Tímaflækjur

Bergur Snorrason

19. janúar 2022

Tímaflækjur

- Látum $f, g\colon [0,\infty) \mapsto \mathbb{R}$.

Tímaflækjur

- Látum $f, g\colon [0,\infty) \mapsto \mathbb{R}$.
- Við segjum að fall g sé í menginu $\mathcal{O}(f)$ ef til eru rauntölur c og x_0 þannig að
$$|g(x)| \le c \cdot f(x)$$
fyrir öll $x > x_0$.

Tímaflækjur

- Látum $f, g\colon [0, \infty) \mapsto \mathbb{R}$.
- Við segjum að fall g sé í menginu $\mathcal{O}(f)$ 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 $|g|$ verður á endanum minna en $c \cdot f$.

Tímaflækjur

- Látum $f, g\colon [0, \infty) \mapsto \mathbb{R}$.
- Við segjum að fall g sé í menginu $\mathcal{O}(f)$ 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 $|g|$ verður á endanum minna en $c \cdot f$.
- Þessi lýsing undirstrikar að f er efra mat á g, það er að segja g hagar sér ekki verr en f.

Tímaflækjur

- Látum $f, g\colon [0, \infty) \mapsto \mathbb{R}$.
- Við segjum að fall g sé í menginu $\mathcal{O}(f)$ 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 $|g|$ verður á endanum minna en $c \cdot f$.
- Þessi lýsing undirstrikar að f er efra mat á g, það er að segja g hagar sér ekki verr en f.
- Ef $g \in \mathcal{O}(f)$ og $f \in \mathcal{O}(g)$ þá segjum við að $f \in \Theta(g)$ (og $g \in \Theta(f)$).

Dæmi

- Tökum nokkur dæmi.

Dæmi

- Tökum nokkur dæmi.
- Ef $f \in \mathcal{O}(g)$ og $r > 0$ þá er $r \cdot f \in \mathcal{O}(g)$.

Dæmi

- Tökum nokkur dæmi.
- Ef $f \in \mathcal{O}(g)$ og $r > 0$ þá er $r \cdot f \in \mathcal{O}(g)$.
- Ef $f_1 \in \mathcal{O}(g_1)$ og $f_2 \in \mathcal{O}(g_2)$ þá er $f_1 \cdot f_2 \in \mathcal{O}(g_1 \cdot g_2)$.

Dæmi

- Tökum nokkur dæmi.
- Ef $f \in \mathcal{O}(g)$ og $r > 0$ þá er $r \cdot f \in \mathcal{O}(g)$.
- Ef $f_1 \in \mathcal{O}(g_1)$ og $f_2 \in \mathcal{O}(g_2)$ þá er $f_1 \cdot f_2 \in \mathcal{O}(g_1 \cdot g_2)$.
- Ef p er n-ta stigs margliða þá er $p \in \Theta(x^n)$.

Dæmi

- Tökum nokkur dæmi.
- Ef $f \in \mathcal{O}(g)$ og $r > 0$ þá er $r \cdot f \in \mathcal{O}(g)$.
- Ef $f_1 \in \mathcal{O}(g_1)$ og $f_2 \in \mathcal{O}(g_2)$ þá er $f_1 \cdot f_2 \in \mathcal{O}(g_1 \cdot g_2)$.
- Ef p er n-ta stigs margliða þá er $p \in \Theta(x^n)$.
- Ef p er n-ta stigs margliða og q er m-ta stigs margliða með $n < m$ þá er $p \in \mathcal{O}(q)$, en $q \notin \mathcal{O}(p)$.

Dæmi

- Tökum nokkur dæmi.
- Ef $f \in \mathcal{O}(g)$ og $r > 0$ þá er $r \cdot f \in \mathcal{O}(g)$.
- Ef $f_1 \in \mathcal{O}(g_1)$ og $f_2 \in \mathcal{O}(g_2)$ þá er $f_1 \cdot f_2 \in \mathcal{O}(g_1 \cdot g_2)$.
- Ef p er n-ta stigs margliða þá er $p \in \Theta(x^n)$.
- Ef p er n-ta stigs margliða og q er m-ta stigs margliða með $n < m$ þá er $p \in \mathcal{O}(q)$, en $q \notin \mathcal{O}(p)$.
- Við höfum að $\log x \in \mathcal{O}(x)$ en $x \notin \mathcal{O}(\log x)$.

Dæmi

- Tökum nokkur dæmi.
- Ef $f \in \mathcal{O}(g)$ og $r > 0$ þá er $r \cdot f \in \mathcal{O}(g)$.
- Ef $f_1 \in \mathcal{O}(g_1)$ og $f_2 \in \mathcal{O}(g_2)$ þá er $f_1 \cdot f_2 \in \mathcal{O}(g_1 \cdot g_2)$.
- Ef p er n-ta stigs margliða þá er $p \in \Theta(x^n)$.
- Ef p er n-ta stigs margliða og q er m-ta stigs margliða með $n < m$ þá er $p \in \mathcal{O}(q)$, en $q \notin \mathcal{O}(p)$.
- Við höfum að $\log x \in \mathcal{O}(x)$ en $x \notin \mathcal{O}(\log x)$.
- Nú er $\log x^n = n \cdot \log x$, svo ef p er n-ta stigs margliða þá er $\log p \in \mathcal{O}(\log x)$.

Í forritun

- Takið eftir að í stað þessa að segja „$f \in \mathcal{O}(g)$'' er oft sagt $f = \mathcal{O}(g)$.

Í forritun

- Takið eftir að í stað þessa að segja „$f \in \mathcal{O}(g)$" er oft sagt $f = \mathcal{O}(g)$.
- Ef forrit framkvæmir $T(n)$ aðgerðir, með $T \in \mathcal{O}(f)$, segjum við að *tímaflækja* (e. *time complexity*) forritsins sé $\mathcal{O}(f)$.

Í forritun

- Takið eftir að í stað þessa að segja „$f \in \mathcal{O}(g)$" er oft sagt $f = \mathcal{O}(g)$.
- Ef forrit framkvæmir $T(n)$ aðgerðir, með $T \in \mathcal{O}(f)$, segjum við að *tímaflækja* (e. *time complexity*) forritsins sé $\mathcal{O}(f)$.
- Skoðum nú nokkur forrit og ákvörðum tímaflækjur þeirra.

```c
#include <stdio.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

- Hér er ekkert inntak svo forritið hefur tímaflækjuna $\mathcal{O}(\quad)$.

```c
#include <stdio.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

- Hér er ekkert inntak svo forritið hefur tímaflækjuna $\mathcal{O}(\ 1\ )$.

```c
#include <stdio.h>

int main()
{
    int i, n, r = 0;
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
        r += i;
    printf("%d\n", r);
    return 0;
}
```

- Forritið les inn töluna n og reiknar svo summuna $1 + 2 + \cdots + n$.

```c
#include <stdio.h>

int main()
{
    int i, n, r = 0;
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
        r += i;
    printf("%d\n", r);
    return 0;
}
```

- Forritið les inn töluna n og reiknar svo summuna $1 + 2 + \cdots + n$.
- Þetta er gert með forlykkju sem keyrir n sinnum.

```c
#include <stdio.h>

int main()
{
    int i, n, r = 0;
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
        r += i;
    printf("%d\n", r);
    return 0;
}
```

- Forritið les inn töluna n og reiknar svo summuna $1 + 2 + \cdots + n$.
- Þetta er gert með forlykkju sem keyrir n sinnum.
- Hvert skipti eru tvær tölur lagðar saman.

```c
#include <stdio.h>

int main()
{
    int i, n, r = 0;
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
        r += i;
    printf("%d\n", r);
    return 0;
}
```

- Forritið les inn töluna n og reiknar svo summuna $1 + 2 + \cdots + n$.
- Þetta er gert með forlykkju sem keyrir n sinnum.
- Hvert skipti eru tvær tölur lagðar saman.
- Svo þetta forrit er $\mathcal{O}(\quad)$.

```c
#include <stdio.h>

int main()
{
    int i, n, r = 0;
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
        r += i;
    printf("%d\n", r);
    return 0;
}
```

- Forritið les inn töluna n og reiknar svo summuna $1 + 2 + \cdots + n$.
- Þetta er gert með forlykkju sem keyrir n sinnum.
- Hvert skipti eru tvær tölur lagðar saman.
- Svo þetta forrit er $\mathcal{O}(\ n\ )$.

```c
#include <stdio.h>

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        for (j = 0; j < i/2; j++)
            r += i^j;
    printf("%d\n", r);
    return 0;
}
```

- Forritið les inn töluna n og reiknar svo summu.

```c
#include <stdio.h>

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        for (j = 0; j < i/2; j++)
            r += i^j;
    printf("%d\n", r);
    return 0;
}
```

- Forritið les inn töluna *n* og reiknar svo summu.
- Summan er reiknuð með tvöfaldri forlykkju.

```c
#include <stdio.h>

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        for (j = 0; j < i/2; j++)
            r += i^j;
    printf("%d\n", r);
    return 0;
}
```

- Forritið les inn töluna n og reiknar svo summu.
- Summan er reiknuð með tvöfaldri forlykkju.
- Ytri forlykkjan keyrir n sinnum og seinni keyrir aldrei oftar en $n/2$ sinnum.

```c
#include <stdio.h>

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        for (j = 0; j < i/2; j++)
            r += i^j;
    printf("%d\n", r);
    return 0;
}
```

- Forritið les inn töluna n og reiknar svo summu.
- Summan er reiknuð með tvöfaldri forlykkju.
- Ytri forlykkjan keyrir n sinnum og seinni keyrir aldrei oftar en $n/2$ sinnum.
- Svo tímaflækja forritsins er $\mathcal{O}(\quad)$.

```c
#include <stdio.h>

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        for (j = 0; j < i/2; j++)
            r += i^j;
    printf("%d\n", r);
    return 0;
}
```

- Forritið les inn töluna n og reiknar svo summu.
- Summan er reiknuð með tvöfaldri forlykkju.
- Ytri forlykkjan keyrir n sinnum og seinni keyrir aldrei oftar en $n/2$ sinnum.
- Svo tímaflækja forritsins er $\mathcal{O}(n^2)$.

```c
#include <stdio.h>

int len(int n, int k)
{
    int r = 0;
    while (n > 0) r++, n /= k;
    return r;
}

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 2; i <= n; i++) r += len(n, i);
    printf("%d\n", r);
    return 0;
}
```

- Sjáum fyrst að lykkja í `main` keyrir $n-1$ sinnum.

```c
#include <stdio.h>

int len(int n, int k)
{
    int r = 0;
    while (n > 0) r++, n /= k;
    return r;
}

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 2; i <= n; i++) r += len(n, i);
    printf("%d\n", r);
    return 0;
}
```

- Sjáum fyrst að lykkja í `main` keyrir $n - 1$ sinnum.
- Hún kallar síðan á fallið `len(...)`.

```c
#include <stdio.h>

int len(int n, int k)
{
    int r = 0;
    while (n > 0) r++, n /= k;
    return r;
}

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 2; i <= n; i++) r += len(n, i);
    printf("%d\n", r);
    return 0;
}
```

- Sjáum fyrst að lykkja í `main` keyrir $n - 1$ sinnum.
- Hún kallar síðan á fallið `len(...)`.
- Það fall hefur eina `while`-lykkju sem deilir tölunni n með k, án afgangs, þar til n er orðin núll.

```c
#include <stdio.h>

int len(int n, int k)
{
    int r = 0;
    while (n > 0) r++, n /= k;
    return r;
}

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 2; i <= n; i++) r += len(n, i);
    printf("%d\n", r);
    return 0;
}
```

- Sjáum fyrst að lykkja í `main` keyrir $n - 1$ sinnum.
- Hún kallar síðan á fallið `len(...)`.
- Það fall hefur eina `while`-lykkju sem deilir tölunni n með k, án afgangs, þar til n er orðin núll.
- Þetta fall hefur því tímaflækjuna $\mathcal{O}(\quad)$.

```c
#include <stdio.h>

int len(int n, int k)
{
    int r = 0;
    while (n > 0) r++, n /= k;
    return r;
}

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 2; i <= n; i++) r += len(n, i);
    printf("%d\n", r);
    return 0;
}
```

- Sjáum fyrst að lykkja í `main` keyrir $n - 1$ sinnum.
- Hún kallar síðan á fallið `len(...)`.
- Það fall hefur eina `while`-lykkju sem deilir tölunni n með k, án afgangs, þar til n er orðin núll.
- Þetta fall hefur því tímaflækjuna $\mathcal{O}(\log n)$.

```c
#include <stdio.h>

int len(int n, int k)
{
    int r = 0;
    while (n > 0) r++, n /= k;
    return r;
}

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 2; i <= n; i++) r += len(n, i);
    printf("%d\n", r);
    return 0;
}
```

- Sjáum fyrst að lykkja í `main` keyrir $n - 1$ sinnum.
- Hún kallar síðan á fallið `len(...)`.
- Það fall hefur eina `while`-lykkju sem deilir tölunni n með k, án afgangs, þar til n er orðin núll.
- Þetta fall hefur því tímaflækjuna $\mathcal{O}(\log n)$.
- Í heildina hefur forritið því tímaflækjuna $\mathcal{O}(\qquad)$.

```c
#include <stdio.h>

int len(int n, int k)
{
    int r = 0;
    while (n > 0) r++, n /= k;
    return r;
}

int main()
{
    int i, j, n, r = 0;
    scanf("%d", &n);
    for (i = 2; i <= n; i++) r += len(n, i);
    printf("%d\n", r);
    return 0;
}
```

- Sjáum fyrst að lykkja í `main` keyrir $n - 1$ sinnum.
- Hún kallar síðan á fallið `len(...)`.
- Það fall hefur eina `while`-lykkju sem deilir tölunni n með k, án afgangs, þar til n er orðin núll.
- Þetta fall hefur því tímaflækjuna $\mathcal{O}(\log n)$.
- Í heildina hefur forritið því tímaflækjuna $\mathcal{O}(n \log n)$.

```c
#include <stdio.h>

int fib(int n)
{
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}

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

- Hér erum við með endurkvæmt fall.

```c
#include <stdio.h>

int fib(int n)
{
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}

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

- Hér erum við með endurkvæmt fall.
- Við sjáum að ef $n \geq 2$ þá kallar `fib(n)` tvisvar á sjálft sig.

```c
#include <stdio.h>

int fib(int n)
{
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}

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

- Hér erum við með endurkvæmt fall.
- Við sjáum að ef $n \geq 2$ þá kallar `fib(n)` tvisvar á sjálft sig.
- Svo þetta forrit er $\mathcal{O}(\quad)$.

```c
#include <stdio.h>

int fib(int n)
{
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}

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

- Hér erum við með endurkvæmt fall.
- Við sjáum að ef $n \geq 2$ þá kallar `fib(n)` tvisvar á sjálft sig.
- Svo þetta forrit er $\mathcal{O}(2^n)$.

```c
#include <stdio.h>

int fib(int n)
{
    if (n < 2) return n;
    return fib(n - 1) + fib(n - 2);
}

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

- Hér erum við með endurkvæmt fall.
- Við sjáum að ef $n \geq 2$ þá kallar `fib(n)` tvisvar á sjálft sig.
- Svo þetta forrit er $\mathcal{O}(2^n)$.
- Það er hægt að fá betra mat (við getum minnkað veldisstofninn).

- Skoðum skiladæmið *Reiknirit*.

- Skoðum skiladæmið *Reiknirit*.
- Fyrsta lína inntaksins inniheldur heiltölu n.

- Skoðum skiladæmið *Reiknirit*.
- Fyrsta lína inntaksins inniheldur heiltölu n.
- Síðan koma n heiltölur $a_1, a_2, \ldots, a_n$.

- Skoðum skiladæmið *Reiknirit*.
- Fyrsta lína inntaksins inniheldur heiltölu n.
- Síðan koma n heiltölur $a_1, a_2, \ldots, a_n$.
- Gerum ráð fyrir að við séum með forrit sem gerir eftirfarandi:

- Skoðum skiladæmið *Reiknirit*.
- Fyrsta lína inntaksins inniheldur heiltölu n.
- Síðan koma n heiltölur $a_1, a_2, \ldots, a_n$.
- Gerum ráð fyrir að við séum með forrit sem gerir eftirfarandi:
  - Prentar tölurnar.

- Skoðum skiladæmið *Reiknirit*.
- Fyrsta lína inntaksins inniheldur heiltölu n.
- Síðan koma n heiltölur $a_1, a_2, \ldots, a_n$.
- Gerum ráð fyrir að við séum með forrit sem gerir eftirfarandi:
    - Prentar tölurnar.
    - Fjarlægir öll eintök af algengustu tölunni í listan.

- Skoðum skiladæmið *Reiknirit*.
- Fyrsta lína inntaksins inniheldur heiltölu n.
- Síðan koma n heiltölur $a_1, a_2, \ldots, a_n$.
- Gerum ráð fyrir að við séum með forrit sem gerir eftirfarandi:
    - Prentar tölurnar.
    - Fjarlægir öll eintök af algengustu tölunni í listan.
    - Endurtekur skrefin að ofan þar til listinn er tómur.

- Skoðum skiladæmið *Reiknirit*.
- Fyrsta lína inntaksins inniheldur heiltölu n.
- Síðan koma n heiltölur $a_1, a_2, \ldots, a_n$.
- Gerum ráð fyrir að við séum með forrit sem gerir eftirfarandi:
  - Prentar tölurnar.
  - Fjarlægir öll eintök af algengustu tölunni í listan.
  - Endurtekur skrefin að ofan þar til listinn er tómur.
- Dæmið snýst um að finna hversu margar tölur eru prentaðar í heildina.

- Skoðum skiladæmið *Reiknirit*.
- Fyrsta lína inntaksins inniheldur heiltölu n.
- Síðan koma n heiltölur $a_1, a_2, \ldots, a_n$.
- Gerum ráð fyrir að við séum með forrit sem gerir eftirfarandi:
  - Prentar tölurnar.
  - Fjarlægir öll eintök af algengustu tölunni í listan.
  - Endurtekur skrefin að ofan þar til listinn er tómur.
- Dæmið snýst um að finna hversu margar tölur eru prentaðar í heildina.
- Tökum dæmi.

- Gerum ráð fyrir að tölurnar í inntakinu séu $1, 2, 1, 4, 4$.

- Gerum ráð fyrir að tölurnar í inntakinu séu $1, 2, 1, 4, 4$.
- Þá myndi forritið í dæminu prenta:

- Gerum ráð fyrir að tölurnar í inntakinu séu $1, 2, 1, 4, 4$.
- Þá myndi forritið í dæminu prenta:

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

- Gerum ráð fyrir að tölurnar í inntakinu séu $1, 2, 1, 4, 4$.
- Þá myndi forritið í dæminu prenta:

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

- Svo það prentar 9 tölur.

- Ein leið til að leysa þetta dæmi er að útfæra forritið.

- Ein leið til að leysa þetta dæmi er að útfæra forritið.
- Við þurfum þá að geta fjarlægt algengasta stakið.

```c
typedef long long ll;

int cmp(const void* p1, const void* p2)
{
    ll x = *(ll*)p1, y = *(ll*)p2;
    return (y <= x) - (x <= y);
}

ll fjarlaegja_algengasta_stakid(ll *a, ll n)
{ // fjarlaegir algengasta stakid og skilar nyju lengd listans.
    ll i = 0, j, mx = 0, r;
    while (i < n)
    { // finnum algengasta stakid i rodudu fylki.
        j = i;
        while (j < n && a[i] == a[j]) j++;
        if (mx < j - i) mx = j - i, r = a[i];
        i = j;
    }
    // fjarlaegjum algengasta stakid.
    for (i = 0, j = 0; i < n; i++) if (a[i] != r) a[j++] = a[i];
    return j;
}

int main()
{
    ll i, n, r = 0;
    scanf("%lld", &n);
    ll a[n];
    for (i = 0; i < n; i++) scanf("%lld", &a[i]);
    qsort(a, n, sizeof *a, cmp);
    while (n > 0)
    {
        r += n;
        n = fjarlaegja_algengasta_stakid(a, n);
    }
    printf("%lld\n", r);
    return 0;
}
```

- Þetta forrit leysir dæmið (að því sem ég best veit).

- Þetta forrit leysir dæmið (að því sem ég best veit).
- Nú er spurning hvort það sé nógu hratt.

- Þetta forrit leysir dæmið (að því sem ég best veit).
- Nú er spurning hvort það sé nógu hratt.
- Í dæminu á Kattis er gefið að $n \leq 10^6$.

- Þetta forrit leysir dæmið (að því sem ég best veit).
- Nú er spurning hvort það sé nógu hratt.
- Í dæminu á Kattis er gefið að $n \leq 10^6$.
- Reynum nú að ákvarða tímaflækjuna.

- Þetta forrit leysir dæmið (að því sem ég best veit).
- Nú er spurning hvort það sé nógu hratt.
- Í dæminu á Kattis er gefið að $n \leq 10^6$.
- Reynum nú að ákvarða tímaflækjuna.
- Sjáum aftur útfærsluna.

```
ll fjarlaegja_algengasta_stakid(ll *a, ll n)
{ // fjarlaegir algengasta stakid og skilar nyju lengd listans.
    ll i = 0, j, mx = 0, r;
    while (i < n)
    { // finnum algengasta stakid i rodudu fylki.
        j = i;
        while (j < n && a[i] == a[j]) j++;
        if (mx < j - i) mx = j - i, r = a[i];
        i = j;
    }
    // fjarlaegjum algengasta stakid.
    for (i = 0, j = 0; i < n; i++) if (a[i] != r) a[j++] = a[i];
    return j;
}

int main()
{
    ll i, n, r = 0;
    scanf("%lld", &n);
    ll a[n];
    for (i = 0; i < n; i++) scanf("%lld", &a[i]);
    qsort(a, n, sizeof *a, cmp);
    while (n > 0)
    {
        r += n;
        n = fjarlaegja_algengasta_stakid(a, n);
    }
    printf("%lld\n", r);
    return 0;
}
```

- Í fyrsta lagi röðum við listanum, sem er $\mathcal{O}(\qquad)$.

- Í fyrsta lagi röðum við listanum, sem er $\mathcal{O}(n \log n)$.

- Í fyrsta lagi röðum við listanum, sem er $\mathcal{O}(n \log n)$.
- Síðan erum við með `while`-lykkju, sem gerir ekkert annað en að kalla á fallið `fjarlaegja_algengasta_stakid(...)`.

- Í fyrsta lagi röðum við listanum, sem er $\mathcal{O}(n \log n)$.
- Síðan erum við með `while`-lykkju, sem gerir ekkert annað en að kalla á fallið `fjarlaegja_algengasta_stakid(...)`.
- Í versta falli fjarlægjum við eitt stak í einu, svo þessi `while`-lykkja keyrir allt að n sinnum.

- Í fyrsta lagi röðum við listanum, sem er $\mathcal{O}(n \log n)$.
- Síðan erum við með `while`-lykkju, sem gerir ekkert annað en að kalla á fallið `fjarlaegja_algengasta_stakid(...)`.
- Í versta falli fjarlægjum við eitt stak í einu, svo þessi `while`-lykkja keyrir allt að n sinnum.
- Í fallinu `fjarlaegja_algengasta_stakid(...)` eru tvær lykkjur, hvor um sig keyrir n sinnum.

- Í fyrsta lagi röðum við listanum, sem er $\mathcal{O}(n \log n)$.
- Síðan erum við með `while`-lykkju, sem gerir ekkert annað en að kalla á fallið `fjarlaegja_algengasta_stakid(...)`.
- Í versta falli fjarlægjum við eitt stak í einu, svo þessi `while`-lykkja keyrir allt að n sinnum.
- Í fallinu `fjarlaegja_algengasta_stakid(...)` eru tvær lykkjur, hvor um sig keyrir n sinnum.
- Takið eftir að n er þá lengd listans á þeim tímapunkti.

- Í fyrsta lagi röðum við listanum, sem er $\mathcal{O}(n \log n)$.
- Síðan erum við með `while`-lykkju, sem gerir ekkert annað en að kalla á fallið `fjarlaegja_algengasta_stakid(...)`.
- Í versta falli fjarlægjum við eitt stak í einu, svo þessi `while`-lykkja keyrir allt að n sinnum.
- Í fallinu `fjarlaegja_algengasta_stakid(...)` eru tvær lykkjur, hvor um sig keyrir n sinnum.
- Takið eftir að n er þá lengd listans á þeim tímapunkti.
- Fallið `fjarlaegja_algengasta_stakid(...)` hefur því tímaflækjuna $\mathcal{O}(\quad)$.

- Í fyrsta lagi röðum við listanum, sem er $\mathcal{O}(n \log n)$.
- Síðan erum við með `while`-lykkju, sem gerir ekkert annað en að kalla á fallið `fjarlaegja_algengasta_stakid(...)`.
- Í versta falli fjarlægjum við eitt stak í einu, svo þessi `while`-lykkja keyrir allt að n sinnum.
- Í fallinu `fjarlaegja_algengasta_stakid(...)` eru tvær lykkjur, hvor um sig keyrir n sinnum.
- Takið eftir að n er þá lengd listans á þeim tímapunkti.
- Fallið `fjarlaegja_algengasta_stakid(...)` hefur því tímaflækjuna $\mathcal{O}(\ n\ )$.

- Í fyrsta lagi röðum við listanum, sem er $\mathcal{O}(n \log n)$.
- Síðan erum við með `while`-lykkju, sem gerir ekkert annað en að kalla á fallið `fjarlaegja_algengasta_stakid(...)`.
- Í versta falli fjarlægjum við eitt stak í einu, svo þessi `while`-lykkja keyrir allt að n sinnum.
- Í fallinu `fjarlaegja_algengasta_stakid(...)` eru tvær lykkjur, hvor um sig keyrir n sinnum.
- Takið eftir að n er þá lengd listans á þeim tímapunkti.
- Fallið `fjarlaegja_algengasta_stakid(...)` hefur því tímaflækjuna $\mathcal{O}(\ n\ )$.
- Forritið í heild sinni hefur því tímaflækjuna $\mathcal{O}(\quad)$.

- Í fyrsta lagi röðum við listanum, sem er $\mathcal{O}(n \log n)$.
- Síðan erum við með `while`-lykkju, sem gerir ekkert annað en að kalla á fallið `fjarlaegja_algengasta_stakid(...)`.
- Í versta falli fjarlægjum við eitt stak í einu, svo þessi `while`-lykkja keyrir allt að n sinnum.
- Í fallinu `fjarlaegja_algengasta_stakid(...)` eru tvær lykkjur, hvor um sig keyrir n sinnum.
- Takið eftir að n er þá lengd listans á þeim tímapunkti.
- Fallið `fjarlaegja_algengasta_stakid(...)` hefur því tímaflækjuna $\mathcal{O}(\ n\ )$.
- Forritið í heild sinni hefur því tímaflækjuna $\mathcal{O}(n^2)$.

- Rifjum upp að $n \leq 10^6$ og tímaflækjan er $\mathcal{O}(n^2)$.

- Rifjum upp að $n \leq 10^6$ og tímaflækjan er $\mathcal{O}(n^2)$.
- Er þetta nógu hratt til að fá `AC`?

- Rifjum upp að $n \leq 10^6$ og tímaflækjan er $\mathcal{O}(n^2)$.
- Er þetta nógu hratt til að fá `AC`?
- Nú segir 10^8 *relgan* okkur að tímamörk dæmisins þurfi að vera $(10^6)^2 \cdot 10^{-8} = 10^4$ sekúndur.

- Rifjum upp að $n \leq 10^6$ og tímaflækjan er $\mathcal{O}(n^2)$.
- Er þetta nógu hratt til að fá `AC`?
- Nú segir 10^8 *relgan* okkur að tímamörk dæmisins þurfi að vera $(10^6)^2 \cdot 10^{-8} = 10^4$ sekúndur.
- Svo er ekki og þessi lausn er því að fara að fá `TLE`.

- Rifjum upp að $n \leq 10^6$ og tímaflækjan er $\mathcal{O}(n^2)$.
- Er þetta nógu hratt til að fá `AC`?
- Nú segir 10^8 *relgan* okkur að tímamörk dæmisins þurfi að vera $(10^6)^2 \cdot 10^{-8} = 10^4$ sekúndur.
- Svo er ekki og þessi lausn er því að fara að fá `TLE`.

ID	DATE	PROBLEM	STATUS	CPU	LANG
	TEST CASES				
8283421	18:18:36	Reiknirit	✖ **Time Limit Exceeded**	> 2.00 s	C