

Deila og drottna

Bergur Snorrason

6. febrúar 2021

Almennar nálganir lausna

- ▶ Þegar við leysum dæmi í keppnisforritun notumst við oftast við eina af eftirfarandi aðferðum:
 - ▶ *Ad hoc.*
 - ▶ *Tæmandi leit* eða *ofbeldis aðferðin* (e. *complete search, brute force*),
 - ▶ *Gráðug reiknirit* (e. *greedy algorithms*),
 - ▶ *Deila og drottna* (e. *divide and conquer*),
 - ▶ *Kvik bestun* (e. *dynamic programming*).
- ▶ Í síðustu vikum fjölluðum við um *Ad hoc* dæmi, *tæmandi leit* og *gráðug reiknirit*.
- ▶ Í þessari viku fjöllum við um *deila og drottna reiknirit* og *kvika bestun*.

Deila og drottna

- ▶ Sum dæmi má endurkvæmt skipta upp þangað til þau verða fáfengileg.
- ▶ Síðan má líma fáfengilegu lausnirnar saman í heildarlausn í lokinn.
- ▶ Slík reiknirit kallast *deila og drottna* reiknirit.
- ▶ Þessi flokkur er sjaldgæfastur.
- ▶ Það eru þó mörg þekkt reiknirit sem nýta sér deila og drottna.

Deila og drottna, þekkt dæmi

- ▶ Mergesort.
- ▶ Helmingunarleit (e. *binary search*).
- ▶ Þriðjungaleit (e. *ternary search*).
- ▶ Margföldunar reiknirit Karatsuba.
- ▶ Margföldunar reiknirit Strassen.
- ▶ Nálægustu punktar í plani.
- ▶ Fourier ummyndun (e. *fast Fourier transform* (FFT)).

Merge

- ▶ Gerum ráð fyrir að við séum með tvo raðaða lista, a og b .
- ▶ Búum til nýjan, tóman lista c .
- ▶ Berum saman fremstu stök a og b og tökum minna stakið og setjum aftast í c .
- ▶ Endurtökum þangað til a eða b er tómur.
- ▶ Skeytum svo því sem er eftir aftan á c .
- ▶ Nú inniheldur c þau stök sem voru í a og b áður.
- ▶ Einnig er c raðaður.
- ▶ Ef fjöldi staka í a og b er n þá er þetta $\mathcal{O}(n)$.

```
a = [1, 2, 5, 6, 8, 9]
b = [0, 3, 4, 7, 10]
c = []
```

```
a = [1, 2, 5, 6, 8, 9]  
b = [3, 4, 7, 10]  
c = [0]
```

a = [2, 5, 6, 8, 9]

b = [3, 4, 7, 10]

c = [0, 1]


```
a = [5, 6, 8, 9]
b = [3, 4, 7, 10]
c = [0, 1, 2]
```

```
a = [5, 6, 8, 9]
b = [4, 7, 10]
c = [0, 1, 2, 3]
```

a = [5, 6, 8, 9]

b = [7, 10]

c = [0, 1, 2, 3, 4]

a = [6, 8, 9]

b = [7, 10]

c = [0, 1, 2, 3, 4, 5]

a = [8, 9]

b = [7, 10]

c = [0, 1, 2, 3, 4, 5, 6]

a = [8, 9]

b = [10]

c = [0, 1, 2, 3, 4, 5, 6, 7]

a = [9]

b = [10]

c = [0, 1, 2, 3, 4, 5, 6, 7, 8]

```
a = []
```

```
b = [10]
```

```
c = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```



```
a = []
```

```
b = []
```

```
c = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Mergesort

- ▶ Við getum notað þessa aðferð til að raða almennum lista.
- ▶ Við skiptum listanum okkar í tvo jafna hluta og köllum endurkvæmt á fallið okkar þangað til við erum með tómann lista.
- ▶ Á leiðinni upp úr endurkvæmninni sameinum við svo helmingana eins og rætt var á undan.

```
1 #include <stdio.h>
2 #include <time.h>
3 #include <stdlib.h>
4 #include <assert.h>
5
6 void merge(int* a, int l, int m, int r)
7 {
8     int i = l, j = m, b[r - l], c = 0;
9     while (i < m && j < r)
10     {
11         if (a[j] < a[i]) b[c++] = a[j++];
12         else b[c++] = a[i++];
13     }
14     while (i < m) b[c++] = a[i++];
15     while (j < r) b[c++] = a[j++];
16     for (i = l; i < r; i++) a[i] = b[i - l];
17 }
18
19 void mergesort(int* a, int l, int r)
20 {
21     if (r - l < 2) return;
22     int m = (l + r)/2;
23     mergesort(a, l, m), mergesort(a, m, r);
24     merge(a, l, m, r);
25 }
26
27 int main()
28 {
29     srand(time(NULL));
30     int i, n;
31     scanf("%d", &n);
32     int a[n];
33     for (i = 0; i < n; i++) a[i] = rand()%(10*n);
34     mergesort(a, 0, n);
35     for (i = 0; i < n - 1; i++) assert(a[i] <= a[i + 1]);
36     return 0;
37 }
```

Mergesort

- ▶ Mergesort er sígilt dæmi um deila og drottna reiknirit.
- ▶ Við helmingum alltaf listann og tökum svo saman í línulegum tíma.
- ▶ Hvert stak kemur fyrir í $\mathcal{O}(\log n)$ sameiningum, svo reikniritið er $\mathcal{O}(n \log n)$.
- ▶ Þetta er mjög algeng tímaflækja í deila og drottna reikniritum.

Helmingunarleit

- ▶ Helmingunarleit er yfirleitt sett fram sem leit í röðuðum lista.
- ▶ Skoðum það fyrst og alhæfum svo.
- ▶ Látum a vera raðaðan lista af n tölum.
- ▶ Gerum ráð fyrir að við viljum finna t í listanum.
- ▶ Látum $m = \lfloor n/2 \rfloor$.
- ▶ Ef m -ta stakið í a er stærra en t þá getur t ekki verið í seinni helming listans.
- ▶ Ef m -ta stakið í a er minna en t þá getur t ekki verið í fyrri helming listans.
- ▶ Svo við getum útilokað helming listans í hverri ítrun.

```

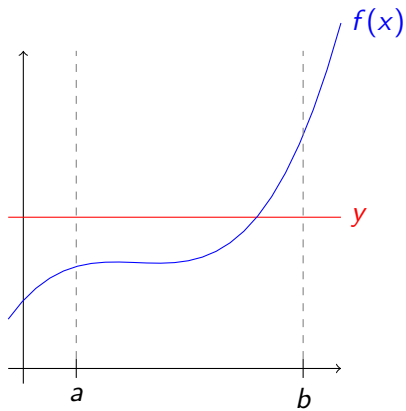
1 #include <stdio.h>
2
3 int bs(int* a, int t, int l, int r)
4 {
5     if (r - l == 1) return a[l] == t ? l : -1;
6     int m = (l + r)/2;
7     if (a[m] <= t) return bs(a, t, m, r);
8     else return bs(a, t, l, m);
9 }
10
11 int main()
12 {
13     int i, n, q, x;
14     scanf("%d%d", &n, &q);
15     int a[n];
16     for (i = 0; i < n; i++) scanf("%d", &a[i]);
17     while (q-- != 0)
18     {
19         scanf("%d", &x);
20         printf("%d\n", bs(a, x, 0, n));
21     }
22     return 0;
23 }

```

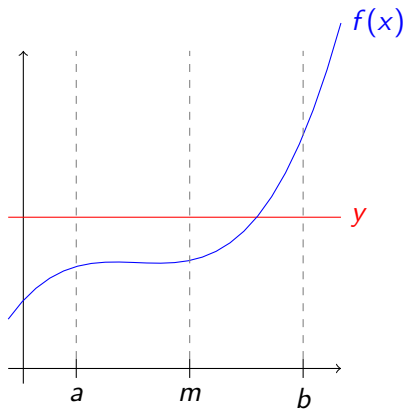
- ▶ Helmingunarleit er $\mathcal{O}(\log n)$, þar sem við helmingum stærð listans í hverri ítrun.
- ▶ Góð æfing í helmingurleit er að útfæra leitina þannig að hún skili vísi á fyrstu (eða síðustu) endurtekningu staksins.
- ▶ Slíkar útgáfur að helmingunarleit nýtast þegar við förum að nota helmingurleit í almennari mynd.

- ▶ Gerum ráð fyrir að við séum með vaxandi fall $f: [a, b] \rightarrow \mathbb{R}$ og $y \in \mathbb{R}$.
- ▶ Getum við fundið $x \in [a, b]$ þannig að $f(x) = y$?
- ▶ Slíkt x þarf ekki að vera til, en ef f er samfelld er það til ef $y \in [f(a), f(b)]$.
- ▶ En hvernig finnum við slíkt x ?
- ▶ Við getum notað nákvæmlega sömu hugmynd og í helmingurleit í lista.
- ▶ Látum $m = (a + b)/2$.
- ▶ Ef $f(m) > t$ þá þarf $x \in [a, m]$.
- ▶ Ef $f(m) < t$ þá þarf $x \in [m, b]$.

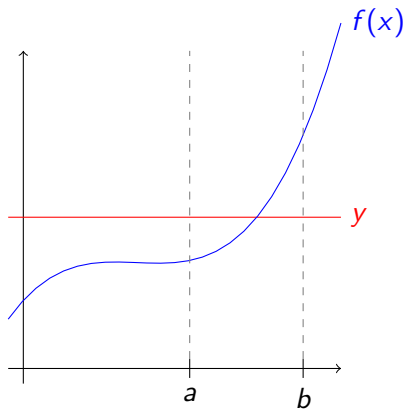
Sýnidæmi



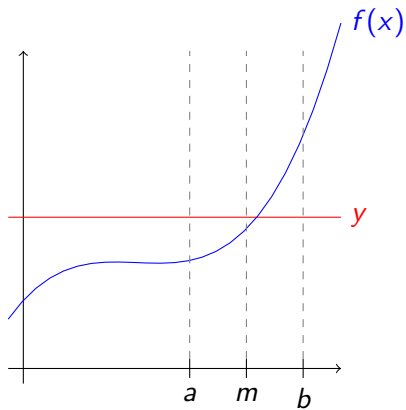
Sýnidæmi



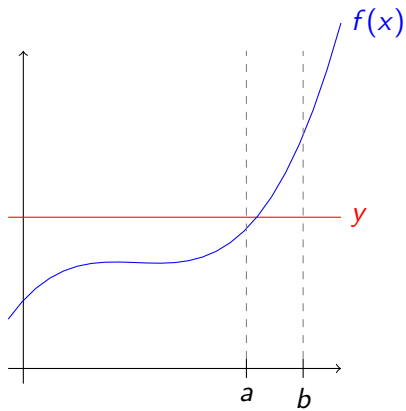
Sýnidæmi



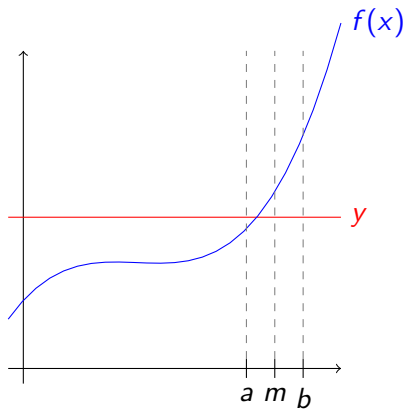
Sýnidæmi



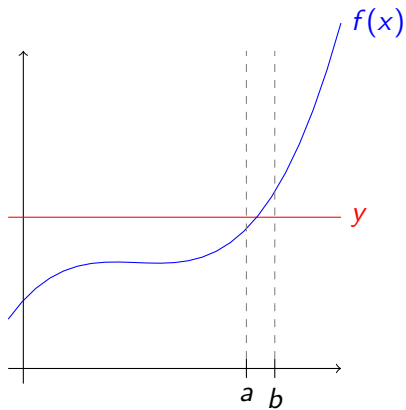
Sýnidæmi



Sýnidæmi



Sýnidæmi



- ▶ Takið eftir við munum ekki beint finna $x \in [a, b]$ þannig að $f(x) = y$.
- ▶ Það sem við finnum er $x \in [a, b]$ þannig að $|f(x) - y| < \varepsilon$, fyrir hvaða ε sem vera skal.
- ▶ Það er þó aldrei ætlast til annars í keppnisforritun og skekkjan er alltaf gefin í úttakslýsingu dæma.

- ▶ Við getum alhæft frekar.
- ▶ Tökum dæmi.
- ▶ Þú átt k ketti og n bæli fyrir kettina þína, með $k \leq n$.
- ▶ Öll bælin eru staðsett á gangi í íbúðinni þinni.
- ▶ Ganginum má lýsa sem talnalínu og staðsetningar kattabælanna eru þá tölurnar $0 < x_1 < x_2 < \dots < x_n < 10^9$ á talnalínunni.
- ▶ En kettir eru einfatar svo þeir vilja hafa sem mesta fjarlægð í næsta kött.
- ▶ Þú átt að raða köttum á bælin þannig að nálægustu kettirnir eru sem lengst frá hvorum öðrum.

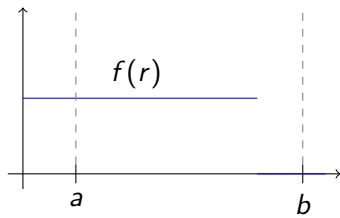
- ▶ Þetta dæmi má leysa með helmingunarleit.
- ▶ Byrjum á að svara annari spurningu:
 - ▶ Er hægt að raða köttunum þannig að nálægustu kettirnir séu að minnsta kosti með fjarlægð r ?
- ▶ Þetta dæmi má leysa gráðugt.
- ▶ Við töpum aldrei á því að setja kött á bælið staðsett á x_1 .
- ▶ Við megum þá ekki setja kött á bæli sem liggja í $[x_1, x_1 + r]$.
- ▶ Útilokum þau og veljum minnsta bælið sem er eftir, og endurtökum.
- ▶ Ef við komum fyrir k , eða fleiri, köttum svona þá er svarið við nýju spurningunni “já”, en annars “nei”.
- ▶ Þetta tekur $\mathcal{O}(n + k)$.

- ▶ Tökum þó eftir að ef við komum fyrir öllum köttunum með fjarlægð r_0 þá gerum við það líka fyrir $r < r_0$.
- ▶ Skilgreinum fall

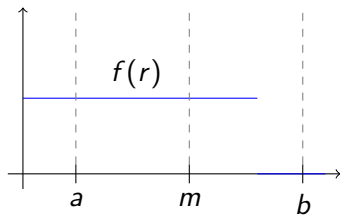
$$f(r) = \begin{cases} 1, & \text{ef koma má fyrir } k, \\ & \text{eða fleiri, köttum með fjarlægð } r \\ 0, & \text{annars.} \end{cases}$$

- ▶ Við getum nú umorðað upprunarlega dæmið sem: “Finnið stærsta r þannig að $f(r) = 1$ ”.
- ▶ En nú er fallið f minnkandi (samkvæmt efsta punktinum á glærunni), svo við getum fundið slíkt gildi með helmingunar leit.

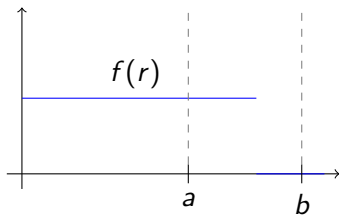
Sýnidæmi



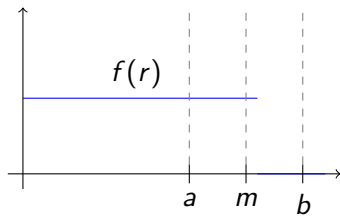
Sýnidæmi



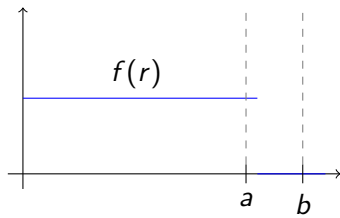
Sýnidæmi



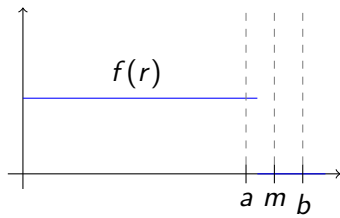
Sýnidæmi



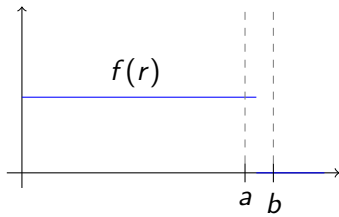
Sýnidæmi



Sýnidæmi



Sýnidæmi



- ▶ Ef við látum $M = 10^9/\varepsilon$, þar sem ε er leyfileg skekkja í dæminu, þá er lausnin $\mathcal{O}((k + n) \log M)$.
- ▶ Hér gerum við ekki ráð fyrir að svarið sé heiltala.
- ▶ Ef við gerum ráð fyrir því verður tímaflækjan eins, nema með $M = 10^9$.

- ▶ Það sem við gerðum í raun var að breyta dæminu úr “finnið minnsta/stærsta gildið þannig að...” yfir í “tökum ákveðið gildi og athugum hvort að...”.
- ▶ Þetta er algengasta notkunin á helmingunarleit í keppnisforritun.
- ▶ Algengt er helmingurleit af þessum toga sé hluti af erfiðum dæmum.

Priðjungaleit

- ▶ Gerum ráð fyrir að við séum með kúpt fall $f: [a, b] \rightarrow \mathbb{R}$.
- ▶ Munið að fall er kúpt ef
$$f(tx_1 + (1 - t)x_2) \leq tf(x_1) + (1 - t)f(x_2),$$
fyrir öll $t \in [0, 1]$ og $x_1, x_2 \in [a, b]$.
- ▶ Hvernig finnum við útgildi (há- og lággildi) fallsins?
- ▶ Auðvelt er að finna hágildi.
- ▶ Gerum ráð fyrir að $f(a) \leq f(b)$.
- ▶ Fyrir öll $x \in [a, b]$ gildir þá að

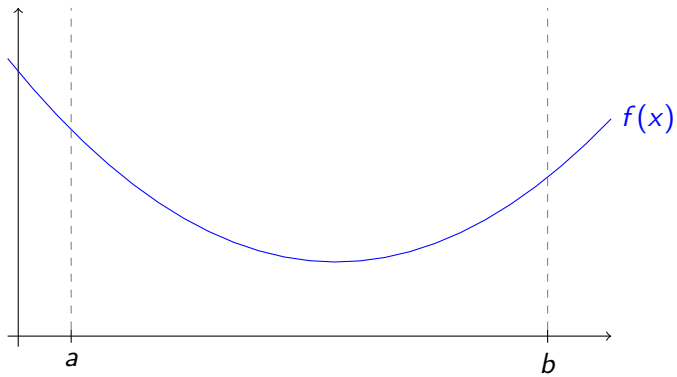
$$\begin{aligned}f(x) &= f(at + (1 - t)b) \leq tf(a) + (1 - t)f(b) \\&\leq tf(b) + (1 - t)f(b) = f(b),\end{aligned}$$

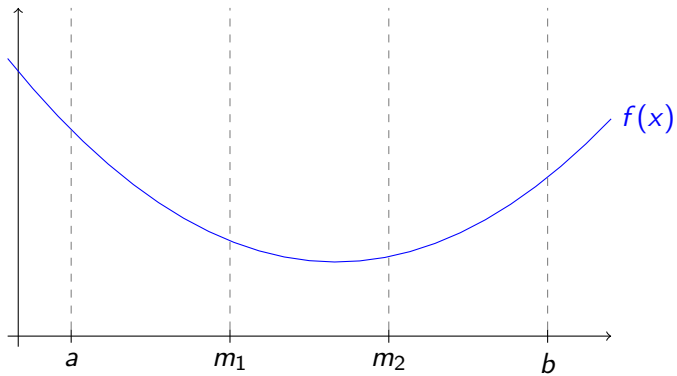
með $t = (x - b)/(a - b)$.

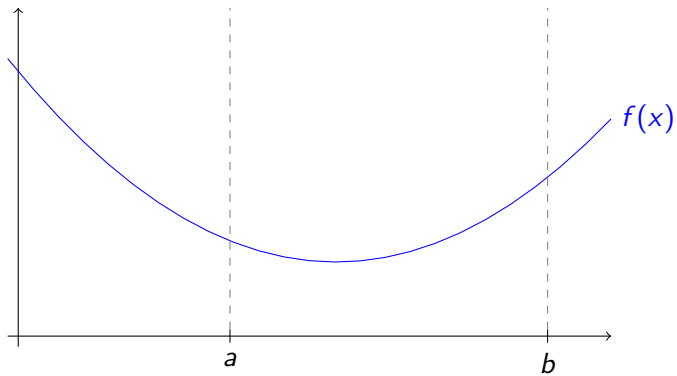
- ▶ Svo hágildi fæst í endapunktunum a eða b .

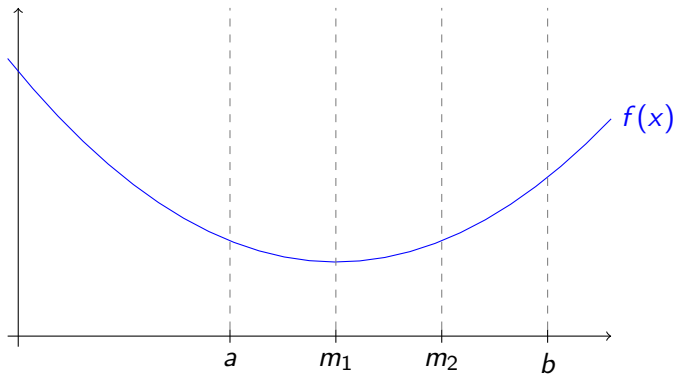
Priðjungaleit

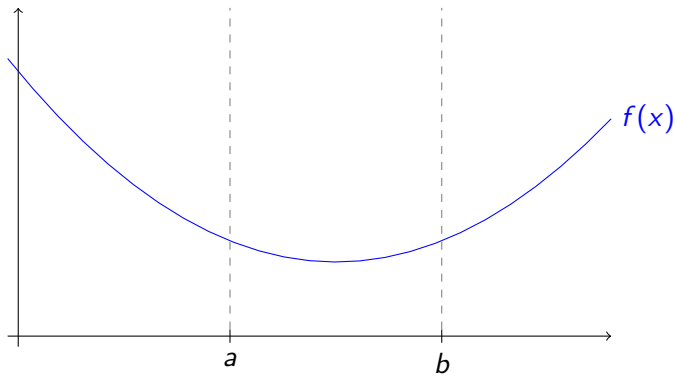
- ▶ Við getum fundið lággildið tölulega, svipað og með helmingunarleit.
- ▶ Við veljum punkta $m_1, m_2 \in [a, b]$ þannig að bilin $[a, m_1]$, $[m_1, m_2]$ og $[m_2, b]$ séu jafn löng.
- ▶ Við skoðum svo fallgildin $f(m_1)$ og $f(m_2)$.
- ▶ Ef $f(m_1) < f(m_2)$ þá getur lággildið ekki legið á bilinu $[m_2, b]$, svo við getum útilokað það bil í leitinni.
- ▶ Ef $f(m_2) < f(m_1)$ þá getur lággildið ekki legið á bilinu $[a, m_1]$, svo við getum útilokað það bil í leitinni.
- ▶ Ef $f(m_2) = f(m_1)$ þá þarf lággildið að liggja á bilinu $[m_1, m_2]$.
- ▶ Þetta stafar allt af því að kúpt föll taka hágildi í öðru hvorum endapunkta sinna.

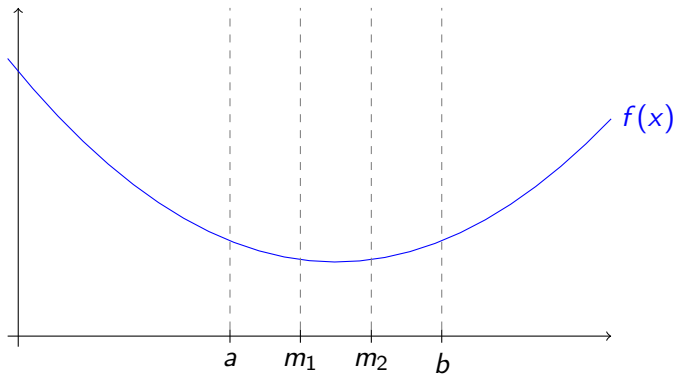


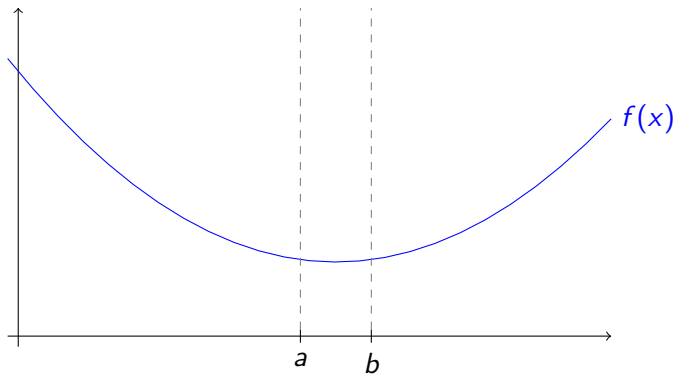


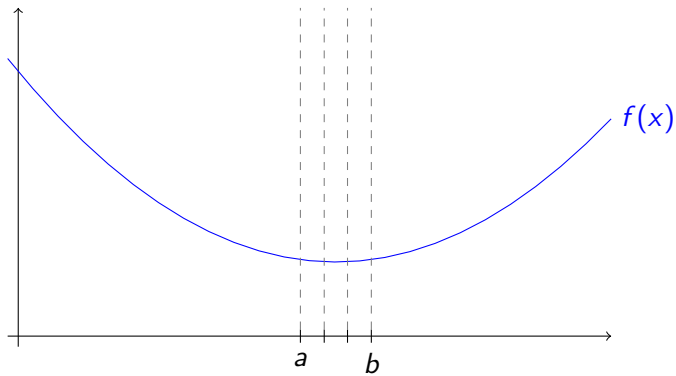


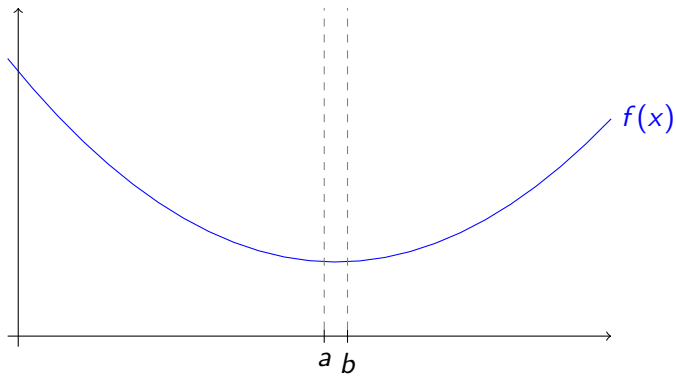












- ▶ Við notum okkur oft að tvídifffranlegt fall er kúpt ef og aðeins ef önnur afleiðan er jákvæð.
- ▶ Umfjöllunin okkar heimfærist á eðlilegan hátt yfir á hvelf föll.
- ▶ Þriðjungaleit er algeng í rúmfræði því Evklíðska firðinni er kúpt.


```
1 #include <stdio.h>
2 #define EPS 1e-9
3
4 double f(double x)
5 {
6     return 5.0 - 1.2*x + 0.1*x*x;
7 }
8
9 int main()
10 {
11     double a = 1.0, b = 10.0, m1, m2;
12     while (b - a > EPS)
13     {
14         m1 = (a + a + b)/3.0;
15         m2 = (a + b + b)/3.0;
16         if (f(m1) > f(m2)) a = m1;
17         else b = m2;
18     }
19     printf("f(%.2f) = %.2f\n", a, f(a));
20     return 0;
21 }
```

- ▶ Tökum eitt hefðbundið sýnidæmi í lokinn.
- ▶ Þú vilt skortselja gjaldmiðil, vitandi framtíðargengi, þannig að þú græðir sem mestan pening.
- ▶ Nánar, þú ætlar að fá lánaðar 100 danskar krónur á einhverjum degi og skipta þeim um leið í íslenskar krónur.
- ▶ Eftir einhvern fjölda daga þarft þú svo að kaupa 100 danskar krónur til að borga lánið, ásamt því að borga K íslenskar krónur á dag í lánaðakostnað.
- ▶ Hver er mesti fjöldi íslenska króna sem þú getur grætt?
- ▶ Fyrsta lína inntaksins inniheldur tvær tölur, $1 \leq n \leq 10^5$ og $1 \leq k \leq 100$.
- ▶ Næsta lína inniheldur n heiltölur $1 \leq x_1, x_2, \dots, x_n \leq 10^5$.
- ▶ Hér tákna x_i fjölda íslenska króna sem ein dönsk króna kostar á i -ta degi.

► Skoðum sýniinntök.

```
1 Sample input 1
2 5 10
3 1000 980 960 940 10
4
5
6 Sample input 2
7 5 100
8 100 100 100 103 100
```

```
Sample output 1
98950
```

```
Sample output 2
100
```

- Í fyrra tilfellinu viljum við taka lánið á fyrsta degi og borga það á síðasta degi.
- Við fáum $100 \cdot 1000 = 10^5$ íslenskar krónur á fyrst degi og borgum $100 \cdot 10 = 10^3$ íslenskar krónur á síðasta degi.
- Við borgum svo $5 \cdot 10 = 50$ íslenskar krónur í lánakostnað.
- Svo við endum með $10^5 - 10^3 - 50 = 98950$ íslenskar krónur.

► Skoðum sýniinntök.

```
1 Sample input 1
2 5 10
3 1000 980 960 940 10
4
5
6 Sample input 2
7 5 100
8 100 100 100 103 100
```

```
Sample output 1
98950
```

```
Sample output 2
100
```

- Í seinna tilfellinu viljum við taka lánið á fjórða degi og borga það á síðasta degi.
- Við fáum $100 \cdot 103 = 10300$ íslenskar krónur á fyrst degi og borgum $100 \cdot 100 = 10^4$ íslenskar krónur á síðasta degi.
- Við borgum svo $2 \cdot 100 = 200$ íslenskar krónur í lánakostnað.
- Svo við endum með $10300 - 10^4 - 200 = 100$ íslenskar krónur.

- ▶ Til að nota deila og drottna þurfum við að taka eftir einu.
- ▶ Gerum ráð fyrir að engin lánaðkostnaður sé greiddur síðasta.
- ▶ Það einfaldar reikninga og við getum alltaf bætt honum við eftir á.
- ▶ Táknum með $f(i, j)$ þann gróða (eða tap) sem fæst með því að taka lánið á i -ta degi og borga það á j -degi degi, og $g(i)$ vera gengið á i -ta degi.
- ▶ Við fáum nú að $f(i, j) = 100 \cdot g(i) - 100 \cdot g(j) - (j - i) \cdot k$.
- ▶ Ef a , b og c eru heiltölur þannig að $1 \leq a < b < c \leq n$ þá fæst

$$\begin{aligned}
 f(a, b) + f(b, c) &= 100 \cdot (g(a) - g(b)) - (b - a) \cdot k \\
 &\quad + 100 \cdot (g(b) - g(c)) - (c - b) \cdot k \\
 &= 100 \cdot (g(a) - g(c)) - (c - a) \cdot k \\
 &= f(a, c).
 \end{aligned}$$

- ▶ Látum nú $m = \lfloor n/2 \rfloor$.
- ▶ Þá gildir eitt af þrennu, fyrir tiltekna lausn:
 - ▶ Báðir endapunktur bestu lausnarinnar liggja á $[1, m - 1]$.
 - ▶ Báðir endapunktur bestu lausnarinnar liggja á $[m, n]$.
 - ▶ Annar endapunktur bestu lausnarinnar liggur á $[1, m - 1]$ og hinn liggur á $[m, n]$.
- ▶ Fyrri tvö tilfellin má leysa með einfaldri endurkvæmni.
- ▶ Fyrir síðasta tilfellið nýtum við gegnvirknina af fyrri glærunni.
- ▶ Við viljum finna bestu lausnina sem liggur í gegnum m -ta stakið.
- ▶ Gegnvirknin segir þó að okkur nægir að finna fyrst bestu lausnina sem endar í m -ta stakinu, finna svo bestu lausnina sem byrjar í m -ta stakinu og sameina svo lausnirnar.

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 typedef long long ll;
4 ll max(ll a, ll b) { if (a < b) return b; return a; }
5
6 ll foo(ll* a, ll l, ll r, ll k)
7 {
8     if (r - l < 5)
9     {
10         ll i, j, mx = 0;
11         for (i = l; i < r; i++) for (j = i + 1; j < r; j++)
12             mx = max(mx, 100*(a[i] - a[j]) - k*(j - i));
13         return mx;
14     }
15     ll m = (l + r)/2, i, j;
16     ll v1 = foo(a, l, m, k), v2 = foo(a, m, r, k), mx1 = 0, mx2 = 0;
17     for (i = l; i < m; i++) mx1 = max(mx1, 100*(a[i] - a[m]) - k*(m - i));
18     for (i = m; i < r; i++) mx2 = max(mx2, 100*(a[m] - a[i]) - k*(i - m));
19     return max(max(v1, v2), mx1 + mx2);
20 }
21
22 int main()
23 {
24     ll i, j;
25     int x, n, k;
26     scanf("%d%d", &n, &k);
27     ll a[n];
28     for (i = 0; i < n; i++)
29     {
30         scanf("%d", &x);
31         a[i] = x;
32     }
33     printf("%lld\n", max(0, foo(a, 0, n, k) - k));
34     return 0;
35 }

```

