Tänk Girigt

Tillämpad Algoritmisk Problemlösning (Våren 2026)

Martin Berglund, mbe@cs.umu.se

Biblioteksgrunderna 1

Vissa saker tar man *alltid* ur bibliotek, många nästan *aldrig*...

Sortering: QuickSort

- Eller rättare sagt: vad som finns i bibliotek, för C t.ex. qsort
- Ibland ad-hoc nästan osynlig counting/bucket/radix-sort

```
# count occurrences of numbers in vs
assert min(vs)>=0
counts=[0]*max(vs)
for v in vs:
    counts[v]+=1
```

Biblioteksgrunderna 1

Vissa saker tar man *alltid* ur bibliotek, många nästan *aldrig*...
Sortering: QuickSort

- Eller rättare sagt: vad som finns i bibliotek, för C t.ex. qsort
- Ibland ad-hoc nästan osynlig counting/bucket/radix-sort

Mappningar: Någon hashtabell (`java.util.HashMap`, Python `Dict`, C++ `std::map`)

- Ordnade mappningar ibland meningsfullt
- Mappar med "defaults" ibland förenklande (aggregering)

Mängder: Hashset (`Java.util.HashSet`, Python `set`, C++ `std::set`)

- Ibland ad-hoc nästan osynliga bitvektorer (ofta fallet i C)

Biblioteksgrunderna 2

Sekvenser: Främst växande arrayer (`java.util.ArrayList`, Python `list`, C++ `std::vector`)

- Att lägga till ett element kan ta $\mathcal{O}(n)$, men amorterat $\mathcal{O}(1)$
- Strikt $\mathcal{O}(n)$ på att lägga till på annat än slutet
- I tävling och prototyp räcker ofta en vanlig array (allokera generöst)

Fritt modifierade sekvenser:

- T.ex. Python `deque` för amorterad $\mathcal{O}(1)$ att lägga till/ta bort första elementet
- Länkade listor ger $\mathcal{O}(n)$ åtkomst, men kan modifieras fritt (`java.util.LinkedList`, Python existerar ej, C++ `std::list`)
- Sällan nödvändigt! Amortera själv med markeringar i array

Ditt eget bibliotek

Andra enkla datatyper är ofta relevanta: t.ex. *stackar*, *köer*, *sorterade sekvenser* (med binärsökning)

- Ofta i bibliotek `java.util.Queue`, `std::stack`, etc.
- Dock så enkla att de ofta blir ad-hoc i större sammanhang

Algoritmer bortom sortering:

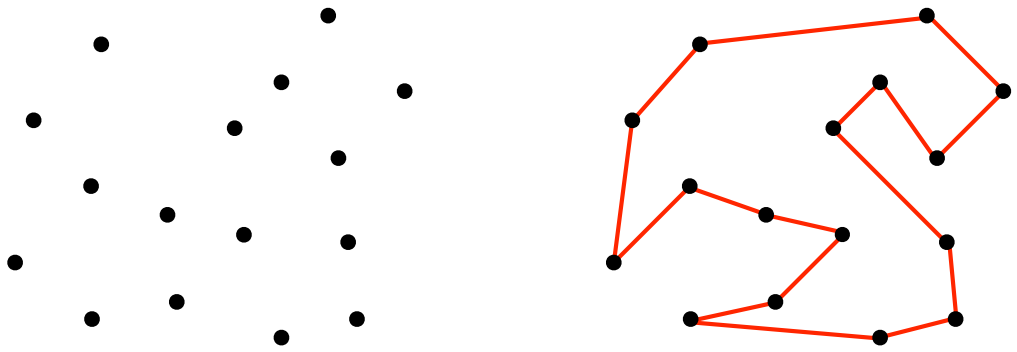
- I prototyp och nöjesprogrammering:
  - ta från ditt eget bibliotek av snippets, eller
  - skriv från grunden.
- I industri/forskning:
  - ta från projektets egna bibliotek,
  - skriv från grunden utökandes projektets bibliotek, eller
  - tredjepartsbibliotek? Ofta för numerik, sällan för grafteori

Tredjepartsbibliotek ibland inte applicerbart! Enklare implementera bredden-först-sökning än att försätta data på graf-format.

För mer avancerade datastrukturer (t.ex. väl implementerade *sorterade träd*, *skip-listor*) är (snippet)bibliotek typiskt lösningen

Suppose you have a robot arm equipped with a tool, say a soldering iron. To enable the robot arm to do a soldering job, we must construct an ordering of the contact points, so the robot visits (and solders) the points in order.
We seek the order which minimizes the testing time (i.e. travel distance) it takes to assemble the circuit board.

Hitta kortaste vägen för roboten



You are given the job to program the robot arm. Give me an algorithm to find the most efficient tour!

A popular solution starts at some point p_0 and then walks to its nearest neighbor p_1 first, then repeats from p_1, etc. until done.

Pick and visit an initial point p_0

$p = p_0$

$i = 0$

While there are still unvisited points

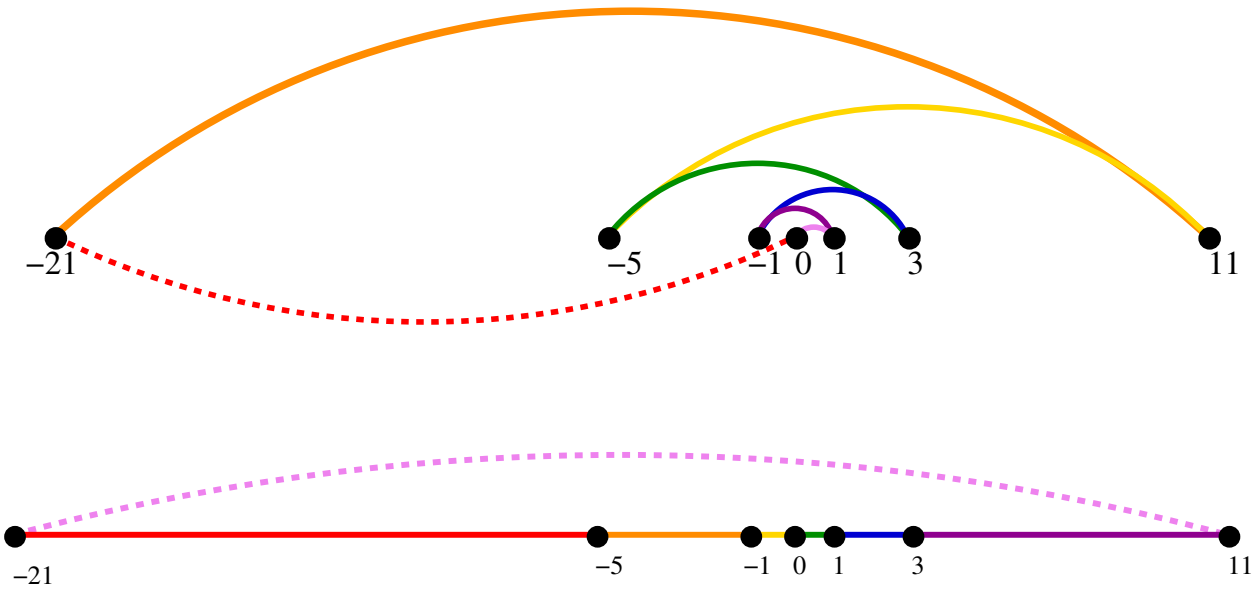
    $i = i + 1$

    Let p_i be the closest unvisited point to p_{i-1}

    Visit p_i

Return to p_0 from p_i

Närmaste granne fungerar ej!



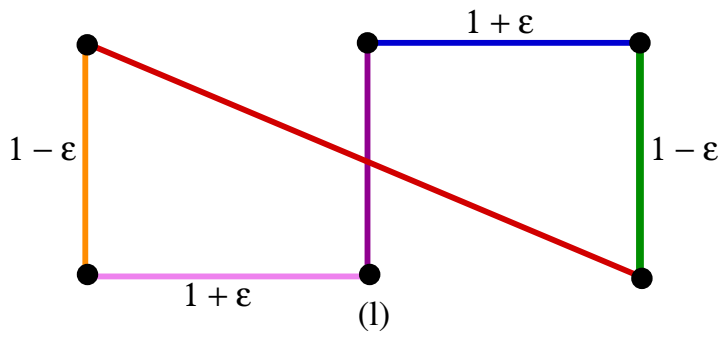
Starting from the leftmost point will not fix the problem.

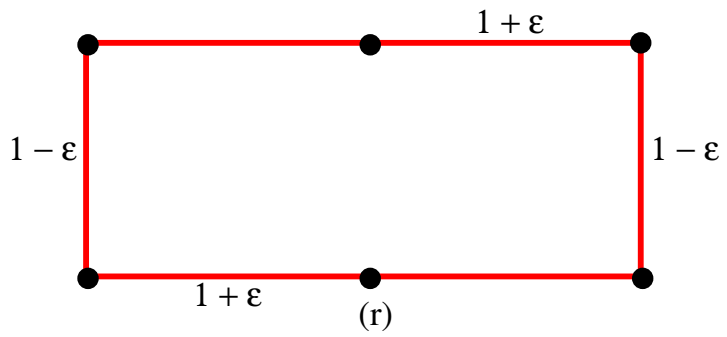
Another idea is to repeatedly connect the closest pair of points whose connection will not cause a cycle or a three-way branch, until all points are in one tour.

Let n be the number of points in the set
For $i = 1$ to $n - 1$ do
    $d = \infty$
    For each pair of endpoints (x, y) of partial paths
        If $dist(x, y) \leq d$ then
            $x_m = x$, $y_m = y$, $d = dist(x, y)$
    Connect (x_m, y_m) by an edge
Connect the two endpoints by an edge.

Närmaste par fungerar ej!

Although it works correctly on the previous example, other data causes trouble:





En *korrekt* algoritm: *uttömmande sökning*

We could try all possible orderings of the points, then select the one which minimizes the total length:

$d = \infty$
For each of the $n!$ permutations Π_i of the n points
    If $(cost(\Pi_i) \leq d)$ then
        $d = cost(\Pi_i)$ and $P_{min} = \Pi_i$
Return P_{min}

Since all possible orderings are considered, we are guaranteed to end up with the shortest possible tour.

Because it tries all $n!$ permutations, it is much too slow to use when there are more than 10-20 points.

No efficient, correct algorithm exists for the *traveling salesman problem*, as we will see later.

Vi återkommer mycket riktigt till TSP. Är NP-komplett, Skiena uttrycker sig slarvigt, vi *känner inte till* någon snabb korrekt algoritm.

$$10! = 3\,628\,800$$
$$20! = 2\,432\,902\,008\,176\,640\,000$$

Uttömmande tar tid...

$$
\begin{aligned}
10! &= 3\,628\,800 \\
20! &= 2\,432\,902\,008\,176\,640\,000
\end{aligned}
$$

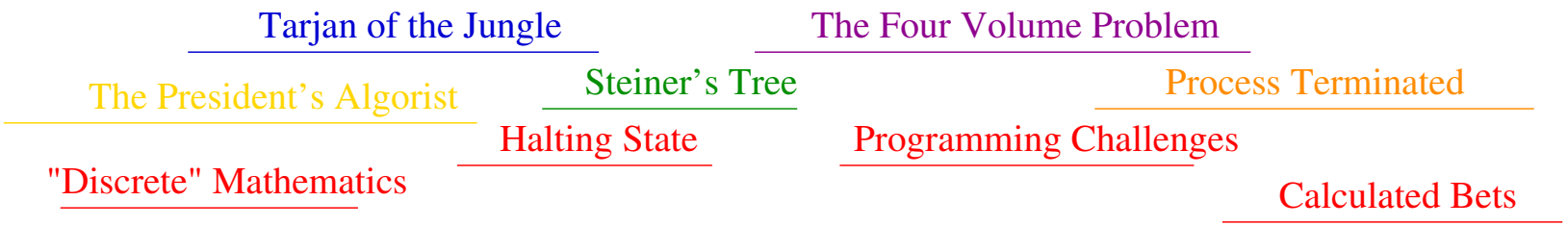
Återkommande typ av överslag:

- antag 1GHz processor (lite lågt, men se nästa)
- antag att *"ett steg"* av algoritmen görs varje klockcykel (extremt generöst! här gör vi minst $\mathcal{O}(n)$ mycket jobb)
- antag vi vill ha ett svar inom överskådlig tid...

$$
\frac{2\,432\,902\,008\,176\,640\,000}{1\,000\,000\,000 * 86\,400} > 2\,433 \text{ dagar} > 66 \text{ år}
$$

Schemalägga jobb

A movie star wants to the select the maximum number of staring roles such that no two jobs require his presence at the same time.



Problement mer formellt

Input: A set I of n intervals on the line.
Output: What is the largest subset of mutually non-overlapping intervals which can be selected from I?
Give an algorithm to solve the problem!

Girigt välja första tillgängliga?

Start working as soon as there is work available:

EarliestJobFirst(I)

Accept the earlest starting job j from I which does not overlap any previously accepted job, and repeat until no more such jobs remain.

Första tillgängliga fungerar ej!

The first job might be so long (War and Peace) that it prevents us from taking any other job.

Girigt välja korta jobb?

Always take the shortest possible job, so you spend the least time working (and thus unavailable).

ShortestJobFirst(I)

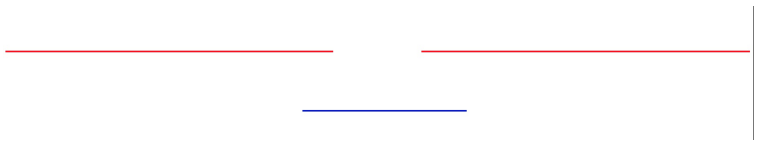
While $(I \neq \emptyset)$ do

Accept the shortest possible job j from I.

Delete j, and intervals which intersect j from I.

Girighet verkar allmänt sämst på allt

Taking the shortest job can prevent us from taking two longer jobs which barely overlap it.



Take the job with the earliest completion date:

OptimalScheduling(I)

    While $(I \neq \emptyset)$ do

        Accept job j with the earliest completion date.

        Delete j, and whatever intersects j from I.

Proof: Other jobs may well have started before the first to complete (say, x), but all must at least partially overlap both x and each other.
Thus we can select at most one from the group.
The first these jobs to complete is x, so selecting any job but x would only block out more opportunties after x.

Hur lång tid tar lösningen?

Take the job with the earliest completion date:

OptimalScheduling(I)
    While $(I \neq \emptyset)$ do
        Accept job j with the earliest completion date.
        Delete j, and whatever intersects j from I.

Själva girigheten: linjär, gå igenom jobben i 'completion date'-ordning och acceptera de som börjar när man är ledig. $\mathcal{O}(n)$

Input-ordningen dock inte specificerad, *sortering efter 'completion date':* $\mathcal{O}(n \log n)$

Baloni

Submit | Stats

Problem ID: baloni

CPU Time limit: 4 seconds

Memory limit: 1024 MB

Difficulty: 3.0

Download:
Sample data files

Author: Dominik Gleich

Source: Croatian Open Competition in Informatics 2015/2016, contest #1

License: For educational use only

There are N balloons floating in the air in a large room, lined up from left to right. Young Perica likes to play with arrows and practice his hunting abilities. He shoots an arrow from the left to the right side of the room from an arbitrary height he chooses. The arrow moves from left to right, at a chosen height H until it finds a balloon. The moment when an arrow touches a balloon, the balloon pops and disappears and the arrow continues its way from left to right at a height decreased by 1. Therefore, if the arrow was moving at height H, after popping the balloon it travels on height $H - 1$.

Our hero's goal is to pop all the balloons using as little arrows as possible.

Input

The first line of input contains the integer N ($1 \leq N \leq 1\ 000\ 000$). The second line of input contains an array of N integers H_i. Each integer H_i ($1 \leq H_i \leq 1\ 000\ 000$) is the height at which the i-th balloon floats, respectively from left to right.

Output

The first and only line of output must contain the minimal number of times Pero needs to shoot an arrow so that all balloons are popped.

Sample Input 1

```
5
2 1 5 4 3
```

Sample Output 1

```
2
```

Sample Input 2

```
5
1 2 3 4 5
```

Sample Output 2

```
5
```

Sample Input 3

```
5
4 5 2 1 4
```

Sample Output 3

```
3
```

Lite boilerplate

```
import sys
import pdb

if sys.stdin.isatty():
    with open("baloni.1.in") as f:
        inp=f.readlines()
else:
    inp=sys.stdin.readlines()

try:
    inp=[[int(v.strip()) for v in l.split()]
        for l in inp]
except:
    pass
```

Vi tar det lite lugnt

```
bal=inp[1]
popped=[False]*len(bal)          # balloons initially unpopped
left_to_pop=len(bal)
arrows=0

while left_to_pop>0:
    for i in range(len(bal)):
        if popped[i]:            # find first unpopped
            continue
        h=bal[i]-1               # fire arrow
        arrows+=1
        left_to_pop-=1
        popped[i]=True
        for j in range(i+1,len(bal)):
            if not popped[j] and bal[j]==h:
                h-=1
                left_to_pop-=1
                popped[j]=True

print(f'{arrows}')
```

Vi tar det lite lugnt

```
bal=inp[1]
popped=[False]*len(bal)          # balloons initially unpopped
left_to_pop=len(bal)
arrows=0

while left_to_pop>0:
    for i in range(len(bal)):
        if popped[i]:                    # find first unpopped
            popped[i]=True
            for j in range(i+1,len(bal)):
                if not popped[j] and bal[j]==h:
                    h-=1
                    left_to_pop-=1
                    popped[j]=True

print(f'{arrows}')
```

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8910188	16:27:24	Baloni	✖ Time Limit Exceeded	> 4.00 s	Python 3

$\Longrightarrow$ Fixa med länkade listor? Nej, vår 'fejk-delete' är inte problemet!

Gör ett överslag

Borde räknat grovt från början: Kattis informerar oss att vi får upp till *1 000 000 ballonger* och *4 sekunder*

Vad är *värsta fall?* Enkelt, *1 000 000 pilar!* Resultatet blir att de två looparna nästar till $1\,000\,000^2 = 10^{12}$ 'steg'

Låt oss säga *1GHz* och att innehållet i inre loopen tar *1 cykel*:

$$\frac{10^{12} \text{ steg}}{10^9 \text{ 'steg' i sekunden}} = 1\,000 \text{ sekunder.}$$

inte ens nära de *4 sekunderna...*

Gör ett överslag

Borde räknat grovt från början: Kattis informerar oss att vi får upp till *1 000 000 ballonger* och *4 sekunder*

Vad är *värsta fall?* Enkelt, *1 000 000 pilar!* Resultatet blir att de två looparna nästar till $1\,000\,000^2 = 10^{12}$ 'steg'

Låt oss säga *1GHz* och att innehållet i inre loopen tar *1 cykel*:

$$\frac{10^{12} \text{ steg}}{10^9 \text{ 'steg' i sekunden}} = 1\,000 \text{ sekunder.}$$

inte ens nära de *4 sekunderna...*

Om vi *tar bort* poppade ballonger *ordentligt?* För värsta-fall-input eliminerar det *hälften* av stegen, men 500 sekunder fortfarande inte nära! 'Mikrooptimering' som inte undviker vår $\mathcal{O}(n^2)$

Lösningsidé: spåra alla pilar samtidigt, gå igenom ballongerna endast *en gång*

Vi gör lite fel

```
arrows=set()
launched=0

for bal in inp[1]:
    if not bal in arrows:
        arrows.add(bal-1)
        launched+=1
    else:
        arrows.remove(bal)
        arrows.add(bal-1)

print(f'{launched}')
```

Vi gör lite fel

```
arrows=set()
launched=0

for bal in inp[1]:
    if not bal in arrows:
        arrows.add(bal-1)
        launched+=1
    else:
        arrows.remove(bal)
        arrows.add(bal-1)

print(f'{launched}')
```

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8910208	16:32:18	Baloni	✖ Wrong Answer	0.10 s	Python 3
	✔✔✔✖				

$\Longrightarrow$ Vad gjorde jag nu fel? OK på de tre testfallen...

Vi gör mer rätt

```
from collections import defaultdict

arrows=defaultdict(lambda: 0)
launched=0

for bal in inp[1]:
    if arrows[bal]>0:
        arrows[bal]-=1
        arrows[bal-1]+=1
    else:
        launched+=1
        arrows[bal-1]+=1

print(f'{launched}')
```

Vi gör mer rätt

```
from collections import defaultdict

arrows=defaultdict(lambda: 0)
launched=0

for bal in inp[1]:
    if arrows[bal]>0:
        arrows[bal]-=1
        arrows[bal-1]+=1
    else:
        launched+=1
        arrows[bal-1]+=1

print(f'{launched}')
```

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8910335	17:01:27	Baloni	✔ Accepted	0.29 s	Python 3
	✔✔✔✔✔✔✔✔✔✔✔✔✔✔				

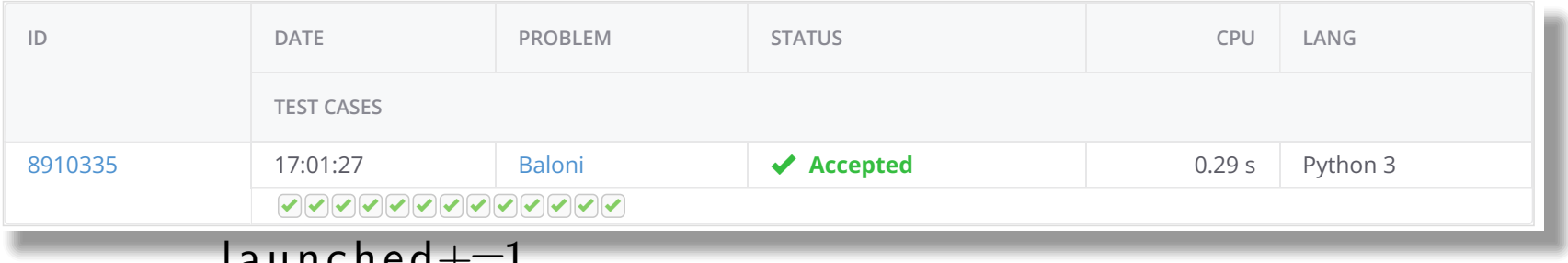
$\Longrightarrow$ Succé!

Vi gör mer rätt (utan defaultdict)

```
arrows={}
launched=0

for bal in inp[1]:
    if bal in arrows.keys() and arrows[bal]>0:
        arrows[bal]-=1
        if (bal-1) in arrows.keys():
            arrows[bal-1]+=1
        else:
            arrows[bal-1]=1
    else:
        launched+=1
        if (bal-1) in arrows.keys():
            arrows[bal-1]+=1
        else:
            arrows[bal-1]=1

print(f'{launched}')
```

Vi gör mer rätt (utan defaultdict)

```
arrows={}
launched=0

for bal in inp[1]:
    if bal in arrows.keys() and arrows[bal]>0:
        arrows[bal]-=1
        if (bal-1) in arrows.keys():
```



ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8910335	17:01:27	Baloni	✔ Accepted	0.29 s	Python 3

```
            launched+=1
        if (bal-1) in arrows.keys():
            arrows[bal-1]+=1
        else:
            arrows[bal-1]=1

print(f'{launched}')
```

$\Longrightarrow$ Succé!

“Giriga” algoritmer

Detta är kärnan i en *girig* algoritm:

- *Inte* egentligen en *lösningsstrategi,* mer en ofta återkommande *egenskap* hos naturliga problem
- Lös naturliga “delproblem” i ordning löser hela problemet
  $\Rightarrow$ skjut helt enkelt första ballongen
- Jämför *dynamisk programmering* senare!

"Giriga" algoritmer

Detta är kärnan i en *girig* algoritm:

- *Inte* egentligen en *lösningsstrategi,* mer en ofta återkommande *egenskap* hos naturliga problem
- Lös naturliga "delproblem" i ordning löser hela problemet
  $\Rightarrow$ skjut helt enkelt första ballongen
- Jämför *dynamisk programmering* senare!

Ett exempel till:

Coloring Socks

Having discolored his white socks in a rather beige shade (as seen on the picture), Luktas Svettocek realised he can't just throw all his laundry into one machine and expect it to retain its original colors. However, he is also too lazy to do his laundry in several rounds. He would much rather buy more laundry machines!

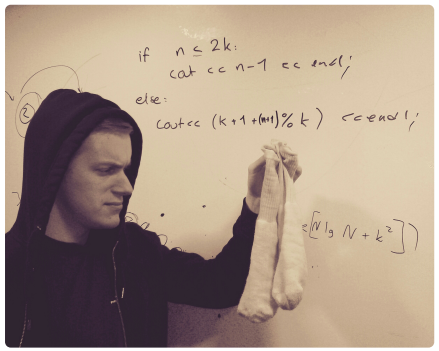


Photo by Johan Sannemo and Oskar Werkelin Ahlin

Each of Luktas' socks have a color D_i which has a number between 0 and 10^9 assigned to it. After some experimentation, he found that he could wash any socks with a maximum absolute color difference of K in the same machine without any discoloring. The color difference of two socks i and j is $|D_i - D_j|$.

Luktas now needs to know how many washing machines he needs to wash his S socks, given that each machine can take at most C socks a time.

Input

The first line consists of three integers $1 \leq S, C \leq 10^5$ and $0 \leq K \leq 10^9$, the number of socks, the capacity of a

Submit | Stats

My Submissions

Problem ID: color

CPU Time limit: 1 second

Memory limit: 1024 MB

Difficulty: 2.2

Download:
Sample data files

Authors: Johan Sannemo and Oskar Werkelin Ahlin

Source: KTH Training

License:

Coloring Socks

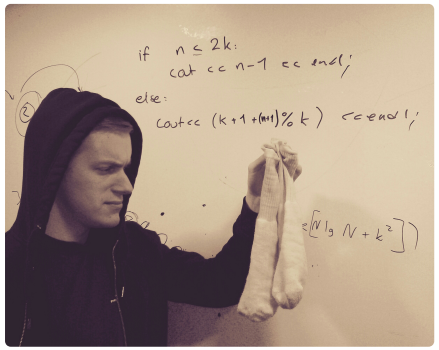


Photo by Johan Sannemo and Oskar Werkelin Ahlin

Having discolored his white socks in a rather beige shade (as seen on the picture), Luktas Svettocek realised he can't just throw all his laundry into one machine and expect it to retain its original colors. However, he is also too lazy to do his laundry in several rounds. He would much rather buy more laundry machines!

Each of Luktas' socks have a color D_i which has a number between 0 and 10^9 assigned to it. After some experimentation, he found that he could wash any socks with a maximum absolute color difference of K in the same machine without any discoloring. The color difference of two socks i and j is $|D_i - D_j|$.

Luktas now needs to know how many washing machines he needs to wash his S socks, given that each machine can take at most C socks a time.

Input

The first line consists of three integers $1 \leq S, C \leq 10^5$ and $0 \leq K \leq 10^9$, the number of socks, the capacity of a laundry machine and the maximum color difference, respectively. Then follow one line with S integers; these are the color values D_i of every sock.

Output

Output a single integer; the number of machines Luktas needs to wash all his socks.

Sample Input 1

```
5 3 0
0 0 1 1 2
```

Sample Output 1

```
3
```

Sample Input 2

```
5 3 1
0 0 1 1 2
```

Sample Output 2

```
2
```

Submit | Stats

My Submissions

Problem ID: color

CPU Time limit: 1 second

Memory limit: 1024 MB

Difficulty: 2.2

Download:
Sample data files

Authors: Johan Sannemo and Oskar Werkelin Ahlin

Source: KTH Training

Girighet vinner åtminstone på kort sikt!

1. Fyll påbörjad maskin med de ljusaste strumporna
2. När full eller för mörk strumpa, börja en ny maskin!

```
s,c,k=inp[0]
socks=inp[1]

machines=0
start_color=None
space_used=c

for sock_color in socks:
    if space_used>=c:
        machines+=1
        start_color=sock_color
        space_used=0
    if sock_color-start_color>k:
        machines+=1
        start_color=sock_color
        space_used=0
    space_used+=1

print(f'{machines}')
```

Girighet vinner åtminstone på kort sikt!

1. Fyll påbörjad maskin med de ljusaste strumporna
2. När full eller för mörk strumpa, börja en ny maskin!

```
s,c,k=inp[0]
socks=inp[1]

machines=0
start_color=None
```

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8997594	20:44:14	Coloring Socks	✖ Wrong Answer	0.12 s	Python 3

```
        machines+=1
        start_color=sock_color
        space_used=0
    if sock_color-start_color>k:
        machines+=1
        start_color=sock_color
        space_used=0
    space_used+=1

print(f'{machines}')
```

Girighet vinner åtminstone på kort sikt!

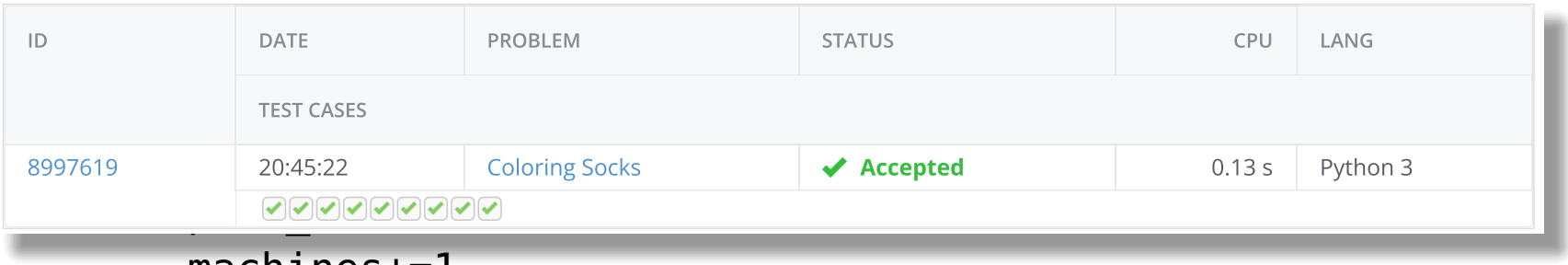
1. Fyll påbörjad maskin med de ljusaste strumporna
2. När full eller för mörk strumpa, börja en ny maskin!

```
s,c,k=inp[0]
socks=sorted(inp[1])

machines=0
start_color=None
space_used=c

for sock_color in socks:
    if space_used>=c:
        machines+=1
        start_color=sock_color
        space_used=0
    if sock_color-start_color>k:
        machines+=1
        start_color=sock_color
        space_used=0
    space_used+=1

print(f'{machines}')
```

Girighet vinner åtminstone på kort sikt!

1. Fyll påbörjad maskin med de ljusaste strumporna
2. När full eller för mörk strumpa, börja en ny maskin!

```
s,c,k=inp[0]
socks=sorted(inp[1])

machines=0
start_color=None
```



ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8997619	20:45:22	Coloring Socks	✔ Accepted	0.13 s	Python 3

```
            machines+=1
            start_color=sock_color
            space_used=0
        if sock_color-start_color>k:
            machines+=1
            start_color=sock_color
            space_used=0
        space_used+=1

print(f'{machines}')
```

Körtid och korrekthet

Samma mönster som filmstjärnans jobbplanering:

- Girigheten direkt linjär promenad genom input, $\mathcal{O}(n)$
- Dock! Måste sorteras först, så $\mathcal{O}(n \log n)$ ($\Theta(n \log n)$?)

Men är algoritmen *korrekt*, och alltså *optimal?*

Bevis.

Den mörkaste strumpan måste vara i *någon* maskin.

- Om denna maskin *inte är full* innehåller den alla strumpor som kan kombineras med den mörkaste, och vi kan inte göra något bättre för maskinen med den mörkaste strumpan.
- Om denna maskin *är full* är innehållet de mörkaste tillgängliga strumporna. Detta är optimalt, då det gör den mörkaste 'kvarvarande' strumpan så ljus som möjligt.

Denna maskin är alltså optimalt fylld. Ta bort maskinen och dess strumpor ur problemet. Om det finns strumpor kvar, upprepa ovanstående argument med den mörkaste *kvarvarande* strumpan för 'nästa' maskin. □