

Grunnatriði og Ad hoc

Bergur Snorrason

22. janúar 2021

Grunntög og takmarkanir þeirra

- ▶ Í grunninn snýst forritun um gögn.

Grunntög og takmarkanir þeirra

- ▶ Í grunninn snýst forritun um gögn.
- ▶ Þegar við forritum flokkum við gögnin okkar með *tögum*.

Grunntög og takmarkanir þeirra

- ▶ Í grunninn snýst forritun um gögn.
- ▶ Þegar við forritum flokkum við gögnin okkar með *tögum*.
- ▶ Dæmi um tög í C/C++ eru `int` og `double`.

Grunntög og takmarkanir þeirra

- ▶ Í grunninn snýst forritun um gögn.
- ▶ Þegar við forritum flokkum við gögnin okkar með *tögum*.
- ▶ Dæmi um tög í C/C++ eru `int` og `double`.
- ▶ Helstu tög in í C/C++ eru (yfirleitt):

Grunntög og takmarkanir þeirra

- ▶ Í grunninn snýst forritun um gögn.
- ▶ Þegar við forritum flokkum við gögnin okkar með *tögum*.
- ▶ Dæmi um tög í C/C++ eru `int` og `double`.
- ▶ Helstu tög in í C/C++ eru (yfirleitt):

Heiti	Lýsing	Skorður
<code>int</code>	Heiltala	Á bilinu $[-2^{31}, 2^{31} - 1]$
<code>unsigned int</code>	Heiltala	Á bilinu $[0, 2^{32} - 1]$
<code>long long</code>	Heiltala	Á bilinu $[-2^{63}, 2^{63} - 1]$
<code>unsigned long long</code>	Heiltala	Á bilinu $[0, 2^{64} - 1]$
<code>double</code>	Fleytitala	Takmörkuð nákvæmni
<code>char</code>	Heiltala	Á bilinu $[-128, 127]$

Hvað með tölur utan þessa bila?

- ▶ Einn helsti kostur Python í keppnisforritun er að heiltölur geta verið eins stórar (eða litlar) og vera skal.

Hvað með tölur utan þessa bila?

- ▶ Einn helsti kostur Python í keppnisforritun er að heiltölur geta verið eins stórar (eða litlar) og vera skal.

```
from math import factorial  
print(factorial(100))
```


Hvað með tölur utan þessa bila?

- ▶ Einn helsti kostur Python í keppnisforritun er að heiltölur geta verið eins stórar (eða litlar) og vera skal.

```
from math import factorial  
print(factorial(100))
```

```
93326215443944152681699238856266700490715968264381621  
46859296389521759999322991560894146397615651828625369  
7920827223758251185210916864000000000000000000000000000
```

Hvað með tölur utan þessa bila?

- ▶ Einn helsti kostur Python í keppnisforritun er að heiltölur geta verið eins stórar (eða litlar) og vera skal.

```
from math import factorial  
print(factorial(100))
```

```
93326215443944152681699238856266700490715968264381621  
46859296389521759999322991560894146397615651828625369  
7920827223758251185210916864000000000000000000000000000
```

- ▶ Það er einnig hægt að nota `fractions` þakkann í Python til að vinna með fleytitölur án þess að tapa nákvæmni.

Hvað með tölur utan þessa bila?

- Sumir C/C++ þýðendur bjóða upp á gagnatagið `__int128` (til dæmis gcc).

Hvað með tölur utan þessa bila?

- ▶ Sumir C/C++ þýðendur bjóða upp á gagnatagið `__int128` (til dæmis `gcc`).
- ▶ Þetta tag býður upp á að nota tölur á bilinu $[-2^{127}, 2^{127} - 1]$.

Hvað með tölur utan þessa bila?

- ▶ Sumir C/C++ þýðendur bjóða upp á gagnatagið `__int128` (til dæmis `gcc`).
- ▶ Þetta tag býður upp á að nota tölur á bilinu $[-2^{127}, 2^{127} - 1]$.
- ▶ Þetta þarf ekki að nota oft.

- ▶ Við munum reglulega þurfa að raða gögnum í einhverja röð.

- ▶ Við munum reglulega þurfa að raða gögnum í einhverja röð.

Forritunarmál	Röðun
---------------	-------

C	<code>qsort(...)</code>
---	-------------------------

C++	<code>sort(...)</code>
-----	------------------------

Python	<code>this.sort()</code> eða <code>sorted(...)</code>
--------	---

- ▶ Við munum reglulega þurfa að raða gögnum í einhverja röð.

Forritunarmál	Röðun
C	<code>qsort(...)</code>
C++	<code>sort(...)</code>
Python	<code>this.sort()</code> eða <code>sorted(...)</code>

- ▶ Skoðum nú hvert forritunarmál til að sjá nánar hvernig föllin eru notuð.

Röðun í C++

- ▶ Í grunninn tekur `sort(...)` við tveimur gildum.

Röðun í C++

- ▶ Í grunninn tekur `sort(...)` við tveimur gildum.
- ▶ Fyrri gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)

Röðun í C++

- ▶ Í grunninn tekur `sort(...)` við tveimur gildum.
- ▶ Fyrri gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)
- ▶ Ef við erum með n staka fylki a þá röðum við því með `sort(a, a + n)`.

Röðun í C++

- ▶ Í grunninn tekur `sort(...)` við tveimur gildum.
- ▶ Fyrri gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)
- ▶ Ef við erum með n staka fylki a þá röðum við því með `sort(a, a + n)`.
- ▶ Við getum raðað flest öllum ílátum með `sort`.

Röðun í C++

- ▶ Í grunninn tekur `sort(...)` við tveimur gildum.
- ▶ Fyrri gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)
- ▶ Ef við erum með n staka fylki a þá röðum við því með `sort(a, a + n)`.
- ▶ Við getum raðað flest öllum ílátum með `sort`.
- ▶ Ef við erum með eitthva ílát (til dæmis `vector`) a má raða með `sort(a.begin(), a.end())`.

Röðun í C++

- ▶ Í grunninn tekur `sort(...)` við tveimur gildum.
- ▶ Fyrri gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)
- ▶ Ef við erum með n staka fylki `a` þá röðum við því með `sort(a, a + n)`.
- ▶ Við getum raðað flest öllum ílátum með `sort`.
- ▶ Ef við erum með eitthva ílát (til dæmis `vector`) `a` má raða með `sort(a.begin(), a.end())`.
- ▶ Við getum líka bætt við okkar eigin samanburðarfalli sem þriðja inntak.

Röðun í C++

- ▶ Í grunninn tekur `sort(...)` við tveimur gildum.
- ▶ Fyrri gildið svarar til fyrsta staks þess sem við viljum raða og seinna gildið vísar á enda þess sem við viljum raða (ekki síðasta stakið)
- ▶ Ef við erum með n staka fylki a þá röðum við því með `sort(a, a + n)`.
- ▶ Við getum raðað flest öllum ílátum með `sort`.
- ▶ Ef við erum með eitthva ílát (til dæmis `vector`) a má raða með `sort(a.begin(), a.end())`.
- ▶ Við getum líka bætt við okkar eigin samanburðarfalli sem þriðja inntak.
- ▶ Það kemur þá í stað “minna eða samasem” samanburðarins sem er sjálfgefinn.

Röðun í Python

- ▶ Til að raða lista í Python þá má nota annað hvort `this.sort()` eða `sorted(...)`.

Röðun í Python

- ▶ Til að raða lista í Python þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- ▶ Gerum ráð fyrir að listinn okkar heiti `a`.

Röðun í Python

- ▶ Til að raða lista í Python þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- ▶ Gerum ráð fyrir að listinn okkar heiti `a`.
- ▶ Þá nægir að kalla á `a.sort()` og eftir það er `a` raðað.

Röðun í Python

- ▶ Til að raða lista í Python þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- ▶ Gerum ráð fyrir að listinn okkar heiti `a`.
- ▶ Þá nægir að kalla á `a.sort()` og eftir það er `a` raðað.
- ▶ Hinsvegar skilar `sorted(a)` afriti af `a` sem hefur verið raðað.

Röðun í Python

- ▶ Til að raða lista í Python þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- ▶ Gerum ráð fyrir að listinn okkar heiti `a`.
- ▶ Þá nægir að kalla á `a.sort()` og eftir það er `a` raðað.
- ▶ Hinsvegar skilar `sorted(a)` afriti af `a` sem hefur verið raðað.
- ▶ Til að raða `a` á þennan hátt þarf `a = sorted(a)`.

Röðun í Python

- ▶ Til að raða lista í Python þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- ▶ Gerum ráð fyrir að listinn okkar heiti `a`.
- ▶ Þá nægir að kalla á `a.sort()` og eftir það er `a` raðað.
- ▶ Hinsvegar skilar `sorted(a)` afriti af `a` sem hefur verið raðað.
- ▶ Til að raða `a` á þennan hátt þarf `a = sorted(a)`.
- ▶ Nota má inntakið `key` til að raða eftir öðrum samanburðum.

Röðun í Python

- ▶ Til að raða lista í Python þá má nota annað hvort `this.sort()` eða `sorted(...)`.
- ▶ Gerum ráð fyrir að listinn okkar heiti `a`.
- ▶ Þá nægir að kalla á `a.sort()` og eftir það er `a` raðað.
- ▶ Hinsvegar skilar `sorted(a)` afriti af `a` sem hefur verið raðað.
- ▶ Til að raða `a` á þennan hátt þarf `a = sorted(a)`.
- ▶ Nota má inntakið `key` til að raða eftir öðrum samanburðum.
- ▶ Það er einnig inntak sem heitir `reverse` sem er Boole gildi sem leyfir auðveldlega að raða öfugt.

Röðun í C

- ▶ Í C er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.

Röðun í C

- ▶ Í C er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- ▶ Til röðunar notum við fallið `qsort(...)`.

Röðun í C

- ▶ Í C er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- ▶ Til röðunar notum við fallið `qsort(...)`.
- ▶ Fallið tekur fjögur viðföng:

Röðun í C

- ▶ Í C er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- ▶ Til röðunar notum við fallið `qsort(...)`.
- ▶ Fallið tekur fjögur viðföng:
 - ▶ `void* a`. Þetta er fylkið sem við viljum raða.

Röðun í C

- ▶ Í C er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- ▶ Til röðunar notum við fallið `qsort(...)`.
- ▶ Fallið tekur fjögur viðföng:
 - ▶ `void* a`. Þetta er fylkið sem við viljum raða.
 - ▶ `size_t n`. Þetta er fjöldi staka í fylkinu sem `a` svarar til.

Röðun í C

- ▶ Í C er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- ▶ Til röðunar notum við fallið `qsort(...)`.
- ▶ Fallið tekur fjögur viðföng:
 - ▶ `void* a`. Þetta er fylkið sem við viljum raða.
 - ▶ `size_t n`. Þetta er fjöldi staka í fylkinu sem `a` svarar til.
 - ▶ `size_t s`. Þetta er stærð hvers staks í fylkinu okkar (í bætum).

Röðun í C

- ▶ Í C er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- ▶ Til röðunar notum við fallið `qsort(...)`.
- ▶ Fallið tekur fjögur viðföng:
 - ▶ `void* a`. Þetta er fylkið sem við viljum raða.
 - ▶ `size_t n`. Þetta er fjöldi staka í fylkinu sem `a` svarar til.
 - ▶ `size_t s`. Þetta er stærð hvers staks í fylkinu okkar (í bætum).
 - ▶ `int (*cmp)(const void *, const void*)`. Þetta er samanburðarfallið okkar.

Röðun í C

- ▶ Í C er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- ▶ Til röðunar notum við fallið `qsort(...)`.
- ▶ Fallið tekur fjögur viðföng:
 - ▶ `void* a`. Þetta er fylkið sem við viljum raða.
 - ▶ `size_t n`. Þetta er fjöldi staka í fylkinu sem `a` svarar til.
 - ▶ `size_t s`. Þetta er stærð hvers staks í fylkinu okkar (í bætum).
 - ▶ `int (*cmp)(const void *, const void*)`. Þetta er samanburðarfallið okkar.
- ▶ Síðasta inntakið er kannski flókið við fyrstu sýn en er einfalt fyrir okkur að nota.

Röðun í C

- ▶ Í C er enginn sjálfgefinn samanburður, svo við þurfum alltaf að skrifa okkar eigið samanburðarfall.
- ▶ Til röðunar notum við fallið `qsort(...)`.
- ▶ Fallið tekur fjögur viðföng:
 - ▶ `void* a`. Þetta er fylkið sem við viljum raða.
 - ▶ `size_t n`. Þetta er fjöldi staka í fylkinu sem `a` svarar til.
 - ▶ `size_t s`. Þetta er stærð hvers staks í fylkinu okkar (í bætum).
 - ▶ `int (*cmp)(const void *, const void*)`. Þetta er samanburðarfallið okkar.
- ▶ Síðasta inntakið er kannski flókið við fyrstu sýn en er einfalt fyrir okkur að nota.
- ▶ Þetta er *fallabendir* (e. *function pointer*) ef þið viljið kynna ykkur það frekar.

Röðun í C

```
#include <stdio.h>
#include <stdlib.h>

int cmp(const void* p1, const void* p2)
{
    return *(int*)p1 - *(int*)p2;
}

int rcmp(const void* p1, const void* p2)
{
    return *(int*)p2 - *(int*)p1;
}

int main()
{
    int n, i;
    scanf("%d", &n);
    int a[n];
    for (i = 0; i < n; i++) scanf("%d", &a[i]);

    qsort(a, n, sizeof(a[0]), cmp);

    for (i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");

    qsort(a, n, sizeof(a[0]), rcmp);

    for (i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");

    return 0;
}
```


Uppsetning dæma

- ▶ Dæmin sem við sjáum á Kattis eru (oftast) af stöðluðu sniði.

Uppsetning dæma

- ▶ Dæmin sem við sjáum á Kattis eru (oftast) af stöðluðu sniði.
 - ▶ Saga.

Uppsetning dæma

- ▶ Dæmin sem við sjáum á Kattis eru (oftast) af stöðluðu sniði.
 - ▶ Saga.
 - ▶ Dæmið.

Uppsetning dæma

- ▶ Dæmin sem við sjáum á Kattis eru (oftast) af stöðluðu sniði.
 - ▶ Saga.
 - ▶ Dæmið.
 - ▶ Inntaks -og úttakslýsingar.

Uppsetning dæma

- ▶ Dæmin sem við sjáum á Kattis eru (oftast) af stöðluðu sniði.
 - ▶ Saga.
 - ▶ Dæmið.
 - ▶ Inntaks -og úttakslýsingar.
 - ▶ Sýnidæmi.

Uppsetning dæma

- ▶ Dæmin sem við sjáum á Kattis eru (oftast) af stöðluðu sniði.
 - ▶ Saga.
 - ▶ Dæmið.
 - ▶ Inntaks -og úttakslýsingar.
 - ▶ Sýnidæmi.
- ▶ Fyrstu tveir punktarnir geta verið blandaðir saman.

Uppsetning dæma

- ▶ Dæmin sem við sjáum á Kattis eru (oftast) af stöðluðu sniði.
 - ▶ Saga.
 - ▶ Dæmið.
 - ▶ Inntaks -og úttakslýsingar.
 - ▶ Sýnidæmi.
- ▶ Fyrstu tveir punktarnir geta verið blandaðir saman.
- ▶ Þeir eru líka lengsti hluti dæmisins.

A Different Problem

Write a program that computes the difference between non-negative integers.

Input

Each line of the input consists of a pair of integers. Each integer is between 0 and 10^{15} (inclusive). The input is terminated by end of file.

Output

For each pair of integers in the input, output one line, containing the absolute value of their difference.

Sample Input 1

```
10 12
71293781758123 72784
1 12345677654321
```



Sample Output 1

```
2
71293781685339
12345677654320
```



Röng lausn. Hver er villan?

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

int main()
{
    int a, b;
    while (cin >> a >> b)
    {
        cout << abs(a - b) << endl;
    }
}
```

Rétt lausn

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

int main()
{
    long long a, b;
    while (cin >> a >> b)
    {
        cout << abs(a - b) << endl;
    }
}
```

long long

- ▶ Þurfum við þó alltaf að skrifa long long?

long long

- ▶ Þurfum við þó alltaf að skrifa long long?
- ▶ Nei!

long long

- ▶ Þurfum við þó alltaf að skrifa long long?
- ▶ Nei!
- ▶ Við getum notað typedef.

long long

- ▶ Þurfum við þó alltaf að skrifa long long?
- ▶ Nei!
- ▶ Við getum notað typedef.
- ▶ Við notum einfaldlega typedef <gamla> <nýja>;.

long long

- ▶ Þurfum við þó alltaf að skrifa `long long`?
- ▶ Nei!
- ▶ Við getum notað `typedef`.
- ▶ Við notum einfaldlega `typedef <gamla> <nýja>;`.
- ▶ Venjan í keppnisforritun er að nota `typedef long long ll;`.

long long

- ▶ Þurfum við þó alltaf að skrifa `long long`?
- ▶ Nei!
- ▶ Við getum notað `typedef`.
- ▶ Við notum einfaldlega `typedef <gamla> <nýja>;`.
- ▶ Venjan í keppnisforritun er að nota `typedef long long ll;`.
- ▶ Við munum nota `typedef` aftur.

Rétt lausn með typedef

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

int main()
{
    ll a, b;
    while (cin >> a >> b)
    {
        cout << abs(a - b) << endl;
    }
}
```

Time Limit Exceede

- ▶ Hvernig vitum að lausnin okkar sé of hæg?

Time Limit Exceede

- ▶ Hvernig vitum að lausnin okkar sé of hæg?
- ▶ Ein leið er að útfæra lausnina, senda hana inn og gá hvað Kattis segir.

Time Limit Exceede

- ▶ Hvernig vitum að lausnin okkar sé of hæg?
- ▶ Ein leið er að útfæra lausnina, senda hana inn og gá hvað Kattis segir.
- ▶ Það myndi þó spara mikla vinnu ef við gætum svarað spurningunni án þess að útfæra.

Time Limit Exceede

- ▶ Hvernig vitum að lausnin okkar sé of hæg?
- ▶ Ein leið er að útfæra lausnina, senda hana inn og gá hvað Kattis segir.
- ▶ Það myndi þó spara mikla vinnu ef við gætum svarað spurningunni án þess að útfæra.
- ▶ Einnig gæti leynst önnur villa í útfærslunni okkar sem gefur okkur Time Limit Exceeded (TLE).

Time Limit Exceede

- ▶ Hvernig vitum að lausnin okkar sé of hæg?
- ▶ Ein leið er að útfæra lausnina, senda hana inn og gá hvað Kattis segir.
- ▶ Það myndi þó spara mikla vinnu ef við gætum svarað spurningunni án þess að útfæra.
- ▶ Einnig gæti leynst önnur villa í útfærslunni okkar sem gefur okkur Time Limit Exceeded (TLE).
- ▶ Til að ákvarða hvort lausn sé nógu hröð þá notum við *tímaflækjur*.

Time Limit Exceede

- ▶ Hvernig vitum að lausnin okkar sé of hæg?
- ▶ Ein leið er að útfæra lausnina, senda hana inn og gá hvað Kattis segir.
- ▶ Það myndi þó spara mikla vinnu ef við gætum svarað spurningunni án þess að útfæra.
- ▶ Einnig gæti leynst önnur villa í útfærslunni okkar sem gefur okkur Time Limit Exceeded (TLE).
- ▶ Til að ákvarða hvort lausn sé nógu hröð þá notum við *tímaflækjur*.
- ▶ Sum ykkar þekka tímaflækjur en önnur ekki.

Time Limit Exceede

- ▶ Hvernig vitum að lausnin okkar sé of hæg?
- ▶ Ein leið er að útfæra lausnina, senda hana inn og gá hvað Kattis segir.
- ▶ Það myndi þó spara mikla vinnu ef við gætum svarað spurningunni án þess að útfæra.
- ▶ Einnig gæti leynst önnur villa í útfærslunni okkar sem gefur okkur Time Limit Exceeded (TLE).
- ▶ Til að ákvarða hvort lausn sé nógu hröð þá notum við *tímaflækjur*.
- ▶ Sum ykkar þekka tímaflækjur en önnur ekki.
- ▶ Skoðum fyrst hvað tímaflækjur eru í grófum dráttum.

Tímaflækjur í grófum dráttum

- ▶ Keyrslutími forrits er háður stærðinni á inntakinu.

Tímaflækjur í grófum dráttum

- ▶ Keyrslutími forrits er háður stærðinni á inntakinu.
- ▶ Tímaflækjan lýsir hvernig keyrslutími forritsins skalast með inntakinu (í versta falli).

Tímaflækjur í grófum dráttum

- ▶ Keyrslutími forrits er háður stærðinni á inntakinu.
- ▶ Tímaflækjan lýsir hvernig keyrslutími forritsins skalast með inntakinu (í versta falli).
- ▶ Ef forritið er með tímaflækju $\mathcal{O}(f(n))$ þýðir það að keyrslutíminn vex eins of f þegar n vex.

Tímaflækjur í grófum dráttum

- ▶ Keyrslutími forrits er háður stærðinni á inntakinu.
- ▶ Tímaflækjan lýsir hvernig keyrslutími forritsins skalast með inntakinu (í versta falli).
- ▶ Ef forritið er með tímaflækju $\mathcal{O}(f(n))$ þýðir það að keyrslutíminn vex eins of f þegar n vex.
- ▶ Til dæmis ef forritið hefur tímaflækju $\mathcal{O}(n)$ þá tvöfaldast keyrslutími þegar inntakið tvöfaldast.

Tímaflækjur í grófum dráttum

- ▶ Keyrslutími forrits er háður stærðinni á inntakinu.
- ▶ Tímaflækjan lýsir hvernig keyrslutími forritsins skalast með inntakinu (í versta falli).
- ▶ Ef forritið er með tímaflækju $\mathcal{O}(f(n))$ þýðir það að keyrslutíminn vex eins of f þegar n vex.
- ▶ Til dæmis ef forritið hefur tímaflækju $\mathcal{O}(n)$ þá tvöfaldast keyrslutími þegar inntakið tvöfaldast.
- ▶ Hér gerum við ráð fyrir að grunnaðgerðirnar okkar taki fastann tíma, eða séu með tímaflækju $\mathcal{O}(1)$.

- ▶ Ef forritið okkar þarf að framkvæma $\mathcal{O}(f(n))$ aðgerð m sinnum þá er tímaflækjan $\mathcal{O}(m \cdot f(n))$.

- ▶ Ef forritið okkar þarf að framkvæma $\mathcal{O}(f(n))$ aðgerð m sinnum þá er tímaflækjan $\mathcal{O}(m \cdot f(n))$.
- ▶ Þetta er reglan sem við notum oftast í keppnisforritun.

- ▶ Ef forritið okkar þarf að framkvæma $\mathcal{O}(f(n))$ aðgerð m sinnum þá er tímaflækjan $\mathcal{O}(m \cdot f(n))$.
- ▶ Þetta er reglan sem við notum oftast í keppnisforritun.
- ▶ Hún segir okkur til dæmis að tvöföld for-lykkja, þar sem hver for-lykkja er n löng, er $\mathcal{O}(n^2)$.

- ▶ Ef forritið okkar þarf að framkvæma $\mathcal{O}(f(n))$ aðgerð m sinnum þá er tímaflækjan $\mathcal{O}(m \cdot f(n))$.
- ▶ Þetta er reglan sem við notum oftast í keppnisforritun.
- ▶ Hún segir okkur til dæmis að tvöföld for-lykkja, þar sem hver for-lykkja er n löng, er $\mathcal{O}(n^2)$.
- ▶ Ef við erum með tvær einfaldar for-lykkjur, báðar af lengd n , þá er forritið $\mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n)$

- ▶ Ef forritið okkar þarf að framkvæma $\mathcal{O}(f(n))$ aðgerð m sinnum þá er tímaflækjan $\mathcal{O}(m \cdot f(n))$.
- ▶ Þetta er reglan sem við notum oftast í keppnisforritun.
- ▶ Hún segir okkur til dæmis að tvöföld for-lykkja, þar sem hver for-lykkja er n löng, er $\mathcal{O}(n^2)$.
- ▶ Ef við erum með tvær einfaldar for-lykkjur, báðar af lengd n , þá er forritið $\mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n)$
- ▶ Einnig gildir að tímaflækja forritsins okkar takmarkast af hægasta hluta forritsins.

- ▶ Ef forritið okkar þarf að framkvæma $\mathcal{O}(f(n))$ aðgerð m sinnum þá er tímaflækjan $\mathcal{O}(m \cdot f(n))$.
- ▶ Þetta er reglan sem við notum oftast í keppnisforritun.
- ▶ Hún segir okkur til dæmis að tvöföld for-lykkja, þar sem hver for-lykkja er n löng, er $\mathcal{O}(n^2)$.
- ▶ Ef við erum með tvær einfaldar for-lykkjur, báðar af lengd n , þá er forritið $\mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n)$
- ▶ Einnig gildir að tímaflækja forritsins okkar takmarkast af hægasta hluta forritsins.
- ▶ Til dæmis er $\mathcal{O}(n + n + n + n + n^2) = \mathcal{O}(n^2)$.

- ▶ Við segjum að fall $g(x)$ sé í menginu $\mathcal{O}(f(x))$ ef til eru rauntölur c og x_0 þannig að

$$|g(x)| \leq c \cdot f(x)$$

fyrir öll $x > x_0$.

Stærðfræði

- ▶ Við segjum að fall $g(x)$ sé í menginu $\mathcal{O}(f(x))$ ef til eru rauntölur c og x_0 þannig að

$$|g(x)| \leq c \cdot f(x)$$

fyrir öll $x > x_0$.

- ▶ Þetta þýðir í raun að fallið $|g(x)|$ verður á endanum minna en $k \cdot f(x)$.

- ▶ Við segjum að fall $g(x)$ sé í menginu $\mathcal{O}(f(x))$ ef til eru rauntölur c og x_0 þannig að

$$|g(x)| \leq c \cdot f(x)$$

fyrir öll $x > x_0$.

- ▶ Þetta þýðir í raun að fallið $|g(x)|$ verður á endanum minna en $k \cdot f(x)$.
- ▶ Þessi lýsing undirstrikar betur að $f(x)$ er efra mat á $g(x)$, og er því að segja að $g(x)$ hagi sér ekki verr en $f(x)$.

Þekktar tímaflækjur

- ▶ Tímaflækjur algrengra aðgerða eru:

Pekktar tímaflækjur

- Tímaflækjur algrengra aðgerða eru:

Aðgerð	Lýsing	Tímaflækja
Línulega leit	Almenn leit í fylki	$\mathcal{O}(n)$
Helsingunarleit	Leit í röðuðu fylki	$\mathcal{O}(\log n)$
Röðun á heiltölum	Röðun á heiltalna fylki	$\mathcal{O}(n \log n)$
Strengjasamanburður	Bera saman tvo strengi af lengd n	$\mathcal{O}(n)$
Almenn röðun	Röðun með $\mathcal{O}(T(m))$ samanburður	$\mathcal{O}(T(m) \cdot n \log n)$

10^8 reglan

- ▶ Þegar við ræðum tímaflækjur er “tími” er ekki endilega rétt orðið.

10^8 reglan

- ▶ Þegar við ræðum tímaflækjur er “tími” er ekki endilega rétt orðið.
- ▶ Við erum frekar að lýsa fjölda aðgerða sem forritið framkvæmir.

10^8 reglan

- ▶ Þegar við ræðum tímaflækjur er “tími” er ekki endilega rétt orðið.
- ▶ Við erum frekar að lýsa fjölda aðgerða sem forritið framkvæmir.
- ▶ Í keppnisforritun notum við 10^8 *regluna*:

10^8 reglan

- ▶ Þegar við ræðum tímaflækjur er “tími” er ekki endilega rétt orðið.
- ▶ Við erum frekar að lýsa fjölda aðgerða sem forritið framkvæmir.
- ▶ Í keppnisforritun notum við 10^8 *regluna*:
 - ▶ Tökum verstu tilfellið sem koma fyrir í inntakslýsingunni á dæminu, stingum því inn í tímaflækjuna okkar og deilum með 10^8 .

10^8 reglan

- ▶ Þegar við ræðum tímaflækjur er “tími” er ekki endilega rétt orðið.
- ▶ Við erum frekar að lýsa fjölda aðgerða sem forritið framkvæmir.
- ▶ Í keppnisforritun notum við 10^8 *regluna*:
 - ▶ Tökum verstu tilfellið sem koma fyrir í inntakslýsingunni á dæminu, stingum því inn í tímaflækjuna okkar og deilum með 10^8 .
 - ▶ Ef útkoman er minni en fjöldi sekúnda í tímamörkum dæmisins þá er lausnin okkar nógu hröð, annars er hún of hæg.

10^8 reglan

- ▶ Þegar við ræðum tímaflækjur er “tími” er ekki endilega rétt orðið.
- ▶ Við erum frekar að lýsa fjölda aðgerða sem forritið framkvæmir.
- ▶ Í keppnisforritun notum við 10^8 *regluna*:
 - ▶ Tökum verstu tilfellið sem koma fyrir í inntakslýsingunni á dæminu, stingum því inn í tímaflækjuna okkar og deilum með 10^8 .
 - ▶ Ef útkoman er minni en fjöldi sekúnda í tímamörkum dæmisins þá er lausnin okkar nógu hröð, annars er hún of hæg.
- ▶ Þessa reglu mætti um orða sem: “Við gerum ráð fyrir að forritið geti framkvæmt 10^8 aðgerðir á sekúndu”.

10^8 reglan

- ▶ Þegar við ræðum tímaflækjur er “tími” er ekki endilega rétt orðið.
- ▶ Við erum frekar að lýsa fjölda aðgerða sem forritið framkvæmir.
- ▶ Í keppnisforritun notum við 10^8 *regluna*:
 - ▶ Tökum verstu tilfellin sem koma fyrir í inntakslýsingunni á dæminu, stingum því inn í tímaflækjuna okkar og deilum með 10^8 .
 - ▶ Ef útkoman er minni en fjöldi sekúnda í tímamörkum dæmisins þá er lausnin okkar nógu hröð, annars er hún of hæg.
- ▶ Þessa reglu mætti um orða sem: “Við gerum ráð fyrir að forritið geti framkvæmt 10^8 aðgerðir á sekúndu”.
- ▶ Þessi regla er gróf nálgun, en virkar mjög vel því þetta er það sem dæmahöfundar hafa í huga þegar þeir semja dæmi.

10^8 reglan

- ▶ Þegar við ræðum tímaflækjur er “tími” er ekki endilega rétt orðið.
- ▶ Við erum frekar að lýsa fjölda aðgerða sem forritið framkvæmir.
- ▶ Í keppnisforritun notum við 10^8 *regluna*:
 - ▶ Tökum verstu tilfellið sem koma fyrir í inntakslýsingunni á dæminu, stingum því inn í tímaflækjuna okkar og deilum með 10^8 .
 - ▶ Ef útkoman er minni en fjöldi sekúnda í tímamörkum dæmisins þá er lausnin okkar nógu hröð, annars er hún of hæg.
- ▶ Þessa reglu mætti um orða sem: “Við gerum ráð fyrir að forritið geti framkvæmt 10^8 aðgerðir á sekúndu”.
- ▶ Þessi regla er gróf nálgun, en virkar mjög vel því þetta er það sem dæmahöfundar hafa í huga þegar þeir semja dæmi.
- ▶ Með þetta í huga fáum við eftirfarandi töflu.

Stærð n	Versta tímaflækja	Dæmi
≤ 10	$\mathcal{O}(n!)$	TSP með tæmandi leit
≤ 15	$\mathcal{O}(n^2 2^n)$	TSP með kvikri bestun
≤ 20	$\mathcal{O}(n 2^n)$	Kvik bestun yfir hlutmengi
≤ 100	$\mathcal{O}(n^4)$	Almenn spyrðing
≤ 400	$\mathcal{O}(n^3)$	Floyd-Warshall
$\leq 10^4$	$\mathcal{O}(n^2)$	Lengsti sameiginlegi hlutstrengur
$\leq 10^5$	$\mathcal{O}(n\sqrt{n})$	Reiknirit sem byggja á rótarpáttun
$\leq 10^6$	$\mathcal{O}(n \log n)$	Of mikið til að þora að taka dæmi
$\leq 10^7$	$\mathcal{O}(n)$	Næsta tala sem er stærri (NGE)
$\leq 2^{10^7}$	$\mathcal{O}(\log n)$	Helmingunarleit
$> 2^{10^7}$	$\mathcal{O}(1)$	Ad hoc

TLE trikk

- ▶ Stundum fær maður TLE þótt maður sé viss um að lausnin sé nógu hröð.

TLE trikk

- ▶ Stundum fær maður TLE þótt maður sé viss um að lausnin sé nógu hröð.
- ▶ Ef forritið þarf að lesa eða skrifa mikið gæti það verið að hægja nóg á forritun til að gefa TLE.

TLE trikk

- ▶ Stundum fær maður TLE þótt maður sé viss um að lausnin sé nógu hröð.
- ▶ Ef forritið þarf að lesa eða skrifa mikið gæti það verið að hægja nóg á forritun til að gefa TLE.
- ▶ Þetta stafar af því að til að lesa eða skrifa þarf forritið að tala við stýrikerfið.

TLE trikk

- ▶ Stundum fær maður TLE þótt maður sé viss um að lausnin sé nógu hröð.
- ▶ Ef forritið þarf að lesa eða skrifa mikið gæti það verið að hægja nóg á forritun til að gefa TLE.
- ▶ Þetta stafar af því að til að lesa eða skrifa þarf forritið að tala við stýrikerfið.
- ▶ Til að leysa þetta skrifa sum forrit í *biðminni* (e. *buffer*) og prenta bara þegar það fyllist.

TLE trikk

- ▶ Stundum fær maður TLE þótt maður sé viss um að lausnin sé nógu hröð.
- ▶ Ef forritið þarf að lesa eða skrifa mikið gæti það verið að hægja nóg á forritun til að gefa TLE.
- ▶ Þetta stafar af því að til að lesa eða skrifa þarf forritið að tala við stýrikerfið.
- ▶ Til að leysa þetta skrifa sum forrit í *biðminni* (e. *buffer*) og prenta bara þegar það fyllist.
- ▶ Svona er þetta gert í C.

TLE trikk

- ▶ Í C++ er biðminnið tæmt þegar `std::endl` er prentað.

TLE trikk

- ▶ Í C++ er biðminnið tæmt þegar `std::endl` er prentað.
- ▶ Til að koma í veg fyrir þetta er hægt að prenta `\n` í staðinn.

TLE trikk

- ▶ Í C++ er biðminnið tæmt þegar `std::endl` er prentað.
- ▶ Til að koma í veg fyrir þetta er hægt að prenta `\n` í staðinn.
- ▶ Til dæmis `cout << x << '\n'`.

TLE trikk

- ▶ Í C++ er biðminnið tæmt þegar `std::endl` er prentað.
- ▶ Til að koma í veg fyrir þetta er hægt að prenta `\n` í staðinn.
- ▶ Til dæmis `cout << x << '\n'`.
- ▶ Það borgar sig einnig að setja `ios::sync_with_stdio(false)` fremst í `main()`.

TLE trikk

- ▶ Í C++ er biðminnið tæmt þegar `std::endl` er prentað.
- ▶ Til að koma í veg fyrir þetta er hægt að prenta `\n` í staðinn.
- ▶ Til dæmis `cout << x << '\n'`.
- ▶ Það borgar sig einnig að setja `ios::sync_with_stdio(false)` fremst í `main()`.
- ▶ Ef þið eruð í Java mæli ég með Kattio.

TLE trikk

- ▶ Í C++ er biðminnið tæmt þegar `std::endl` er prentað.
- ▶ Til að koma í veg fyrir þetta er hægt að prenta `\n` í staðinn.
- ▶ Til dæmis `cout << x << '\n'`.
- ▶ Það borgar sig einnig að setja `ios::sync_with_stdio(false)` fremst í `main()`.
- ▶ Ef þið eruð í Java mæli ég með Kattio.
- ▶ Það má finna á GitHub.

Innbyggðar gagnagrindur í C++

- ▶ Grunnur C++ býr yfir mörgum sterkum gagnagrindum.

Innbyggðar gagnagrindur í C++

- ▶ Grunnur C++ býr yfir mörgum sterkum gagnagrindum.
- ▶ Skoðum helstu slíku gagnagrindur og tímaflækjur mikilvægust aðgerða þeirra.

Innbyggðar gagnagrindur í C++

- ▶ Grunnur C++ býr yfir mörgum sterkum gagnagrindum.
- ▶ Skoðum helstu slíku gagnagrindur og tímaflækjur mikilvægust aðgerða þeirra.
- ▶ Við munum bara fjalla um gagnagrindurnar í grófum dráttum.

Innbyggðar gagnagrindur í C++

- ▶ Grunnur C++ býr yfir mörgum sterkum gagnagrindum.
- ▶ Skoðum helstu slíku gagnagrindur og tímaflækjur mikilvægust aðgerða þeirra.
- ▶ Við munum bara fjalla um gagnagrindurnar í grófum dráttum.
- ▶ Það er hægt að finna ítarlegra efni og dæmi um notkun á netinu.

Fylki

- ▶ Lýkt og í mörgum öðrum forritunarmálum eru fylki í C++.

Fylki

- ▶ Lýkt og í mörgum öðrum forritunarmálum eru fylki í C++.
- ▶ Fylki geyma gögn og eru af fastri stærð.

Fylki

- ▶ Lýkt og í mörgum öðrum forritunarmálum eru fylki í C++.
- ▶ Fylki geyma gögn og eru af fastri stærð.
- ▶ Þar sem þau eru af fastri stærð má gefa þeim tileinkað, aðliggjandi svæði í minni.

Fylki

- ▶ Lýkt og í mörgum öðrum forritunarmálum eru fylki í C++.
- ▶ Fylki geyma gögn og eru af fastri stærð.
- ▶ Þar sem þau eru af fastri stærð má gefa þeim tileinkað, aðliggjandi svæði í minni.
- ▶ Þetta leyfir manni að vísa í fylkið í $\mathcal{O}(1)$.

Fylki

- ▶ Lýkt og í mörgum öðrum forritunarmálum eru fylki í C++.
- ▶ Fylki geyma gögn og eru af fastri stærð.
- ▶ Þar sem þau eru af fastri stærð má gefa þeim tileinkað, aðliggjandi svæði í minni.
- ▶ Þetta leyfir manni að vísa í fylkið í $\mathcal{O}(1)$.

Aðgerð	Tímaflækja
Lesi eða skrifa ótiltekið stak	$\mathcal{O}(1)$
Bæta staki aftast	$\mathcal{O}(n)$
Skeyta saman tveimur	$\mathcal{O}(n)$

vector

- ▶ Gagnagrindin vector er að mestu leiti eins og fylki.

vector

- ▶ Gagnagrindin vector er að mestu leiti eins og fylki.
- ▶ Það má þó bæta stökum aftan á vector í $\mathcal{O}(1)$.

vector

- ▶ Gagnagrindin `vector` er að mestu leiti eins og fylki.
- ▶ Það má þó bæta stökum aftan á `vector` í $\mathcal{O}(1)$.
- ▶ Margir nota bara `vector` og aldrei fylki sem slík.

vector

- ▶ Gagnagrindin vector er að mestu leiti eins og fylki.
- ▶ Það má þó bæta stökum aftan á vector í $\mathcal{O}(1)$.
- ▶ Margir nota bara vector og aldrei fylki sem slík.

Aðgerð	Tímaflækja
Lesi eða skrifa ótiltekið stak	$\mathcal{O}(1)$
Bæta staki aftast	$\mathcal{O}(1)$
Skeyta saman tveimur	$\mathcal{O}(n)$

list

- ▶ Gagnagrindin `list` geymir gögn líkt og fylki gera, en stökin eru ekki aðliggjandi í minni.

list

- ▶ Gagnagrindin `list` geymir gögn líkt og fylki gera, en stökin eru ekki aðliggjandi í minni.
- ▶ Því er uppfletting ekki hröð.

list

- ▶ Gagnagrindin `list` geymir gögn líkt og fylki gera, en stökin eru ekki aðliggjandi í minni.
- ▶ Því er uppfletting ekki hröð.
- ▶ Aftur á móti er hægt að gera smávægilegar breytingar á `list` sem er ekki hægt að gera á fylkjum.

list

- ▶ Gagnagrindin `list` geymir gögn líkt og fylki gera, en stökin eru ekki aðliggjandi í minni.
- ▶ Því er uppfletting ekki hröð.
- ▶ Aftur á móti er hægt að gera smávægilegar breytingar á `list` sem er ekki hægt að gera á fylkjum.

Aðgerð	Tímaflækja
Finna stak	$O(n)$
Bæta staki aftast	$O(1)$
Bæta staki fremst	$O(1)$
Bæta staki fyrir aftan tiltekið stak	$O(1)$
Bæta staki fyrir framan tiltekið stak	$O(1)$
Skeyta saman tveimur	$O(1)$

stack

- ▶ Gagnagrindin `stack` geymir gögn og leyfir aðgang að síðasta staki sem var bætt við.

stack

- Gagnagrindin stack geymir gögn og leyfir aðgang að síðasta staki sem var bætt við.

Aðgerð	Tímaflækja
Bæta við staki	$O(1)$
Lesa nýjasta stakið	$O(1)$
Fjarlægja nýjasta stakið	$O(1)$

queue

- ▶ Gagnagrindin queue geymir gögn og leyfir aðgang að fyrsta stakinu sem var bætt við.

queue

- Gagnagrindin queue geymir gögn og leyfir aðgang að fyrsta stakinu sem var bætt við.

Aðgerð	Tímaflækja
Bæta við staki	$\mathcal{O}(1)$
Lesi elsta stakið	$\mathcal{O}(1)$
Fjarlægja elsta stakið	$\mathcal{O}(1)$

set

- ▶ Gagnagrindin `set` geymir gögn án endurtekninga og leyfir hraða uppflettingu.

set

- Gagnagrindin set geymir gögn án endurtekninga og leyfir hraða uppflettingu.

Aðgerð	Tímaflækja
Bæta við staki	$\mathcal{O}(\log n)$
Fjarlægja stak	$\mathcal{O}(\log n)$
Gá hvort staki hafi verið bætt við	$\mathcal{O}(\log n)$

Lausnar aðferðir

- ▶ Lausnir okkar má flokka í fimm flokka:

Lausnar aðferðir

- ▶ Lausnir okkar má flokka í fimm flokka:
 - ▶ Ad hoc.

Lausnar aðferðir

- ▶ Lausnir okkar má flokka í fimm flokka:
 - ▶ Ad hoc.
 - ▶ Tæmandi leit.

Lausnar aðferðir

- ▶ Lausnir okkar má flokka í fimm flokka:
 - ▶ Ad hoc.
 - ▶ Tæmandi leit.
 - ▶ Gráðugar lausnir.

Lausnar aðferðir

- ▶ Lausnir okkar má flokka í fimm flokka:
 - ▶ Ad hoc.
 - ▶ Tæmandi leit.
 - ▶ Gráðugar lausnir.
 - ▶ Deila og drottna (D&C).

Lausnar aðferðir

- ▶ Lausnir okkar má flokka í fimm flokka:
 - ▶ Ad hoc.
 - ▶ Tæmandi leit.
 - ▶ Gráðugar lausnir.
 - ▶ Deila og drottna (D&C).
 - ▶ Kvik bestun (DP).

Lausnar aðferðir

- ▶ Lausnir okkar má flokka í fimm flokka:
 - ▶ Ad hoc.
 - ▶ Tæmandi leit.
 - ▶ Gráðugar lausnir.
 - ▶ Deila og drottna (D&C).
 - ▶ Kvik bestun (DP).
- ▶ Þessi skipting er ekki fullkomin, en það er þó gott að hafa hana í huga.

Lausnar aðferðir

- ▶ Lausnir okkar má flokka í fimm flokka:
 - ▶ Ad hoc.
 - ▶ Tæmandi leit.
 - ▶ Gráðugar lausnir.
 - ▶ Deila og drottna (D&C).
 - ▶ Kvik bestun (DP).
- ▶ Þessi skipting er ekki fullkomin, en það er þó gott að hafa hana í huga.
- ▶ Til dæmis má færa rök fyrir því að gráðugar lausnir og D&C séu sértilfelli af kvikri bestun.

Lausnar aðferðir

- ▶ Lausnir okkar má flokka í fimm flokka:
 - ▶ Ad hoc.
 - ▶ Tæmandi leit.
 - ▶ Gráðugar lausnir.
 - ▶ Deila og drottna (D&C).
 - ▶ Kvik bestun (DP).
- ▶ Þessi skipting er ekki fullkomin, en það er þó gott að hafa hana í huga.
- ▶ Til dæmis má færa rök fyrir því að gráðugar lausnir og D&C séu sértilfelli af kvikri bestun.
- ▶ Við munum byrja á því að fjalla almennt um þessar aðferðir og fara svo í sértækara efni.

Lausnar aðferðir

- ▶ Lausnir okkar má flokka í fimm flokka:
 - ▶ Ad hoc.
 - ▶ Tæmandi leit.
 - ▶ Gráðugar lausnir.
 - ▶ Deila og drottna (D&C).
 - ▶ Kvik bestun (DP).
- ▶ Þessi skipting er ekki fullkomin, en það er þó gott að hafa hana í huga.
- ▶ Til dæmis má færa rök fyrir því að gráðugar lausnir og D&C séu sértilfelli af kvikri bestun.
- ▶ Við munum byrja á því að fjalla almennt um þessar aðferðir og fara svo í sértækara efni.
- ▶ Þá er oft gott að hafa í huga hvernig flokka megi reikniritin.

Ad hoc

- ▶ Ef lausn dæmisins byggir ekki á sérþekkingu flokkast dæmið sem *Ad hoc*.

Ad hoc

- ▶ Ef lausn dæmisins byggir ekki á sérþekkingu flokkast dæmið sem *Ad hoc*.
- ▶ Þessi dæmi eru stundum flokkuð undir “implementation”, eða sem *útfærsludæmi*.

Ad hoc

- ▶ Ef lausn dæmisins byggir ekki á sérþekkingu flokkast dæmið sem *Ad hoc*.
- ▶ Þessi dæmi eru stundum flokkuð undir “implementation”, eða sem *útfærsludæmi*.
- ▶ Þetta er gert því flest Ad hoc dæmi snúast um að fylgja beint leiðbeiningum.

Ad hoc

- ▶ Ef lausn dæmisins byggir ekki á sérþekkingu flokkast dæmið sem *Ad hoc*.
- ▶ Þessi dæmi eru stundum flokkuð undir “implementation”, eða sem *útfærsludæmi*.
- ▶ Þetta er gert því flest Ad hoc dæmi snúast um að fylgja beint leiðbeiningum.
- ▶ Það eru þó undantekningar.

Ad hoc

- ▶ Ef lausn dæmisins byggir ekki á sérþekkingu flokkast dæmið sem *Ad hoc*.
- ▶ Þessi dæmi eru stundum flokkuð undir “implementation”, eða sem *útfærsludæmi*.
- ▶ Þetta er gert því flest Ad hoc dæmi snúast um að fylgja beint leiðbeiningum.
- ▶ Það eru þó undantekningar.
- ▶ Í NCPC 2020 var Ad hoc dæmi sem mætti ekki flokkast sem útfærsludæmi.

Ad hoc

- ▶ Ef lausn dæmisins byggir ekki á sérþekkingu flokkast dæmið sem *Ad hoc*.
- ▶ Þessi dæmi eru stundum flokkuð undir “implementation”, eða sem *útfærsludæmi*.
- ▶ Þetta er gert því flest Ad hoc dæmi snúast um að fylgja beint leiðbeiningum.
- ▶ Það eru þó undantekningar.
- ▶ Í NCPC 2020 var Ad hoc dæmi sem mætti ekki flokkast sem útfærsludæmi.
- ▶ Ad hoc dæmi flokkast oft til léttari dæma í keppnum.

Ad hoc

- ▶ Ef lausn dæmisins byggir ekki á sérþekkingu flokkast dæmið sem *Ad hoc*.
- ▶ Þessi dæmi eru stundum flokkuð undir “implementation”, eða sem *útfærsludæmi*.
- ▶ Þetta er gert því flest Ad hoc dæmi snúast um að fylgja beint leiðbeiningum.
- ▶ Það eru þó undantekningar.
- ▶ Í NCPC 2020 var Ad hoc dæmi sem mætti ekki flokkast sem útfærsludæmi.
- ▶ Ad hoc dæmi flokkast oft til léttari dæma í keppnum.
- ▶ Áður nefnt NCPC dæmi er þó aftur undanteking, því engin keppandi náði að leysa það dæmi.

Ad hoc

- ▶ Ef lausn dæmisins byggir ekki á sérþekkingu flokkast dæmið sem *Ad hoc*.
- ▶ Þessi dæmi eru stundum flokkuð undir “implementation”, eða sem *útfærsludæmi*.
- ▶ Þetta er gert því flest Ad hoc dæmi snúast um að fylgja beint leiðbeiningum.
- ▶ Það eru þó undantekningar.
- ▶ Í NCPC 2020 var Ad hoc dæmi sem mætti ekki flokkast sem útfærsludæmi.
- ▶ Ad hoc dæmi flokkast oft til léttari dæma í keppnum.
- ▶ Áður nefnt NCPC dæmi er þó aftur undanteking, því engin keppandi náði að leysa það dæmi.
- ▶ Samkvæmt skilgreiningu getum við ekki rætt Ad hoc dæmi ítarlega. Tökum því nokkur dæmi.

Blandað brot

- ▶ Þú átt að breyta almennu broti í blandað brot.

Blandað brot

- ▶ Þú átt að breyta almennu broti í blandað brot.
- ▶ Munið að almenna brotið p/q , og blandaða brotið $a + b/c$ tákna sömu töluna ef $p/q = a + b/c$.

Blandað brot

- ▶ Þú átt að breyta almennu broti í blandað brot.
- ▶ Munið að almenna brotið p/q , og blandaða brotið $a\ b/c$ tákna sömu töluna ef $p/q = a + b/c$.
- ▶ Munið einnig að ef $a\ b/c$ er almennt brot þá gildir $b < c$.

Blandað brot

- ▶ Þú átt að breyta almennu broti í blandað brot.
- ▶ Munið að almenna brotið p/q , og blandaða brotið $a\ b/c$ tákna sömu töluna ef $p/q = a + b/c$.
- ▶ Munið einnig að ef $a\ b/c$ er almennt brot þá gildir $b < c$.
- ▶ Blandaða brotið ykkar á að hafa sama nefnara og upprunarlega brotið.

Blandað brot

- ▶ Þú átt að breyta almennu broti í blandað brot.
- ▶ Munið að almenna brotið p/q , og blandaða brotið $a\ b/c$ tákna sömu töluna ef $p/q = a + b/c$.
- ▶ Munið einnig að ef $a\ b/c$ er almennt brot þá gildir $b < c$.
- ▶ Blandaða brotið ykkar á að hafa sama nefnara og upprunarlega brotið.
- ▶ Inntakið inniheldur tvær heiltölur $1 \leq p, q \leq 10^9$.

Blandað brot

- ▶ Þú átt að breyta almennu broti í blandað brot.
- ▶ Munið að almenna brotið p/q , og blandaða brotið $a\ b/c$ tákna sömu töluna ef $p/q = a + b/c$.
- ▶ Munið einnig að ef $a\ b/c$ er almennt brot þá gildir $b < c$.
- ▶ Blandaða brotið ykkar á að hafa sama nefnara og upprunarlega brotið.
- ▶ Inntakið inniheldur tvær heiltölur $1 \leq p, q \leq 10^9$.
- ▶ Úttakið skal innihalda blandaða brotið sem svarar til p/q .

Blandað brot

- ▶ Þú átt að breyta almennu broti í blandað brot.
- ▶ Munið að almenna brotið p/q , og blandaða brotið $a\ b/c$ tákna sömu töluna ef $p/q = a + b/c$.
- ▶ Munið einnig að ef $a\ b/c$ er almennt brot þá gildir $b < c$.
- ▶ Blandaða brotið ykkar á að hafa sama nefnara og upprunarlega brotið.
- ▶ Inntakið inniheldur tvær heiltölur $1 \leq p, q \leq 10^9$.
- ▶ Úttakið skal innihalda blandaða brotið sem svarar til p/q .

	Inntak	Úttak
Sýnidæmi 1	27 12	2 3 / 12
Sýnidæmi 2	2460000 98400	25 0 / 98400
Sýnidæmi 3	3 4000	0 3 / 4000

Lausn á blandað brot

- ▶ Hér nægir okkur að reikna.

Lausn á blandað brot

- ▶ Hér nægir okkur að reikna.
- ▶ Við getum aðeins stytt okkur leið með því að nota heiltöludeilingu.

Lausn á blandað brot

- ▶ Hér nægir okkur að reikna.
- ▶ Við getum aðeins stytt okkur leið með því að nota heiltöludeilingu.
- ▶ Við fáum þá að a er heiltalan sem fæst með deilingunni p/q og b er afgangurinn.

```
#include <stdio.h>
```

```
int main()  
{  
    int p, q, a, b, c;  
    scanf("%d%d", &p, &q);  
    a = p/q;  
    b = p%q;  
    c = q;  
    printf("%d %d / %d\n", a, b, c);  
    return 0;  
}
```

```
/* */
```

```
#include <stdio.h>
```

```
int main()  
{  
    int p, q;  
    scanf("%d%d", &p, &q);  
    printf("%d %d / %d\n", p/q, p%q, q);  
    return 0;  
}
```


Barnahjal

- ▶ Pið eruð að reyna að kenna barni að telja.

Barnahjal

- ▶ Þið eruð að reyna að kenna barni að telja.
- ▶ Það er þó ekki alltaf hægt að heyra hvað barnið segir.

Barnahjal

- ▶ Þið eruð að reyna að kenna barni að telja.
- ▶ Það er þó ekki alltaf hægt að heyra hvað barnið segir.
- ▶ Þið viljið ákvarða hvort það sem barnið er að segja gæti mögulega verið rétt.

Barnahjal

- ▶ Þið eruð að reyna að kenna barni að telja.
- ▶ Það er þó ekki alltaf hægt að heyra hvað barnið segir.
- ▶ Þið viljið ákvarða hvort það sem barnið er að segja gæti mögulega verið rétt.
- ▶ Fyrsta lína inntaksins inniheldur heiltölu $1 \leq n \leq 10^3$.

Barnahjal

- ▶ Þið eruð að reyna að kenna barni að telja.
- ▶ Það er þó ekki alltaf hægt að heyra hvað barnið segir.
- ▶ Þið viljið ákvarða hvort það sem barnið er að segja gæti mögulega verið rétt.
- ▶ Fyrsta lína inntaksins inniheldur heiltölu $1 \leq n \leq 10^3$.
- ▶ Síðan fylgir ein lína með n strengjum.

Barnahjal

- ▶ Þið eruð að reyna að kenna barni að telja.
- ▶ Það er þó ekki alltaf hægt að heyra hvað barnið segir.
- ▶ Þið viljið ákvarða hvort það sem barnið er að segja gæti mögulega verið rétt.
- ▶ Fyrsta lína inntaksins inniheldur heiltölu $1 \leq n \leq 10^3$.
- ▶ Síðan fylgir ein lína með n strengjum.
- ▶ Hver strengur er annaðhvort heiltala á bilinu $[0, 10^4]$ eða strengurinn “mumble”.

Barnahjal

- ▶ Þið eruð að reyna að kenna barni að telja.
- ▶ Það er þó ekki alltaf hægt að heyra hvað barnið segir.
- ▶ Þið viljið ákvarða hvort það sem barnið er að segja gæti mögulega verið rétt.
- ▶ Fyrsta lína inntaksins inniheldur heiltölu $1 \leq n \leq 10^3$.
- ▶ Síðan fylgir ein lína með n strengjum.
- ▶ Hver strengur er annaðhvort heiltala á bilinu $[0, 10^4]$ eða strengurinn “mumble”.
- ▶ Ef það er hægt að skipta út öllum “mumble” fyrir tölu þannig að talningin sé rétt skal prenta “jebb”.

Barnahjal

- ▶ Þið eruð að reyna að kenna barni að telja.
- ▶ Það er þó ekki alltaf hægt að heyra hvað barnið segir.
- ▶ Þið viljið ákvarða hvort það sem barnið er að segja gæti mögulega verið rétt.
- ▶ Fyrsta lína inntaksins inniheldur heiltölu $1 \leq n \leq 10^3$.
- ▶ Síðan fylgir ein lína með n strengjum.
- ▶ Hver strengur er annaðhvort heiltala á bilinu $[0, 10^4]$ eða strengurinn “mumble”.
- ▶ Ef það er hægt að skipta út öllum “mumble” fyrir tölu þannig að talningin sé rétt skal prenta “jebb”.
- ▶ Annars skal prenta “neibb”.

Lausn á Barnahjal

- ▶ Ef i -ti strengurinn inniheldur strenginn sem svarar til tölunnar i eða “mumble”, fyrir öll i , þá er barnið kannski að telja rétt.

Lausn á Barnahjal

- ▶ Ef i -ti strengurinn inniheldur strenginn sem svarar til tölunnar i eða “mumble”, fyrir öll i , þá er barnið kannski að telja rétt.
- ▶ Annars er barnið að telja rangt.

```
n = int(input())
l = input().split()
f = True
for i in range(n):
    if l[i] != 'mumble' and l[i] != str(i + 1):
        f = False
        break
if f: print('jebb')
else: print('neibb')
```