the greedy solution. Then $\sum_{i=1}^{n} x'_i o_i = \sum_{i=1}^{n} x'_i g_i$. In other words, $\mathcal{G}$ is optimal.

Proof.

While there exists integers i, j with $1 \leq i < j \leq n$ and $o_i < o_j$, we swap them. At each iteration, the cost increases by at most $\left(x'_i o_j + x'_j o_i\right) - \left(x'_i o_i + x'_j o_j\right) = (x'_i - x'_j)(o_j - o_i) \leq 0$.

When this loop stops (why?), the final $\mathcal{O}$ becomes exactly $\mathcal{G}$. Since $\mathcal{O}$ is optimal, the cost stays the same after each iteration. Thus, $\mathcal{G}$ is optimal. $\square$

Minimizing Dot Product

Theorem

Let $x' = [x'_1, \ldots, x'_n]$ equal x sorted in ascending (more precisely, non-decreasing) order. Let $\mathcal{O} = [o_1, \ldots, o_n]$ be any optimal permutation of y. Let $\mathcal{G} = [g_1, \ldots, g_n]$ be the greedy solution. Then $\sum_{i=1}^{n} x'_i o_i = \sum_{i=1}^{n} x'_i g_i$. In other words, $\mathcal{G}$ is optimal.

Proof.

While there exists integers i, j with $1 \leq i < j \leq n$ and $o_i < o_j$, we swap them. At each iteration, the cost increases by at most $\left(x'_i o_j + x'_j o_i\right) - \left(x'_i o_i + x'_j o_j\right) = (x'_i - x'_j)(o_j - o_i) \leq 0.$

When this loop stops (why?), the final $\mathcal{O}$ becomes exactly $\mathcal{G}$. Since $\mathcal{O}$ is optimal, the cost stays the same after each iteration. Thus, $\mathcal{G}$ is optimal. $\square$

Minimizing Dot Product

Theorem

Let $x' = [x'_1, \ldots, x'_n]$ equal x sorted in ascending (more precisely, non-decreasing) order. Let $\mathcal{O} = [o_1, \ldots, o_n]$ be any optimal permutation of y. Let $\mathcal{G} = [g_1, \ldots, g_n]$ be the greedy solution. Then $\sum_{i=1}^{n} x'_i o_i = \sum_{i=1}^{n} x'_i g_i$. In other words, $\mathcal{G}$ is optimal.

Proof.

While there exists integers i, j with $1 \leq i < j \leq n$ and $o_i < o_j$, we swap them. At each iteration, the cost increases by at most $\left(x'_i o_j + x'_j o_i\right) - \left(x'_i o_i + x'_j o_j\right) = (x'_i - x'_j)(o_j - o_i) \leq 0$.

When this loop stops (why?), the final $\mathcal{O}$ becomes exactly $\mathcal{G}$.

Since $\mathcal{O}$ is optimal, the cost stays the same after each iteration.

Thus, $\mathcal{G}$ is optimal. $\square$

Minimizing Dot Product

Theorem

Let $x' = [x'_1, \ldots, x'_n]$ equal x sorted in ascending (more precisely, non-decreasing) order. Let $\mathcal{O} = [o_1, \ldots, o_n]$ be any optimal permutation of y. Let $\mathcal{G} = [g_1, \ldots, g_n]$ be the greedy solution. Then $\sum_{i=1}^{n} x'_i o_i = \sum_{i=1}^{n} x'_i g_i$. In other words, $\mathcal{G}$ is optimal.

Proof.

While there exists integers i, j with $1 \leq i < j \leq n$ and $o_i < o_j$, we swap them. At each iteration, the cost increases by at most $\left(x'_i o_j + x'_j o_i\right) - \left(x'_i o_i + x'_j o_j\right) = (x'_i - x'_j)(o_j - o_i) \leq 0$.

When this loop stops (why?), the final $\mathcal{O}$ becomes exactly $\mathcal{G}$.

Since $\mathcal{O}$ is optimal, the cost stays the same after each iteration.

Thus, $\mathcal{G}$ is optimal. □

Minimizing Dot Product

Theorem

Let $x' = [x'_1, \ldots, x'_n]$ equal x sorted in ascending (more precisely, non-decreasing) order. Let $\mathcal{O} = [o_1, \ldots, o_n]$ be any optimal permutation of y. Let $\mathcal{G} = [g_1, \ldots, g_n]$ be the greedy solution. Then $\sum_{i=1}^{n} x'_i o_i = \sum_{i=1}^{n} x'_i g_i$. In other words, $\mathcal{G}$ is optimal.

Proof.

While there exists integers i, j with $1 \leq i < j \leq n$ and $o_i < o_j$, we swap them. At each iteration, the cost increases by at most

$$\left(x'_i o_j + x'_j o_i\right) - \left(x'_i o_i + x'_j o_j\right) = (x'_i - x'_j)(o_j - o_i) \leq 0.$$

When this loop stops (why?), the final $\mathcal{O}$ becomes exactly $\mathcal{G}$. Since $\mathcal{O}$ is optimal, the cost stays the same after each iteration. Thus, $\mathcal{G}$ is optimal. □

Minimizing Dot Product

Theorem

Let $x' = [x'_1, \ldots, x'_n]$ equal x sorted in ascending (more precisely, non-decreasing) order. Let $\mathcal{O} = [o_1, \ldots, o_n]$ be any optimal permutation of y. Let $\mathcal{G} = [g_1, \ldots, g_n]$ be the greedy solution. Then $\sum_{i=1}^{n} x'_i o_i = \sum_{i=1}^{n} x'_i g_i$. In other words, $\mathcal{G}$ is optimal.

Proof.

While there exists integers i, j with $1 \leq i < j \leq n$ and $o_i < o_j$, we swap them. At each iteration, the cost increases by at most

$$\left(x'_i o_j + x'_j o_i\right) - \left(x'_i o_i + x'_j o_j\right) = (x'_i - x'_j)(o_j - o_i) \leq 0.$$

When this loop stops (why?), the final $\mathcal{O}$ becomes exactly $\mathcal{G}$.

Since $\mathcal{O}$ is optimal, the cost stays the same after each iteration.

Thus, $\mathcal{G}$ is optimal. □

Minimizing Dot Product

Theorem

Let $x' = [x'_1, \ldots, x'_n]$ equal x sorted in ascending (more precisely, non-decreasing) order. Let $\mathcal{O} = [o_1, \ldots, o_n]$ be any optimal permutation of y. Let $\mathcal{G} = [g_1, \ldots, g_n]$ be the greedy solution. Then $\sum_{i=1}^{n} x'_i o_i = \sum_{i=1}^{n} x'_i g_i$. In other words, $\mathcal{G}$ is optimal.

Proof.

While there exists integers i, j with $1 \leq i < j \leq n$ and $o_i < o_j$, we swap them. At each iteration, the cost increases by at most $\left(x'_i o_j + x'_j o_i\right) - \left(x'_i o_i + x'_j o_j\right) = (x'_i - x'_j)(o_j - o_i) \leq 0$.

When this loop stops (why?), the final $\mathcal{O}$ becomes exactly $\mathcal{G}$.

Since $\mathcal{O}$ is optimal, the cost stays the same after each iteration.

Thus, $\mathcal{G}$ is optimal. $\square$

Minimizing Dot Product

Theorem

Let $x' = [x'_1, \ldots, x'_n]$ equal x sorted in ascending (more precisely, non-decreasing) order. Let $\mathcal{O} = [o_1, \ldots, o_n]$ be any optimal permutation of y. Let $\mathcal{G} = [g_1, \ldots, g_n]$ be the greedy solution. Then $\sum_{i=1}^{n} x'_i o_i = \sum_{i=1}^{n} x'_i g_i$. In other words, $\mathcal{G}$ is optimal.

Proof.

While there exists integers i, j with $1 \leq i < j \leq n$ and $o_i < o_j$, we swap them. At each iteration, the cost increases by at most $\left(x'_i o_j + x'_j o_i\right) - \left(x'_i o_i + x'_j o_j\right) = (x'_i - x'_j)(o_j - o_i) \leq 0$.

When this loop stops (why?), the final $\mathcal{O}$ becomes exactly $\mathcal{G}$.

Since $\mathcal{O}$ is optimal, the cost stays the same after each iteration.

Thus, $\mathcal{G}$ is optimal. $\square$

Minimizing Dot Product

Theorem

Let $x' = [x'_1, \ldots, x'_n]$ equal x sorted in ascending (more precisely, non-decreasing) order. Let $\mathcal{O} = [o_1, \ldots, o_n]$ be any optimal permutation of y. Let $\mathcal{G} = [g_1, \ldots, g_n]$ be the greedy solution. Then $\sum_{i=1}^{n} x'_i o_i = \sum_{i=1}^{n} x'_i g_i$. In other words, $\mathcal{G}$ is optimal.

Proof.

While there exists integers i, j with $1 \leq i < j \leq n$ and $o_i < o_j$, we swap them. At each iteration, the cost increases by at most $\left(x'_i o_j + x'_j o_i\right) - \left(x'_i o_i + x'_j o_j\right) = (x'_i - x'_j)(o_j - o_i) \leq 0.$

When this loop stops (why?), the final $\mathcal{O}$ becomes exactly $\mathcal{G}$.

Since $\mathcal{O}$ is optimal, the cost stays the same after each iteration.

Thus, $\mathcal{G}$ is optimal. $\square$

Minimizing Dot Product

Theorem

Let $x' = [x'_1, \ldots, x'_n]$ equal x sorted in ascending (more precisely, non-decreasing) order. Let $\mathcal{O} = [o_1, \ldots, o_n]$ be any optimal permutation of y. Let $\mathcal{G} = [g_1, \ldots, g_n]$ be the greedy solution. Then $\sum_{i=1}^{n} x'_i o_i = \sum_{i=1}^{n} x'_i g_i$. In other words, $\mathcal{G}$ is optimal.

Proof.

While there exists integers i, j with $1 \leq i < j \leq n$ and $o_i < o_j$, we swap them. At each iteration, the cost increases by at most $\left(x'_i o_j + x'_j o_i\right) - \left(x'_i o_i + x'_j o_j\right) = (x'_i - x'_j)(o_j - o_i) \leq 0$.

When this loop stops (why?), the final $\mathcal{O}$ becomes exactly $\mathcal{G}$.

Since $\mathcal{O}$ is optimal, the cost stays the same after each iteration.

Thus, $\mathcal{G}$ is optimal. □

Movie Festival (CSES)

Input

We are given a positive integer n ($1 \le n \le 2 \cdot 10^5$). For each i with $1 \le i \le n$, we are also given integers a_i and b_i ($1 \le a_i < b_i \le 10^9$) representing the starting time and the ending time of the ith movie. We can not watch two movie at the same time.

Output

Find the maximum number of movies that we can watch.

Sample Input

3

3 5 4 9 5 8

Sample Output

2

Movie Festival (CSES)

Input

We are given a positive integer n ($1 \leq n \leq 2 \cdot 10^5$). For each i with $1 \leq i \leq n$, we are also given integers a_i and b_i ($1 \leq a_i < b_i \leq 10^9$) representing the starting time and the ending time of the ith movie. We can not watch two movie at the same time.

Output

Find the maximum number of movies that we can watch.

Sample Input

3

3 5 4 9 5 8

Sample Output

2

Movie Festival (CSES)

Input

We are given a positive integer n ($1 \leq n \leq 2 \cdot 10^5$). For each i with $1 \leq i \leq n$, we are also given integers a_i and b_i ($1 \leq a_i < b_i \leq 10^9$) representing the starting time and the ending time of the ith movie. We can not watch two movie at the same time.

Output

Find the maximum number of movies that we can watch.

Sample Input

3

3 5 4 9 5 8

Sample Output

2

Movie Festival (CSES)

Input

We are given a positive integer n ($1 \leq n \leq 2 \cdot 10^5$). For each i with $1 \leq i \leq n$, we are also given integers a_i and b_i ($1 \leq a_i < b_i \leq 10^9$) representing the starting time and the ending time of the ith movie. We can not watch two movie at the same time.

Output

Find the maximum number of movies that we can watch.

Sample Input

3

3 5 4 9 5 8

Sample Output

2

Movie Festival (CSES)

Movie Festival

Algorithm 4 MovieFestival(n, a, b)

```
procedure CMP((x, y), (p, q))
    if y ≠ q then
        return y < q.                ▷ The movie that finishes earlier should come first.
    else                             ▷ If two movies finish at the same time
        return x < p.                ▷ then the movie that starts ealier should come first.
    end if
end procedure
Set movies ← [(a_1, b_1), ..., (a_n, b_n)].
Sort movies by CMP.
Set counter ← 0.
Set current_time ← 0.
for (movie_start, movie_end) in movies do
    if movie_start ≥ current_time then
        Set counter ← counter + 1.
        Set current_time ← movie_end.
    end if
end for
return counter.
```

Movie Festival (CSES)

Theorem

For a given input, let $\mathcal{O}$ denote an optimal set of movies and let $\mathcal{G}$ denote the set of movies given by our greedy algorithm. Then $|\mathcal{O}| = |\mathcal{G}|$.

Proof.

Let $i_1, \ldots, i_r$ and $j_1, \ldots, j_s$ denote the list of movies watched in $\mathcal{O}$ and $\mathcal{G}$ respectively, sorted in order of their ending times. Of course, $r \geq s$.

While we do not have $i_1 = j_1, \ldots, i_s = j_s$, we can pick the smallest k such that $i_k \neq j_k$. This means that $b_{i_k} \geq b_{j_k}$ and $k = 1$ or $b_{i_{k-1}} \leq a_{j_k}$. So we can replace i_k with j_k in $\mathcal{O}$.

After the iterations end, we must have $i_1 = j_1, \ldots, i_s = j_s$. This implies that $r = s$ (why? exercise!) and thus $|\mathcal{O}| = |\mathcal{G}|$. $\square$

Movie Festival (CSES)

Theorem

For a given input, let $\mathcal{O}$ denote an optimal set of movies and let $\mathcal{G}$ denote the set of movies given by our greedy algorithm. Then $|\mathcal{O}| = |\mathcal{G}|$.

Proof.

Let $i_1, \ldots, i_r$ and $j_1, \ldots, j_s$ denote the list of movies watched in $\mathcal{O}$ and $\mathcal{G}$ respectively, sorted in order of their ending times. Of course, $r \geq s$.

While we do not have $i_1 = j_1, \ldots, i_s = j_s$, we can pick the smallest k such that $i_k \neq j_k$. This means that $b_{i_k} \geq b_{j_k}$ and $k = 1$ or $b_{i_{k-1}} \leq a_{j_k}$. So we can replace i_k with j_k in $\mathcal{O}$.

After the iterations end, we must have $i_1 = j_1, \ldots, i_s = j_s$. This implies that $r = s$ (why? exercise!) and thus $|\mathcal{O}| = |\mathcal{G}|$. $\square$

Movie Festival (CSES)

Theorem

For a given input, let $\mathcal{O}$ denote an optimal set of movies and let $\mathcal{G}$ denote the set of movies given by our greedy algorithm. Then $|\mathcal{O}| = |\mathcal{G}|$.

Proof.

Let $i_1, \ldots, i_r$ and $j_1, \ldots, j_s$ denote the list of movies watched in $\mathcal{O}$ and $\mathcal{G}$ respectively, sorted in order of their ending times. Of course, $r \geq s$.

While we do not have $i_1 = j_1, \ldots, i_s = j_s$, we can pick the smallest k such that $i_k \neq j_k$. This means that $b_{i_k} \geq b_{j_k}$ and $k = 1$ or $b_{i_{k-1}} \leq a_{j_k}$. So we can replace i_k with j_k in $\mathcal{O}$.

After the iterations end, we must have $i_1 = j_1, \ldots, i_s = j_s$. This implies that $r = s$ (why? exercise!) and thus $|\mathcal{O}| = |\mathcal{G}|$. $\square$

Movie Festival (CSES)

Theorem

For a given input, let $\mathcal{O}$ denote an optimal set of movies and let $\mathcal{G}$ denote the set of movies given by our greedy algorithm. Then $|\mathcal{O}| = |\mathcal{G}|$.

Proof.

Let $i_1, \ldots, i_r$ and $j_1, \ldots, j_s$ denote the list of movies watched in $\mathcal{O}$ and $\mathcal{G}$ respectively, sorted in order of their ending times. Of course, $r \geq s$.

While we do not have $i_1 = j_1, \ldots, i_s = j_s$, we can pick the smallest k such that $i_k \neq j_k$. This means that $b_{i_k} \geq b_{j_k}$ and $k = 1$ or $b_{i_{k-1}} \leq a_{j_k}$. So we can replace i_k with j_k in $\mathcal{O}$.

After the iterations end, we must have $i_1 = j_1, \ldots, i_s = j_s$. This implies that $r = s$ (why? exercise!) and thus $|\mathcal{O}| = |\mathcal{G}|$. $\square$

Movie Festival (CSES)

Theorem

For a given input, let $\mathcal{O}$ denote an optimal set of movies and let $\mathcal{G}$ denote the set of movies given by our greedy algorithm. Then $|\mathcal{O}| = |\mathcal{G}|$.

Proof.

Let $i_1, \ldots, i_r$ and $j_1, \ldots, j_s$ denote the list of movies watched in $\mathcal{O}$ and $\mathcal{G}$ respectively, sorted in order of their ending times. Of course, $r \geq s$.

While we do not have $i_1 = j_1, \ldots, i_s = j_s$, we can pick the smallest k such that $i_k \neq j_k$. This means that $b_{i_k} \geq b_{j_k}$ and $k = 1$ or $b_{i_{k-1}} \leq a_{j_k}$. So we can replace i_k with j_k in $\mathcal{O}$.

After the iterations end, we must have $i_1 = j_1, \ldots, i_s = j_s$. This implies that $r = s$ (why? exercise!) and thus $|\mathcal{O}| = |\mathcal{G}|$. □

Movie Festival (CSES)

Theorem

For a given input, let $\mathcal{O}$ denote an optimal set of movies and let $\mathcal{G}$ denote the set of movies given by our greedy algorithm. Then $|\mathcal{O}| = |\mathcal{G}|$.

Proof.

Let $i_1, \ldots, i_r$ and $j_1, \ldots, j_s$ denote the list of movies watched in $\mathcal{O}$ and $\mathcal{G}$ respectively, sorted in order of their ending times. Of course, $r \geq s$.

While we do not have $i_1 = j_1, \ldots, i_s = j_s$, we can pick the smallest k such that $i_k \neq j_k$. This means that $b_{i_k} \geq b_{j_k}$ and $k = 1$ or $b_{i_{k-1}} \leq a_{j_k}$. So we can replace i_k with j_k in $\mathcal{O}$.

After the iterations end, we must have $i_1 = j_1, \ldots, i_s = j_s$. This implies that $r = s$ (why? exercise!) and thus $|\mathcal{O}| = |\mathcal{G}|$. □

Movie Festival (CSES)

Theorem

For a given input, let $\mathcal{O}$ denote an optimal set of movies and let $\mathcal{G}$ denote the set of movies given by our greedy algorithm. Then $|\mathcal{O}| = |\mathcal{G}|$.

Proof.

Let $i_1, \ldots, i_r$ and $j_1, \ldots, j_s$ denote the list of movies watched in $\mathcal{O}$ and $\mathcal{G}$ respectively, sorted in order of their ending times. Of course, $r \geq s$.

While we do not have $i_1 = j_1, \ldots, i_s = j_s$, we can pick the smallest k such that $i_k \neq j_k$. This means that $b_{i_k} \geq b_{j_k}$ and $k = 1$ or $b_{i_{k-1}} \leq a_{j_k}$. So we can replace i_k with j_k in $\mathcal{O}$.

After the iterations end, we must have $i_1 = j_1, \ldots, i_s = j_s$. This implies that $r = s$ (why? exercise!) and thus $|\mathcal{O}| = |\mathcal{G}|$. □

Movie Festival (CSES)

Theorem

For a given input, let $\mathcal{O}$ denote an optimal set of movies and let $\mathcal{G}$ denote the set of movies given by our greedy algorithm. Then $|\mathcal{O}| = |\mathcal{G}|$.

Proof.

Let $i_1, \ldots, i_r$ and $j_1, \ldots, j_s$ denote the list of movies watched in $\mathcal{O}$ and $\mathcal{G}$ respectively, sorted in order of their ending times. Of course, $r \geq s$.

While we do not have $i_1 = j_1, \ldots, i_s = j_s$, we can pick the smallest k such that $i_k \neq j_k$. This means that $b_{i_k} \geq b_{j_k}$ and $k = 1$ or $b_{i_{k-1}} \leq a_{j_k}$. So we can replace i_k with j_k in $\mathcal{O}$.

After the iterations end, we must have $i_1 = j_1, \ldots, i_s = j_s$. This implies that $r = s$ (why? exercise!) and thus $|\mathcal{O}| = |\mathcal{G}|$. $\square$

Maximum Subarray Sum

Input

We are given an array A of n integers ($-10^9 \leq A[i] \leq 10^9$).

Output

We need to find the maximum sum of a contiguous subarray (possibly empty) of A. i.e., for all $1 \leq i \leq j \leq n$, we need to find the maximum value of $A[i] + A[i+1] + \ldots + A[j]$, or 0 in case of empty subarray.

Sample Input

9

-2 1 -3 4 -1 2 1 -5 4

Sample Output

6

Maximum Subarray Sum

Input

We are given an array A of n integers ($-10^9 \leq A[i] \leq 10^9$).

Output

We need to find the maximum sum of a contiguous subarray (possibly empty) of A. i.e., for all $1 \leq i \leq j \leq n$, we need to find the maximum value of $A[i] + A[i+1] + \ldots + A[j]$, or 0 in case of empty subarray.

Sample Input

9

-2 1 -3 4 -1 2 1 -5 4

Sample Output

6

Maximum Subarray Sum

Input

We are given an array A of n integers ($-10^9 \leq A[i] \leq 10^9$).

Output

We need to find the maximum sum of a contiguous subarray (possibly empty) of A. i.e., for all $1 \leq i \leq j \leq n$, we need to find the maximum value of $A[i] + A[i+1] + \ldots + A[j]$, or 0 in case of empty subarray.

Sample Input

9

-2 1 -3 4 -1 2 1 -5 4

Sample Output

6

Maximum Subarray Sum

Input

We are given an array A of n integers ($-10^9 \leq A[i] \leq 10^9$).

Output

We need to find the maximum sum of a contiguous subarray (possibly empty) of A. i.e., for all $1 \leq i \leq j \leq n$, we need to find the maximum value of $A[i] + A[i+1] + \ldots + A[j]$, or 0 in case of empty subarray.

Sample Input

9

-2 1 -3 4 -1 2 1 -5 4

Sample Output

6

Maximum Subarray Sum

Algorithm 5 **Kadane**$(\mathbf{n}, \mathbf{A})$

Set $ans \leftarrow 0$.
Set $cur \leftarrow 0$.
for $i = 1, \ldots, n$ **do**
    Set $cur \leftarrow cur + A[i]$.
    if $cur < 0$ **then**
        Set $cur = 0$.
    end if
    Set $ans = \max(ans, cur)$.
end for
return ans.

Maximum Subarray Sum

Theorem

Let $A[l \ldots r]$ be a subarray with maximum sum. Then there is no such index i such that $l \leq i \leq r$ and cur becomes negative at i.

Proof.

Let $sum(l', r') = A[l'] + A[l' + 1] + \ldots + A[r']$.

The value of cur before index l must be 0. Otherwise, we can simply extend l to the left and get a better answer.

Assume that ($l \leq i \leq r$) is the first index where cur becomes negative . Then, $cur = sum(l, i) < 0$.

$$sum(l, r) = sum(l, i) + sum(i + 1, r)$$

$$sum(l, r) < sum(i + 1, r)$$

Then, $A[i + 1 \ldots r]$ is a subarray with a larger sum than $A[l \ldots r]$, contradicting the assumption that $A[l \ldots r]$ is optimal.

Maximum Subarray Sum

Theorem

Let $A[l \dots r]$ be a subarray with maximum sum. Then there is no such index i such that $l \leq i \leq r$ and cur becomes negative at i.

Proof.

Let $sum(l', r') = A[l'] + A[l' + 1] + \dots + A[r']$.

The value of cur before index l must be 0. Otherwise, we can simply extend l to the left and get a better answer.

Assume that ($l \leq i \leq r$) is the first index where cur becomes negative . Then, $cur = sum(l, i) < 0$.

$$sum(l, r) = sum(l, i) + sum(i + 1, r)$$

$$sum(l, r) < sum(i + 1, r)$$

Then, $A[i + 1 \dots r]$ is a subarray with a larger sum than $A[l \dots r]$, contradicting the assumption that $A[l \dots r]$ is optimal.

Maximum Subarray Sum

Theorem

Let $A[l \dots r]$ be a subarray with maximum sum. Then there is no such index i such that $l \le i \le r$ and cur becomes negative at i.

Proof.

Let $sum(l', r') = A[l'] + A[l' + 1] + \dots + A[r']$.

The value of cur before index l must be 0. Otherwise, we can simply extend l to the left and get a better answer.

Assume that $(l \le i \le r)$ is the first index where cur becomes negative . Then, $cur = sum(l, i) < 0$.

$$sum(l, r) = sum(l, i) + sum(i + 1, r)$$

$$sum(l, r) < sum(i + 1, r)$$

Then, $A[i + 1 \dots r]$ is a subarray with a larger sum than $A[l \dots r]$, contradicting the assumption that $A[l \dots r]$ is optimal.

Maximum Subarray Sum

Theorem

Let $A[l \dots r]$ be a subarray with maximum sum. Then there is no such index i such that $l \leq i \leq r$ and cur becomes negative at i.

Proof.

Let $sum(l', r') = A[l'] + A[l'+1] + \dots + A[r']$.

The value of *cur* before index l must be 0. Otherwise, we can simply extend l to the left and get a better answer.

Assume that $(l \leq i \leq r)$ is the first index where *cur* becomes negative . Then, $cur = sum(l, i) < 0$.

$$sum(l, r) = sum(l, i) + sum(i+1, r)$$

$$sum(l, r) < sum(i+1, r)$$

Then, $A[i+1 \dots r]$ is a subarray with a larger sum than $A[l \dots r]$, contradicting the assumption that $A[l \dots r]$ is optimal.

Maximum Subarray Sum

Theorem

Let $A[l \ldots r]$ be a subarray with maximum sum. Then there is no such index i such that $l \leq i \leq r$ and cur becomes negative at i.

Proof.

Let $sum(l', r') = A[l'] + A[l' + 1] + \ldots + A[r']$.
The value of *cur* before index l must be 0. Otherwise, we can simply extend l to the left and get a better answer.
Assume that ($l \leq i \leq r$) is the first index where *cur* becomes negative . Then, $cur = sum(l, i) < 0$.

$$sum(l, r) = sum(l, i) + sum(i + 1, r)$$
$$sum(l, r) < sum(i + 1, r)$$

Then, $A[i + 1 \ldots r]$ is a subarray with a larger sum than $A[l \ldots r]$, contradicting the assumption that $A[l \ldots r]$ is optimal.

Maximum Subarray Sum

Correctness of Kadane's Algorithm

- The optimal answer is 0 iff there is no positive integer in A. In that case, Kadene's algorithm returns 0.
- If the optimal subarray is $A[l \ldots r]$, then cur is 0 before l.
- cur does not become negative between l and r.
- $cur = sum(l, r)$ at r.
- cur does not increase after r.

Correctness of Kadane's Algorithm

- The optimal answer is 0 iff there is no positive integer in A. In that case, Kadene's algorithm returns 0.
- If the optimal subarray is $A[l \dots r]$, then cur is 0 before l.
- cur does not become negative between l and r.
- $cur = sum(l, r)$ at r.
- cur does not increase after r.

Maximum Subarray Sum

Correctness of Kadane's Algorithm

- The optimal answer is 0 iff there is no positive integer in A. In that case, Kadene's algorithm returns 0.
- If the optimal subarray is $A[l \dots r]$, then cur is 0 before l.
- cur does not become negative between l and r.
- $cur = sum(l, r)$ at r.
- cur does not increase after r.

Correctness of Kadane's Algorithm

- The optimal answer is 0 iff there is no positive integer in A. In that case, Kadene's algorithm returns 0.
- If the optimal subarray is $A[l \dots r]$, then cur is 0 before l.
- cur does not become negative between l and r.
- $cur = sum(l, r)$ at r.
- cur does not increase after r.

Maximum Subarray Sum

Correctness of Kadane's Algorithm

- The optimal answer is 0 iff there is no positive integer in A. In that case, Kadene's algorithm returns 0.
- If the optimal subarray is $A[l \dots r]$, then cur is 0 before l.
- cur does not become negative between l and r.
- $cur = sum(l, r)$ at r.
- cur does not increase after r.

Topcoder SRM 502 Div1 Medium (Simplified)

Input

We are participating in a programming contest. There are n problems in the contest. Each problem is assigned 3 values p_i, d_i, and t_i, where p_i is the maximum points, d_i is the decay of points per minutes and t_i is the time required to solve the problem. We need to solve all the problems.

Output

Find the maximum points we can get by solving the problems if we can solve them in any order.

Topcoder SRM 502 Div1 Medium (Simplified)

Input

We are participating in a programming contest. There are n problems in the contest. Each problem is assigned 3 values p_i, d_i, and t_i, where p_i is the maximum points, d_i is the decay of points per minutes and t_i is the time required to solve the problem. We need to solve all the problems.

Output

Find the maximum points we can get by solving the problems if we can solve them in any order.

Topcoder SRM 502 Div1 Medium (Simplified)

Sample Input 1

4

45 3 5

50 5 4

15 2 2

100 4 7

Sample Output 1

73

Topcoder SRM 502 Div1 Medium (Simplified)

Sample Input 1

4

45 3 5

50 5 4

15 2 2

100 4 7

Sample Output 1

73

Topcoder SRM 502 Div1 Medium (Simplified)

Sample Input 2

3

10 1 3

20 1 1

30 1 2

Sample Output 2

50

Topcoder SRM 502 Div1 Medium (Simplified)

Sample Input 2

```
3
10 1 3
20 1 1
30 1 2
```

Sample Output 2

```
50
```

Topcoder SRM 502 Div1 Medium (Simplified)

Optimal Order between two problems

Assume that we have solved some problems in x minutes. We will solve problem i and j. Then solve the rest. If order of every other problem is fixed, should we solve problem i first or j first?

The points we get from solving first few problems remain the same. Similarly, points we get from solving the last few problems remain the same.

Topcoder SRM 502 Div1 Medium (Simplified)

Optimal Order between two problems

Assume that we have solved some problems in x minutes. We will solve problem i and j. Then solve the rest. If order of every other problem is fixed, should we solve problem i first or j first? The points we get from solving first few problems remain the same. Similarly, points we get from solving the last few problems remain the same.

Topcoder SRM 502 Div1 Medium (Simplified)

Optimal Order between two problems

Points solving i before j: $p_i - d_i \cdot (x + t_i) + p_j - d_j \cdot (x + t_i + t_j)$

Points solving j before i: $p_j - d_j \cdot (x + t_j) + p_i - d_i \cdot (x + t_i + t_j)$

Solving i before j would give us more points if:

$$
\begin{aligned}
& p_i - d_i \cdot (x + t_i) + p_j - d_j \cdot (x + t_i + t_j) \\
& > p_j - d_j \cdot (x + t_j) + p_i - d_i \cdot (x + t_i + t_j) \\
& \implies -d_j t_i > -d_i t_j \\
& \implies \frac{d_i}{t_i} > \frac{d_j}{t_j}
\end{aligned}
$$

Therefore, it is better to solve problem i before j if $\frac{d_i}{t_i} > \frac{d_j}{t_j}$.

Topcoder SRM 502 Div1 Medium (Simplified)

Optimal Order between two problems

Points solving i before j: $p_i - d_i \cdot (x + t_i) + p_j - d_j \cdot (x + t_i + t_j)$

Points solving j before i: $p_j - d_j \cdot (x + t_j) + p_i - d_i \cdot (x + t_i + t_j)$

Solving i before j would give us more points if:

$$
\begin{aligned}
& p_i - d_i \cdot (x + t_i) + p_j - d_j \cdot (x + t_i + t_j) \\
& > p_j - d_j \cdot (x + t_j) + p_i - d_i \cdot (x + t_i + t_j) \\
& \implies -d_j t_i > -d_i t_j \\
& \implies \frac{d_i}{t_i} > \frac{d_j}{t_j}
\end{aligned}
$$

Therefore, it is better to solve problem i before j if $\frac{d_i}{t_i} > \frac{d_j}{t_j}$.

Topcoder SRM 502 Div1 Medium (Simplified)

Optimal Order between two problems

Points solving i before j: $p_i - d_i \cdot (x + t_i) + p_j - d_j \cdot (x + t_i + t_j)$

Points solving j before i: $p_j - d_j \cdot (x + t_j) + p_i - d_i \cdot (x + t_i + t_j)$

Solving i before j would give us more points if:

$$
\begin{aligned}
& p_i - d_i \cdot (x + t_i) + p_j - d_j \cdot (x + t_i + t_j) \\
& > p_j - d_j \cdot (x + t_j) + p_i - d_i \cdot (x + t_i + t_j) \\
& \implies -d_j t_i > -d_i t_j \\
& \implies \frac{d_i}{t_i} > \frac{d_j}{t_j}
\end{aligned}
$$

Therefore, it is better to solve problem i before j if $\frac{d_i}{t_i} > \frac{d_j}{t_j}$.

Topcoder SRM 502 Div1 Medium (Simplified)

Optimal Order between two problems

Points solving i before j: $p_i - d_i \cdot (x + t_i) + p_j - d_j \cdot (x + t_i + t_j)$

Points solving j before i: $p_j - d_j \cdot (x + t_j) + p_i - d_i \cdot (x + t_i + t_j)$

Solving i before j would give us more points if:

$$
\begin{aligned}
& p_i - d_i \cdot (x + t_i) + p_j - d_j \cdot (x + t_i + t_j) \\
& > p_j - d_j \cdot (x + t_j) + p_i - d_i \cdot (x + t_i + t_j) \\
& \implies -d_j t_i > -d_i t_j \\
& \implies \frac{d_i}{t_i} > \frac{d_j}{t_j}
\end{aligned}
$$

Therefore, it is better to solve problem i before j if $\frac{d_i}{t_i} > \frac{d_j}{t_j}$.

Topcoder SRM 502 Div1 Medium (Simplified)

Greedy Choice

We can sort the problems in decreasing order of $\frac{d_i}{t_i}$ and solve them in that order.

Proof.

Let $p_1, p_2, \ldots, p_n$ be the optimal order of solving the problems.
There is no adjacent pair p_i, p_{i+1} such that $\frac{d_i}{t_i} < \frac{d_{i+1}}{t_{i+1}}$. Because if there is, we can swap them and get a better order.
If there is no adjacent pair p_i, p_{i+1} such that $\frac{d_i}{t_i} < \frac{d_{i+1}}{t_{i+1}}$, then the problems are sorted in decreasing order of $\frac{d_i}{t_i}$.

Topcoder SRM 502 Div1 Medium (Simplified)

Greedy Choice

We can sort the problems in decreasing order of $\frac{d_i}{t_i}$ and solve them in that order.

Proof.

Let $p_1, p_2, \ldots, p_n$ be the optimal order of solving the problems.
There is no adjacent pair p_i, p_{i+1} such that $\frac{d_i}{t_i} < \frac{d_{i+1}}{t_{i+1}}$. Because if there is, we can swap them and get a better order.
If there is no adjacent pair p_i, p_{i+1} such that $\frac{d_i}{t_i} < \frac{d_{i+1}}{t_{i+1}}$, then the problems are sorted in decreasing order of $\frac{d_i}{t_i}$.

□

Topcoder SRM 502 Div1 Medium (Simplified)

Greedy Choice

We can sort the problems in decreasing order of $\frac{d_i}{t_i}$ and solve them in that order.

Proof.

Let $p_1, p_2, \ldots, p_n$ be the optimal order of solving the problems.
There is no adjacent pair p_i, p_{i+1} such that $\frac{d_i}{t_i} < \frac{d_{i+1}}{t_{i+1}}$. Because if there is, we can swap them and get a better order.
If there is no adjacent pair p_i, p_{i+1} such that $\frac{d_i}{t_i} < \frac{d_{i+1}}{t_{i+1}}$, then the problems are sorted in decreasing order of $\frac{d_i}{t_i}$.

$\square$

Topcoder SRM 502 Div1 Medium (Simplified)

Algorithm 6 MaxPoints(n, p, d, t)

Sort problems in decreasing order of $\frac{d_i}{t_i}$.
Set $sum \leftarrow 0$.
Set $time \leftarrow 0$.
for $i = 1, \ldots, n$ **do**
  Set $sum \leftarrow sum + p_i - d_i \cdot (time + t_i)$.
  Set $time \leftarrow time + t_i$.
end for
return sum.

Pretty Good Proportion

Problem

Given a string of length n consisting of 1s and 0s and a real number $0 \leq r \leq 1$. Find the substrings whose ratio of number of 1's to it's length is closest to r. Output the starting index of the substring. If there are multiple such substrings, return the one with smallest starting index.

Pretty Good Proportion

Sample Input

5 (test case)
12 0.666667
001001010111
11 0.400000
100001000011
3 0.000000
101

Sample Output

5
5
1

Pretty Good Proportion

Sample Input

5 (test case)
12 0.666667
001001010111
11 0.400000
100001000011
3 0.000000
101

Sample Output

5
5
1

Pretty Good Proportion

Finding ratio exactly equal to r

We try to solve the easier version. We want to find if there is any substring whose ratio is exactly equal to r.

Consider the prefix sum of the string. Let $pre[i]$ be the number of 1's in the substring from index 1 to i.

Then, the ratio of the substring $s[i+1, j]$ is given by $\frac{pre[j]-pre[i]}{j-i}$.

We want,

$$\frac{pre[j] - pre[i]}{j - i} = r$$

$$\implies pre[j] - pre[i] = r \cdot (j - i)$$

$$\implies r \cdot i - pre[i] = r \cdot j - pre[j]$$

Pretty Good Proportion

Finding ratio exactly equal to r

We try to solve the easier version. We want to find if there is any substring whose ratio is exactly equal to r.

Consider the prefix sum of the string. Let $pre[i]$ be the number of 1's in the substring from index 1 to i.

Then, the ratio of the substring $s[i+1, j]$ is given by $\frac{pre[j]-pre[i]}{j-i}$.

We want,

$$\frac{pre[j]-pre[i]}{j-i} = r$$

$$\implies pre[j]-pre[i] = r \cdot (j-i)$$

$$\implies r \cdot i - pre[i] = r \cdot j - pre[j]$$

Pretty Good Proportion

Finding ratio exactly equal to r

We try to solve the easier version. We want to find if there is any substring whose ratio is exactly equal to r.

Consider the prefix sum of the string. Let $pre[i]$ be the number of 1's in the substring from index 1 to i.

Then, the ratio of the substring $s[i+1, j]$ is given by $\frac{pre[j]-pre[i]}{j-i}$.

We want,

$$\frac{pre[j] - pre[i]}{j - i} = r$$

$$\implies pre[j] - pre[i] = r \cdot (j - i)$$

$$\implies r \cdot i - pre[i] = r \cdot j - pre[j]$$

Pretty Good Proportion

Converting to a slope problem

Consider each index as points $p_i = (i, r \cdot i - pre[i])$.

Then distance between ratio of substring $s[i+1, j]$ and r is given by,

$$\begin{aligned}
&\left| r - \frac{pre[j] - pre[i]}{j - i} \right| \\
&= \left| \frac{r(j-i) - (pre[j] - pre[i])}{j - i} \right| \\
&= |\text{slope}(p_i, p_j)|
\end{aligned}$$

Therefore, we need to minimize the slopes.

Pretty Good Proportion

Converting to a slope problem

Consider each index as points $p_i = (i, r \cdot i - pre[i])$.

Then distance between ratio of substring $s[i+1, j]$ and r is given by,

$$\begin{aligned}
&\left| r - \frac{pre[j] - pre[i]}{j - i} \right| \\
&= \left| \frac{r(j-i) - (pre[j] - pre[i])}{j - i} \right| \\
&= |\text{slope}(p_i, p_j)|
\end{aligned}$$

Therefore, we need to minimize the slopes.

Pretty Good Proportion

Theorem

If we sort all points by increasing order of 2nd element, then the closest pair of points will be adjacent in the sorted array.

Proof.

Assume that minimum is achived by some points (i, y_i) and (j, y_j) where $i < j$ but there exists point (k, y_k) such that $y_i < y_k < y_j$. Denote the slopes as

$$s_{ij} = \frac{y_j - y_i}{j - i}$$

$$s_{ik} = \frac{y_k - y_i}{k - i}$$

$$s_{kj} = \frac{y_j - y_k}{j - k}$$

□

Pretty Good Proportion

Theorem

If we sort all points by increasing order of 2nd element, then the closest pair of points will be adjacent in the sorted array.

Proof.

Assume that minimum is achived by some points (i, y_i) and (j, y_j) where $i < j$ but there exists point (k, y_k) such that $y_i < y_k < y_j$. Denote the slopes as

$$s_{ij} = \frac{y_j - y_i}{j - i}$$
$$s_{ik} = \frac{y_k - y_i}{k - i}$$
$$s_{kj} = \frac{y_j - y_k}{j - k}$$

□

Pretty Good Proportion

Proof.

One can show that,

$$s_{ij} = \frac{k-i}{j-i} \cdot s_{ik} + \frac{j-k}{j-i} \cdot s_{kj}$$

I.e s_{ij} is weighted average of s_{ik} and s_{kj}.

But weighted average of two real numbers must be between them. Hence,

$$\begin{aligned} &\min(s_{ik}, s_{kj}) < s_{ij} < \max(s_{ik}, s_{kj}) \\ \implies &\min(|s_{ik}|, |s_{kj}|) < |s_{ij}| \end{aligned}$$

Therefore, s_{ij} cannot be minimum. $\square$

Pretty Good Proportion

Proof.

One can show that,

$$s_{ij} = \frac{k-i}{j-i} \cdot s_{ik} + \frac{j-k}{j-i} \cdot s_{kj}$$

I.e s_{ij} is weighted average of s_{ik} and s_{kj}.

But weighted average of two real numbers must be between them. Hence,

$$\begin{aligned} &\min(s_{ik}, s_{kj}) < s_{ij} < \max(s_{ik}, s_{kj}) \\ \implies &\min(|s_{ik}|, |s_{kj}|) < |s_{ij}| \end{aligned}$$

Therefore, s_{ij} cannot be minimum. $\square$

Pretty Good Proportion

Proof.

One can show that,

$$s_{ij} = \frac{k-i}{j-i} \cdot s_{ik} + \frac{j-k}{j-i} \cdot s_{kj}$$

I.e s_{ij} is weighted average of s_{ik} and s_{kj}.
But weighted average of two real numbers must be between them. Hence,

$$\begin{aligned} &\min(s_{ik}, s_{kj}) < s_{ij} < \max(s_{ik}, s_{kj}) \\ \implies &\min(|s_{ik}|, |s_{kj}|) < |s_{ij}| \end{aligned}$$

Therefore, s_{ij} cannot be minimum. $\square$

Pretty Good Proportion

Algorithm 7 **Closest**(n, s)

Set $pre[n+1] \leftarrow$ The prefix sum of s.

Set $points[n+1] \leftarrow (i, r \cdot i - pre[i])$ for $i = 0$ to n.

Sort *points* in increasing order of 2nd element. Break ties by 1st element.

$dist \leftarrow \infty$

$ans \leftarrow 0$

for $i = 0$ to $n-1$ **do**

  if **then** $\frac{points[i].second - points[i+1].second}{points[i].first - points[i+1].first} < dist$

    $dist \leftarrow \frac{points[i].second - points[i+1].second}{points[i].first - points[i+1].first}$

    $ans \leftarrow points[i].first$

  end if

end for

return *ans*