

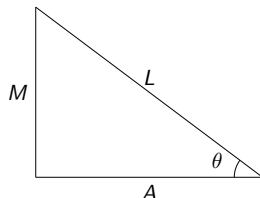
Rúmfræði

Bergur Snorrason

28. mars 2022

Hornaföll

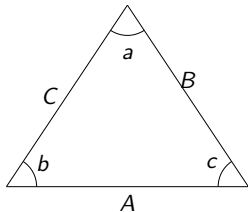
- ▶ Þessi glæra ætti að vera upprifjun fyrir flest ykkar.
- ▶ Þríhyrningur er sagður rétthyrndur ef eitt horna hans er 90° .
- ▶ Fyrir rétthyrnda þríhyrninga gildir:
 - ▶ $\frac{A}{L} = \cos \theta$.
 - ▶ $\frac{M}{L} = \sin \theta$.
 - ▶ $\frac{M}{A} = \frac{M}{L} \frac{L}{A} = \frac{\sin \theta}{\cos \theta} = \tan \theta$.
- ▶ Einnig gildir regla Pýþagorasar, $L^2 = A^2 + M^2$.



- ▶ Almennar gildir um þríhyrninga:

- ▶ $\frac{\sin a}{A} = \frac{\sin b}{B} = \frac{\sin c}{C}$ (sínus reglan).
- ▶ $A^2 = B^2 + C^2 - 2BC \cos a$ (kósínus reglan)

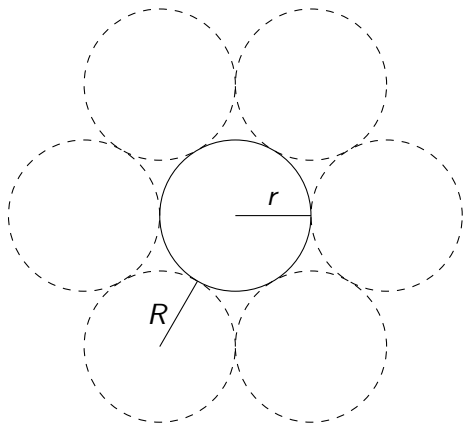
- ▶ **Æfing:** Sannið reglu Pýthagorasar með kósínus reglunni.



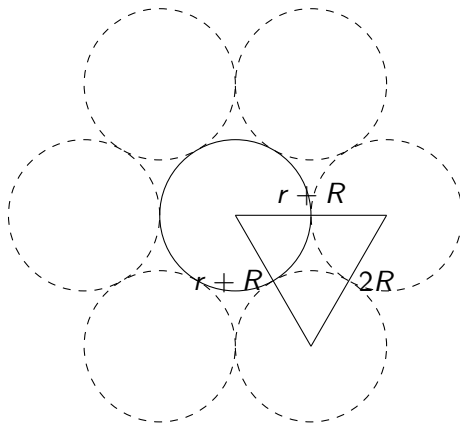
Sýnidæmi: NN and the Optical Illusion - Codeforces

- ▶ Þér eru gefið heiltölu n og rauntölu r .
- ▶ Þú teiknar hring á blað með geilsa r .
- ▶ Síðan vilt þú teikna n jafn stóra hringi í kringum hringinn þinn þannig að þeir skeri hringinn þinn og aðlæga hringi í nákvæmlega einum punkti.
- ▶ Hver þarf geilsu ytri hringjanna að vera?
- ▶ <https://codeforces.com/problemset/problem/1100/C>

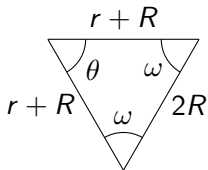
Ef $n = 6$ fæst eftirfarandi mynd, þar sem R er svarið.



Sjáum að fjarlægðin frá miðju myndarinnar að miðju ytri hringjanna er $r + R$. Við fáum því eftirfarandi jafnarma þríhyrning.

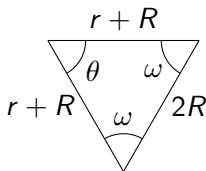


► Nú er $\theta = \frac{360^\circ}{n}$ og $\omega = \frac{180^\circ - \theta}{2}$.



- Símus reglan gefur okkur loks að

$$\begin{aligned}\frac{2R}{\sin \theta} &= \frac{r + R}{\sin \omega} \Rightarrow 2R \sin \omega = r \sin \theta + R \sin \theta \\ &\Rightarrow 2R \sin \omega - R \sin \theta = r \sin \theta \\ &\Rightarrow R = \frac{r \sin \theta}{2 \sin \omega - \sin \theta}.\end{aligned}$$



Tvinntölur

- ▶ Skilgreinum mengið $\mathbb{C} := \mathbb{R} \times \mathbb{R}$.
- ▶ Skilgreinum svo samlagningu á \mathbb{C} þannig að fyrir $(a, b), (c, d) \in \mathbb{C}$ þá er

$$(a, b) + (c, d) = (a + c, b + d).$$

- ▶ Skilgreinum svo margföldun á \mathbb{C} þannig að fyrir $(a, b), (c, d) \in \mathbb{C}$ þá er

$$(a, b) \cdot (c, d) = (ac - bd, ad + bc).$$

- ▶ Við táknum iðulega $(0, 1) \in \mathbb{C}$ með i og $(x, y) \in \mathbb{C}$ með $x + yi$.
- ▶ Takið eftir að $(x, y) = (x, 0) + i \cdot (y, 0)$.
- ▶ Tölurnar í \mathbb{C} köllum við *tvinntölur*.

- ▶ Ef $z = x + yi \in \mathbb{C}$ þá...
 - ▶ ...köllum við x *raunhluta* z og y *þverhluta* z .
 - ▶ ...er *lengd* z gefin með $|z| = \sqrt{x^2 + y^2}$.
 - ▶ ...köllum við $x - yi$ *samoka* z , táknað \bar{z} .
 - ▶ ...köllum hornið sem (x, y) myndar við jákvæða hluta x -ás í $(0, 0)$ *stefnuhorn* z og táknum það með $\text{Arg } z$.

- ▶ Látum nú $z, w \in \mathbb{C}$.
- ▶ Þá er rúmfræðileg túlkun $z + w$ einfaldlega hliðrun á z um w (eða öfugt).
- ▶ Einnig, ef $|w| = 1$ þá er rúmfræðileg túlkun $z \cdot w$ snúningur á z í kringum 0 um $\text{Arg } w$ gráður.
- ▶ Ef $|z| = r$ og $\text{Arg } z = \theta$ þá skrifum við oft $z = re^{i\theta}$.
- ▶ Ef $z = r_1 e^{i\theta_1}$ og $w = r_2 e^{i\theta_2}$ þá er $z \cdot w = r_1 r_2 e^{i(\theta_1 + \theta_2)}$.
- ▶ Þetta eru dæmi um hvernig við getum stytt okkur leiðir í rúmfræði með því að nota tvinntölur.
- ▶ Fleiri (minna augljós) dæmi koma á eftir.

Rúmfræði í forritun

- ▶ Við munum bara fjalla um tvívíða rúmfræði í þessum fyrirlestri.
- ▶ Þegar leysa þarf þrívíð rúmfræði dæmi er oft góð hugmynd að byrja á að leysa dæmin í tveimur víddum (ef unnt er) og reyna svo að yfirfæra tvívíðu lausnina í þriðju víddina.
- ▶ Hingað til í námskeiðinu höfum við að mestu fengist við heiltölur og stöku sinnum þurft að vinna með fleytitölur.
- ▶ Í rúmfræði þurfum við þó oft að vinna með fleytitölur.
- ▶ Þegar við notum fleytitölur er mikilvægt að passa að samanburðir er ekki fullkomnir.
- ▶ Við látum því duga að tvær tölur sé *nógu* líkar, í vissum skilningi, til að þær séu jafnar.

Fleytitölu samanburður

```
3 #define EPS 1e-9
4
5 int eq(double a, double b)
6 { // Eru |a| og |b| nógu líkar?
7     return fabs(a - b) < EPS;
8 }
9
10 int neq(double a, double b)
11 { // Eru |a| og |b| nógu ólíkar?
12     return fabs(a - b) >= EPS;
13 }
```

Punktar

- ▶ Þegar kemur að því að geyma punkta í forritun munum við notast við tvinntölur.
- ▶ Þó þessi aðgerð gæti verið nokkuð heimulleg þá er hún þægileg og fljótleg í útfærslu.
- ▶ Hennar helsti kostur er að grunnaðgerðir á tvinntölum eru útfærðar í mörgum forritunarmálum.

Notkun á `complex.h` úr \mathbb{C} í rúmfræði

- ▶ Fallið `creal(p)` skilar x -hniti p .
- ▶ Fallið `cimag(p)` skilar y -hniti p .
- ▶ Fallið `cabs(p)` skilar fjarlægð p frá $(0, 0)$.
- ▶ Fallið `cabs(p - q)` skilar fjarlægð milli p og q .
- ▶ Fallið `carg(p)` skilar stefnuhorninu p .
- ▶ Fallið `conj(p)` speglar p um x -ás.

- ▶ Við notum svo `typedef double complex pt;`
- ▶ Eftir það getum við skilgreint punktar með `pt p = x + I*y;`
- ▶ Þessi punktur svarar til punktsins (x, y) .

Sýnidæmi

- ▶ Skoðum dæmi.
- ▶ Þú byrjar í $(0, 0)$ og færð gefnar skipanir.
- ▶ Skipanirnar eru allar einn bókstafur og ein tala.
- ▶ Ef skipunin er...
 - ▶ ...f x gengur þú áfram um x metra.
 - ▶ ...b x gengur þú aftur á bak um x metra.
 - ▶ ...r x snýrð þú þér um x radíana til hægri.
 - ▶ ...l x snýrð þú þér um x radíana til vinstri.
- ▶ Hversu langt ertu frá $(0, 0)$, eftir að hafa fylgt öllum skipununum.

- ▶ Ef við erum í $p \in \mathbb{C}$ og viljum taka r metra skref í stefnu θ getum við einfaldlega lagt $re^{i\theta}$ við p .
- ▶ Hvert við snúum í upphafi skiptir ekki mál því það hefur ekki áhrif á fjarlægðinni til $(0, 0)$.

```
4 typedef double complex pt;
5
6 int main()
7 {
8     int n;
9     double x, r = 0.0;
10    pt p = 0;
11    scanf("%d", &n);
12    while (n--)
13    {
14        char c = getchar();
15        while (c != 'f' && c != 'b' && c != 'l' && c != 'r') c = getchar();
16        scanf("%lf", &x);
17        if (c == 'f') p += x*cexp(1*r);
18        else if (c == 'b') p -= x*cexp(1*r);
19        else if (c == 'l') r += x;
20        else if (c == 'r') r -= x;
21        else assert(0);
22    }
23    printf("%.8f\n", cabs(p));
24    return 0;
25 }
```

Línur

- ▶ Tveir ólíkir punktar skilgreina nákvæmlega eina línu.
- ▶ Svo við getum einfaldlega sagt að lína sé tveir ólíkir punktar.
- ▶ Helsti ókostur þessa aðferðar er að sama línan getur verið skilgreind með mismunandi pörum af punktum.
- ▶ Stundum hentar betur að skilgreina línu sem skurðpunkt við y -ás og hallatölu.
- ▶ Þá er einfaldara að bera saman línur en það þarf að höndla sérstaklega línur samsíða y -ás.

Línustrik

- ▶ Venjan er að skilgreina línustrik sem par punkta, nánar tiltekið endapunkta línustriksins.
- ▶ Markmið okkar í þessum hluta af fyrirlestrinu verður að útfæra fall sem finnur fjarlægð milli línustrika.
- ▶ Við munum byrja á að útfæra góð hjálparföll sem nýtast meðal annars í að finna þessa fjarlægð, en eru einnig hentug í öðrum dæmum.

► Við munum útfæra:

- Fall sem skoðar hvort tvö bil skerist (`bxb(...)`).
- Fall sem skoðar hvort línustrik skerist (`1x1(...)`).
- Fall sem finnur stystu fjarlægð punkts og línustriks (`p2l(...)`).
- Fall sem finnur stystu fjarlægð tveggja línustrika (`12l(...)`).

Skurður bila

- Þegar við viljum skoða skurð tveggja bila nægir okkur að skoða hvort annar endapunktur bils er í hinu bilinu.

```
8 int bxb(double a, double b, double c, double d)
9 { // Skerast [a, b] og [c, d]?
10   if (a > b) return bxb(b, a, c, d);
11   if (c > d) return bxb(a, b, d, c);
12   return fmax(a, c) < fmin(b, d) + EPS;
13 }
```

Skurður tveggja línustrika

- ▶ Látum a, b, c, d vera endapunkta línustrikanna.
- ▶ Við viljum vita hvort punktarnir c og d séu hvor sínum megin við línuna gegnum punktanna a og b .
- ▶ Við getum gert þetta með því að skoða í hvora áttina við beygjum ef göngum frá a til b og svo til c , og síðan frá a til b og svo til d .

- ▶ Það eru leiðinleg sértílfelli þegar þrír af endapunktunum liggja á sömu línunni.
- ▶ Ég mun eftirláta ykkur að laga þessi sértílfelli, því þar sem við höfum aðallega áhuga á að finna fjarlægð línubila nægir að segja að línustrikin skerist ekki í þessu sértílfelli.
- ▶ Þetta mun skýrast betur á eftir.

```

15 int ccw(pt a, pt b, pt c)
16 { // Í hvora áttina er verið að beygja?
17     double f = cimag((c - a)/(b - a));
18     if (fabs(f) < EPS) return 0;
19     return f < EPS ? -1 : 1;
20 }
21
22 int lxl(pt a, pt b, pt c, pt d)
23 { // Skerast <a, b> og <c, d>?
24     int a1 = ccw(a, b, c), a2 = ccw(a, b, d),
25         a3 = ccw(c, d, a), a4 = ccw(c, d, b);
26     if (a1*a2*a3*a4 == 0) return 0; // Tæknilega séð ekki rétt.
27     if (a1*a2 != -1 || a3*a4 != -1) return 0;
28     return bxb(creal(a), creal(b), creal(c), creal(d))
29         && bxb(cimag(a), cimag(b), cimag(c), cimag(d));
30 }

```

Fjarlægð punkts og línustriks

- ▶ Til að finna fjarlægð frá punkti að línustriki getum við nýtt okkur að fjarlægðin breytist ekki ef við hliðrum og snúum punktunum.
- ▶ Við byrjum því á að hliðra þannig að annar endapunktur línustriksins verði $(0, 0)$.
- ▶ Við getum svo snúið um $(0, 0)$ þannig að línustrikið verði samsíða x -ásnum.
- ▶ Ef punkturinn liggur nú beint fyrir ofan eða neðan línustrikið þá er y -hnit punktsins (án formerkis) fjarlægðin frá línustrikinu.
- ▶ Annars þurfum við að skoða hvor endapunktur línustriksins er nær punktinum.

```
32 double p2l(pt p, pt l1, pt l2)
33 { // Finnur fjarlægð frá punkti í línustrik.
34     p = (p - l1)*cexp(-l*carg(l2 - l1));
35     if (-EPS < creal(p) && creal(p) < cabs(l2 - l1) + EPS)
36         return fabs(cimag(p));
37     return fmin(cabs(p), cabs(p - cabs(l2 - l1)));
38 }
```

Fjarlægð milli tveggja línustrika

- ▶ Gefum okkur línustrikin $\langle a, b \rangle$ og $\langle c, d \rangle$.
- ▶ Ef þau skerast þá er fjarlægðin á milli þeirra 0.
- ▶ Gerum því ráð fyrir að þau skerist ekki.
- ▶ Þá er bersýnilega fjarlægð línustrikana minnst á milli
 - ▶ a og $\langle c, d \rangle$,
 - ▶ b og $\langle c, d \rangle$,
 - ▶ c og $\langle a, b \rangle$ eða
 - ▶ d og $\langle a, b \rangle$.

- ▶ Ég hef ekki minnst á það hingað til, en við höfum gert ráð fyrir að endapunktur línustrikana séu mismunandi.
- ▶ Ef $a = b$ og $c = d$ þá er fjarlægð línustrikana fjarlægðin milli a og c .
- ▶ Ef $a = b$ og $c \neq d$ þá er fjarlægð línustrikana fjarlægðin milli a og $\langle c, d \rangle$.
- ▶ Ef $a \neq b$ og $c = d$ þá er fjarlægð línustrikana fjarlægðin milli c og $\langle a, b \rangle$.

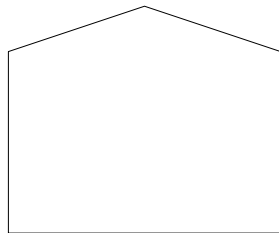
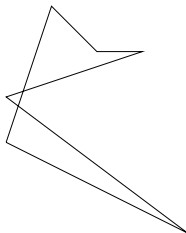
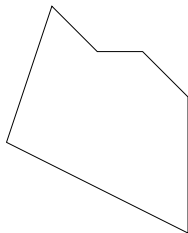
```



40 double l2l(pt a1, pt a2, pt b1, pt b2)
41 { // Finnur fjarlægð frá línustrikinu <a1, a2> til <b1, b2>.
42   if (cabs(a1 - a2) < EPS && cabs(b1 - b2) < EPS)
43     return cabs(a1 - b1);
44   if (cabs(a1 - a2) < EPS) return p2l(a1, b1, b2);
45   if (cabs(b1 - b2) < EPS) return p2l(b1, a1, a2);
46   if (lxl(a1, a2, b1, b2)) return 0.0;
47   return fmin(fmin(p2l(a1, b1, b2), p2l(a2, b1, b2)),
48              fmin(p2l(b1, a1, a2), p2l(b2, a1, a2)));
49 }

```

Marghyrningar

- ▶ *Marghyrningur* er samfelldur, lokaður ferill í plani sem samanstendur af endanlega mörgum beinum línustrikum.
- ▶ Ef ferillinn er einfaldur þá kallast marghyrningurinn *einfaldur*.
- ▶ Marghyrningar eru sagðar vera *kúptur* ef sérhver beina lína, sem liggur ekki ofan á hlið marghyrningsins, sker mest tvo punkta á marghyrningnum.



- ▶ Þegar við viljum tákna marghyrning í tölvu notum við einfaldlega hornpunkta hans.
- ▶ Röð punktanna skiptir máli.  
- ▶ Til þæginda geymum við oft einn punkt tvisvar, nánar tiltekið er fremsti og aftasti punkturinn eins.
- ▶ Þetta er því við höfum oft meira áhuga á línustrikunum milli hornpunktanna heldur en hornpunktunum sjálfum.
- ▶ Þar sem marghyrningar er mjög vinsælir í keppnum munum við fara í nokkur atriði sem er gott að kunna.

Ummál marghyrnings

- ▶ Ummála marghyrnings er einfalt að reikna í línulegum tíma.
- ▶ Við leggjum bara saman allar hliðarlengdirnar.

```
19 double ummal(pt* p, int n)
20 { // p[0] == p[n - 1]
21     int i;
22     double r = 0.0;
23     for (i = 0; i < n - 1; i++) r += cabs(p[i] - p[i + 1]);
24     return r;
25 }
```

Flatarmál marghyrninga

- ▶ Ummál marghyrninga er þó ekki jafnt algengt í keppnum og flatarmál marghyrninga.
- ▶ Það er einnig auðvelt að reikna flatarmálið í línulegum tíma, þó það sé ekki endilega augljóst að þetta skili flatarmálinu.
- ▶ Fyrir áhugasama er hægt að nota setningu Green til að leiða út eftirfarandi forritsbút.

```
10 double flatarmal(pt* p, int n)
11 { // p[0] == p[n - 1]
12     int i;
13     double r = 0.0;
14     for (i = 0; i < n - 1; i++)
15         r += creal(p[i])*cimag(p[i + 1]) - creal(p[i + 1])*cimag(p[i]);
16     return fabs(0.5*r);
17 }
```

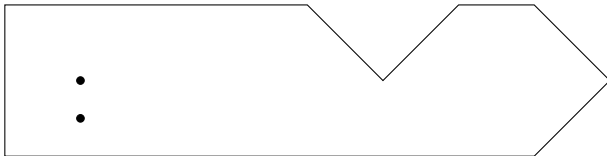
- ▶ Við þurfum að skila tölugildinu því við getum fengið neikvætt flatarmál.
- ▶ Neikvætt flatarmál hljómar kannski furðulega en það fellur eðlilega úr sönnun summunar ef notast er við setningu Green.
- ▶ Til að nota hana þarf að reikna ferilheildi og útkoman úr ferilheildum skiptir um formerki þegar breytt er um átt stikunar ferilsins.
- ▶ Þetta þýðir að formerki x eftir forlykkjuna er jákvætt ef punktar p eru gefnir rangsælis og neikvætt ef þeir eru gefnir réttsælis.

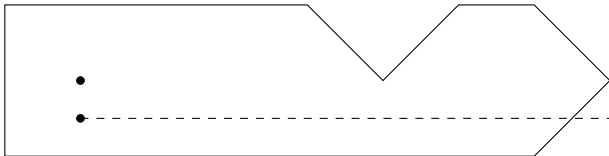
Punktur í marghyrning

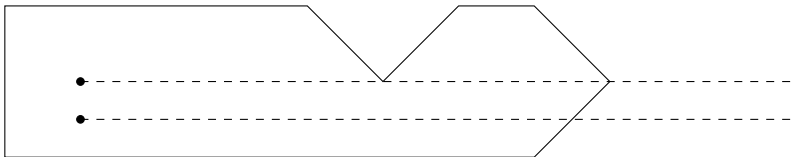
- ▶ Að ákvarða hvort punktur sé inni í marghyrning er algengt undirvandamál í rúmfræði dæmum.
- ▶ Aðallega er gengist við tvær aðferðir til að leysa slík dæmi, sú fyrri er að nota *geislarakningu* (e. *raytracing*) og hin er að reikna summu aðliggjandi horna marghyrningsins miðað við punktinn.
- ▶ Það er svo til þriðja, hraðari, aðferð sem virkar bara ef marghyrnurinn er kúptur
- ▶ Hún byggist á að helmingunarleita stefnuhornið punktsins.

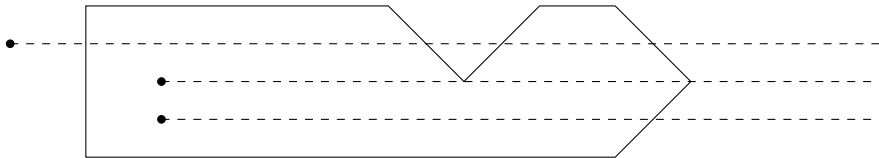
Geislarakning

- ▶ Við getum dregið geisla frá punktinum sem við viljum skoða í einhverja átt og talið hversu oft við skerum jaðar marghyrningsins.
- ▶ Ef við erum fyrir utan marghyrninginn og skerum jaðarinn erum við inni í honum, en ef við erum fyrir innan og skerum jaðarinn erum við fyrir utan (þetta er í raun skilgreining á því hvenær geislinn *sker* marghyrninginn).
- ▶ Svo ef við skerum jaðarinn slétt tölu sinnum er punkturinn fyrir innan, og annars fyrir utan (setning Jordan).
- ▶ Við getum látið geislann vera línustrik, nógu langt til að vera út fyrir marghyrninginn, og notað síðan $1 \times 1(\dots)$ til að ákvarða í $\mathcal{O}(n)$ hversu oft geislinn sker marghyrninginn.





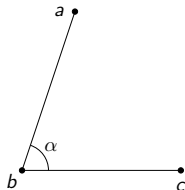


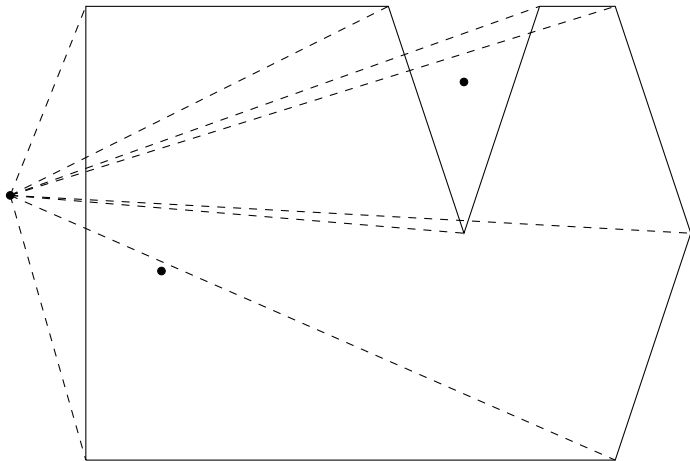


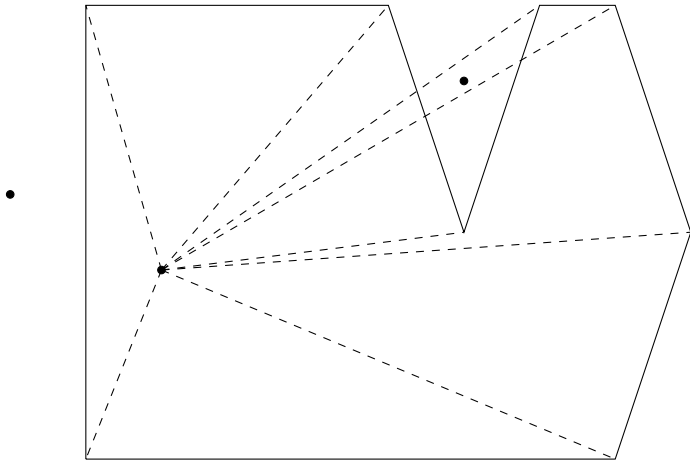
- ▶ Þessi aðferð er með nokkur sértilfelli sem gerir hana óþægilega í útfærslu.
- ▶ Öll sértilfellin eiga það sameiginlegt að vera þegar geislinn sker endapunkta línustrika marghyrningsins.
- ▶ Ef marghyrningurinn er kúptur er nokkuð auðvelt að eiga við þessi sértilfelli, en það gildir ekki í flestum dæmum.
- ▶ Þessi aðferð verður því ekki úrfærð hér.

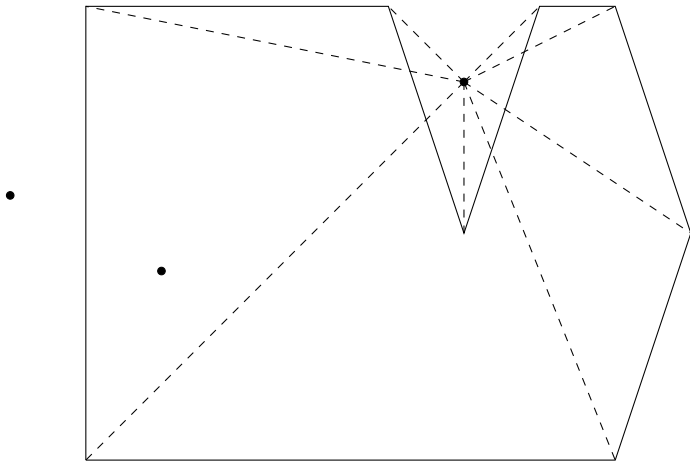
Afstæð hornasumma

- ▶ Látum p_i , $i < n$, tákna hornpunkta marghyrnings, q einhvern punkt, $\alpha(a, b, c)$ vera hornið milli a , b og c og $\beta(a, b, c)$ vera 1 ef brotna línustrikið $\langle a, b, c \rangle$ „beygir” til vinstri en -1 annars.
- ▶ *Afstæð hornsumma marghyrnings með tilliti til punkts q er $\sum_{i=0}^n \beta(q, p_i, p_{i+1})\alpha(p_i, q, p_{i+1})$.*
- ▶ Ef q er inni í marghyrningnum þá er þessi summa bersýnilega 2π .
- ▶ Ef q er fyrir utan marghyrninginn þá verður summan hins vegar 0.









```

10 double angle(pt a, pt o, pt b)
11 { // Fallið alpha(a, b, c) í glærunum.
12     double r = fabs(carg(a - o) - carg(b - o));
13     return r < M_PI ? r : 2*M_PI - r;
14 }
15
16 int ccw(pt a, pt b, pt c)
17 { // Fallið beta(a, b, c) í glærunum.
18     if (fabs(a - b) < EPS || fabs(cimag((c - a)/(b - a))) < EPS) return 0;
19     return cimag((c - a)/(b - a)) > 0.0 ? 1 : -1;
20 }
21
22 int is_in(pt* p, pt q, int n)
23 { // Er punkturinn q inni í marghyrningnum p?
24     int i;
25     double s = 0.0;
26     for (i = 0; i < n - 1; i++)
27         s += ccw(q, p[i], p[i + 1]) * angle(p[i], q, p[i + 1]);
28     return fabs(s) > M_PI;
29 }

```


Kútpur hjúpur punktastarfs

- ▶ *Kúptur hjúpur punktastarfs* er minnsti kúpti marghyrningur sem inniheldur all punkta punktastarfsins.
- ▶ Maður getur ímyndað sér að maður taki teygju og strekki hana yfir punkta safnið og sleppi henni svo.
- ▶ Við munum nota aðferð sem kallast *Graham's scan* til að finna kúpta hjúp punktastarfs.

Kútpur hjúpur punktastarfs

- ▶ *Kúptur hjúpur punktastarfs* er minnsti kúpti marghyrningur sem inniheldur all punkta punktastarfsins.
- ▶ Maður getur ímyndað sér að maður taki teygju og strekki hana yfir punkta safnið og sleppi henni svo.
- ▶ Við munum nota aðferð sem kallast *Graham's scan* til að finna kúpta hjúp punktastarfs.

Kútpur hjúpur punktastarfs

- ▶ *Kúptur hjúpur punktastarfs* er minnsti kúpti marghyrningur sem inniheldur all punkta punktastarfsins.
- ▶ Maður getur ímyndað sér að maður taki teygju og strekki hana yfir punkta safnið og sleppi henni svo.
- ▶ Við munum nota aðferð sem kallast *Graham's scan* til að finna kúpta hjúp punktastarfs.

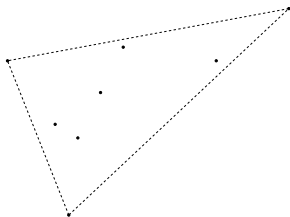
Kútpur hjúpur punktastarfs

- ▶ *Kúptur hjúpur punktastarfs* er minnsti kúpti marghyrningur sem inniheldur all punkta punktastarfsins.
- ▶ Maður getur ímyndað sér að maður taki teygju og strekki hana yfir punkta safnið og sleppi henni svo.
- ▶ Við munum nota aðferð sem kallast *Graham's scan* til að finna kúpta hjúp punktastarfs.



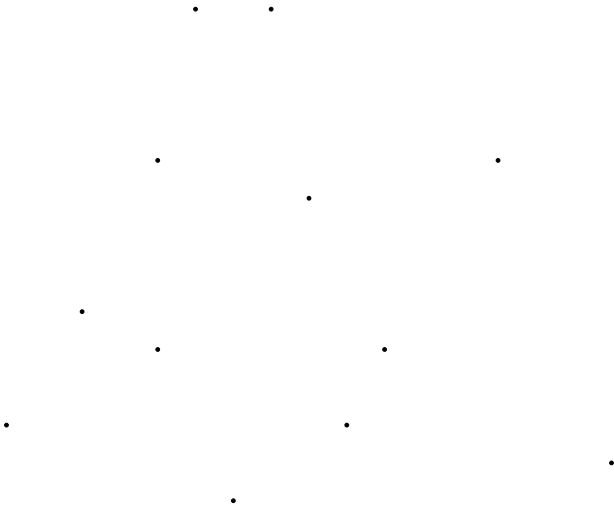
Kútpur hjúpur punktastarfs

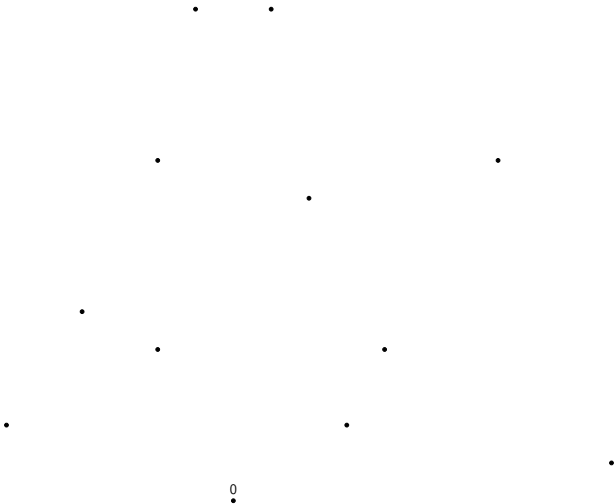
- ▶ *Kúptur hjúpur punktastarfs* er minnsti kúpti marghyrningur sem inniheldur all punkta punktastarfsins.
- ▶ Maður getur ímyndað sér að maður taki teygju og strekki hana yfir punkta safnið og sleppi henni svo.
- ▶ Við munum nota aðferð sem kallast *Graham's scan* til að finna kúpta hjúp punktastarfs.

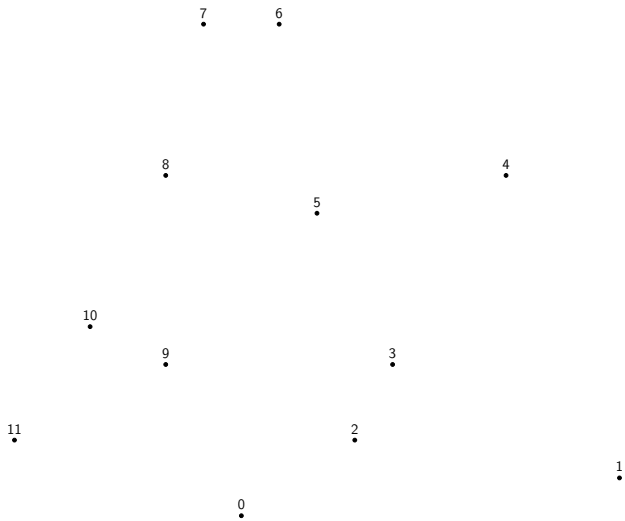


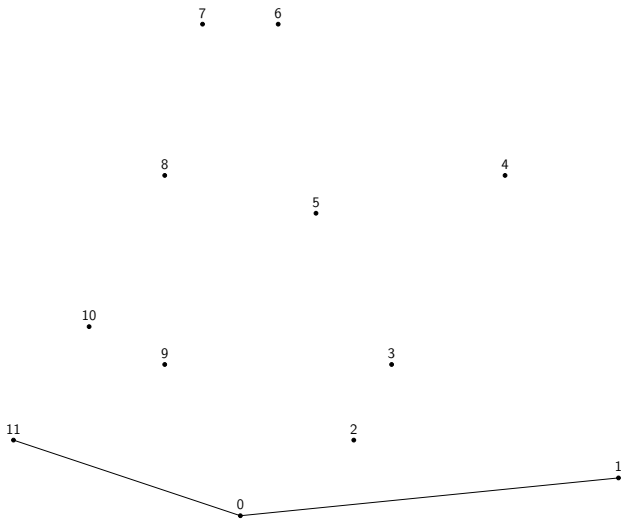
Graham's Scan

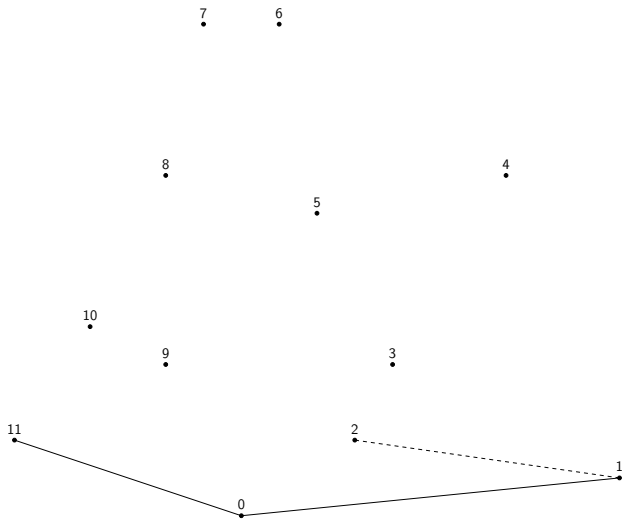
- ▶ Við byrjum á að velja vendipunkt, yfirleitt látinn vera punkturinn neðst til vinstri.
- ▶ Við látum hann fremst í safnið og röðum svo restin miðið við hornið sem þeir mynda við vendipunktinn.
- ▶ Við gefum okkur svo hlaða og látum aftasta, fremsta og næst fremsta punktin á hlaðann.
- ▶ Við göngum síðan í gegnum raðaða punkta safnið okkar og fyrir hvert stak fjarlægjum við ofan af hlaðanum á meðan efstu tvö stökin á hlaðanum og stakið sem við erum á í listanum mynda hægri beygju. Þegar þau mynda vinstri beygju bætum við stakinu úr safninu á hlaðann.
- ▶ Þegar við erum búin að fara í gegnum allt safnið er hlaðinn kúpti hjúpurinn.

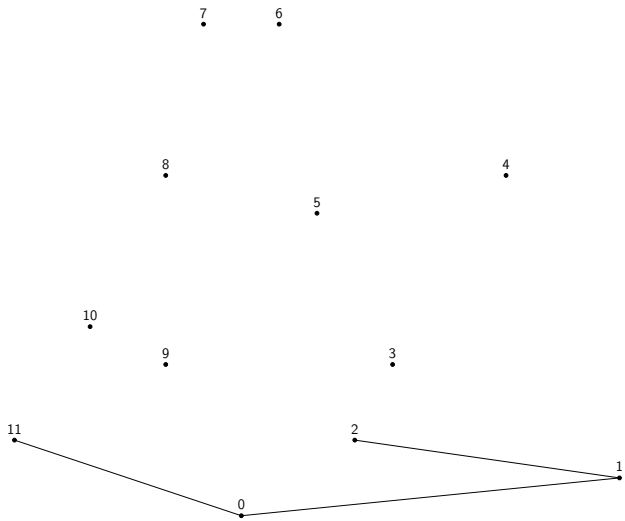


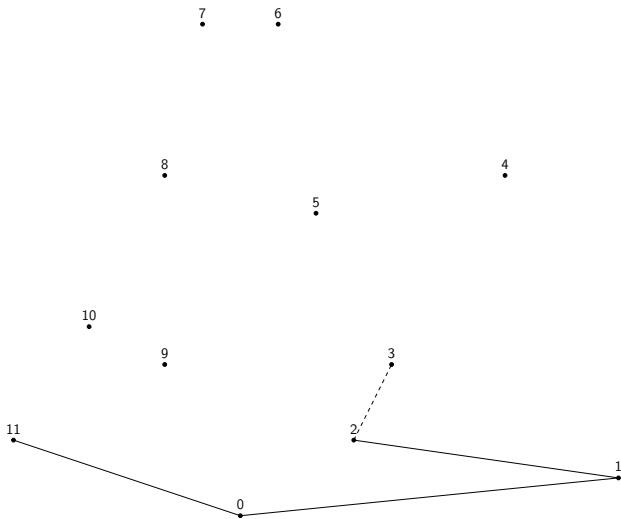


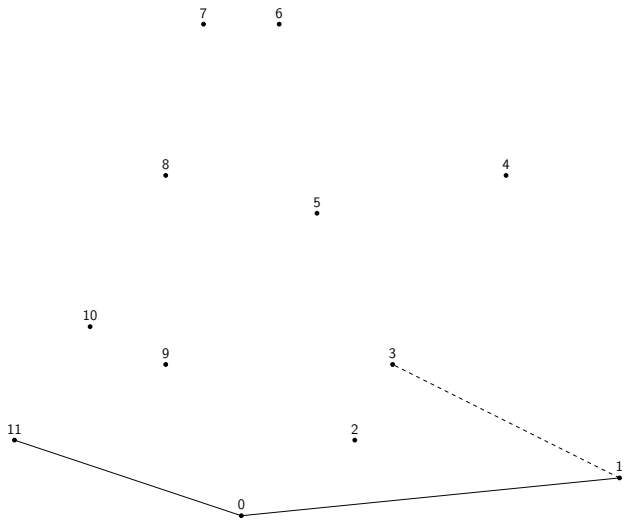


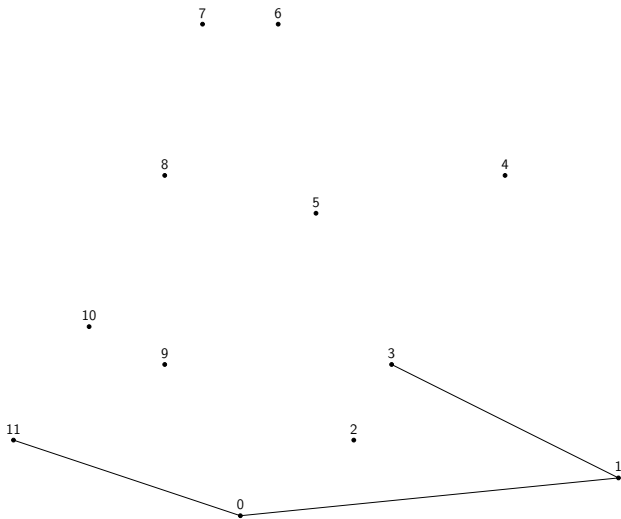


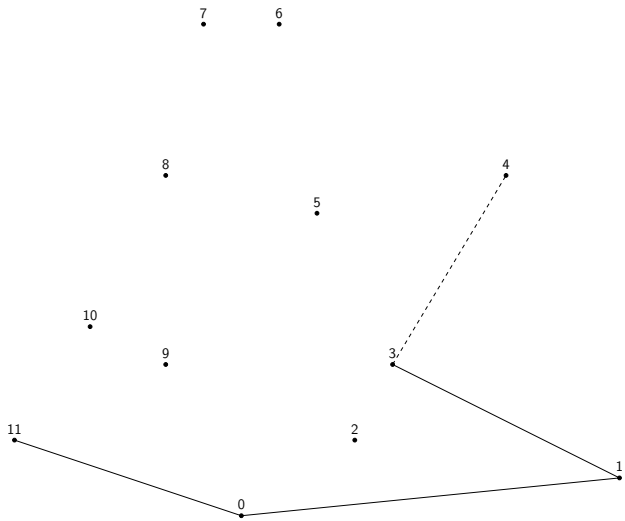


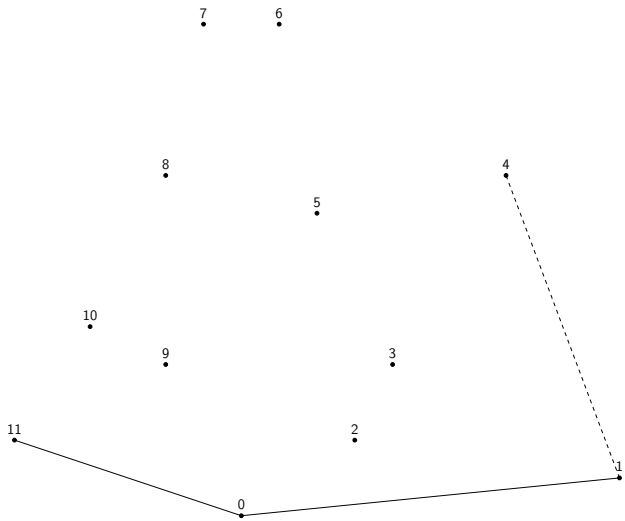


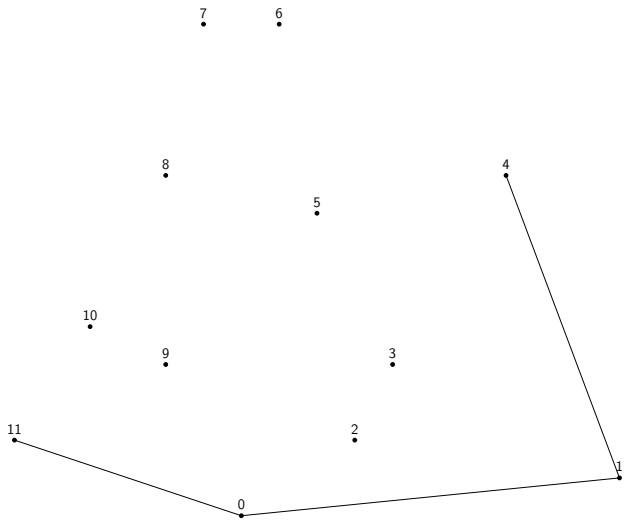


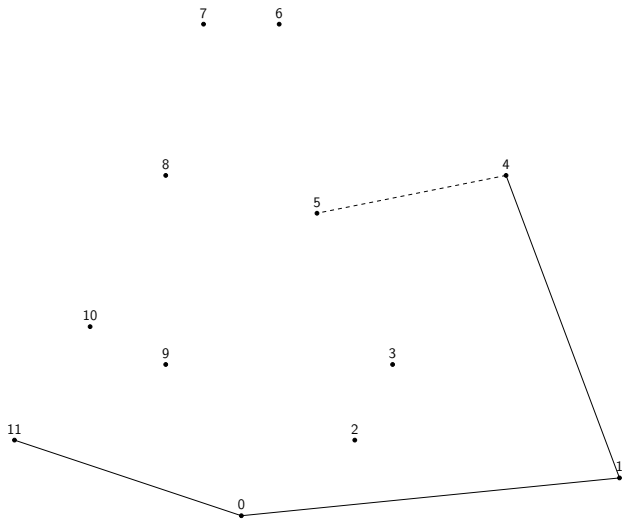


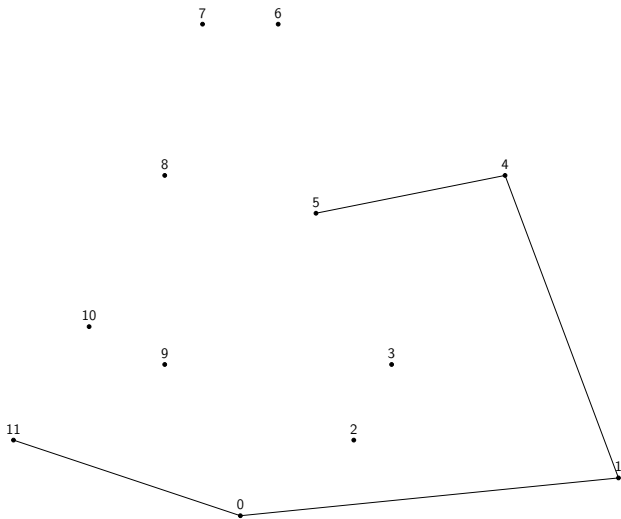


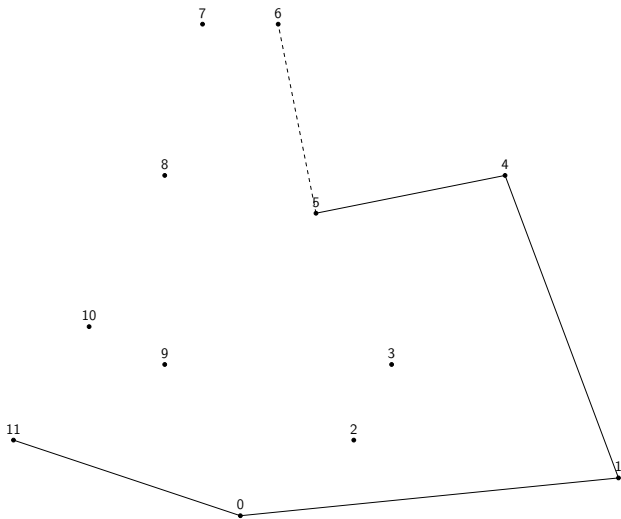


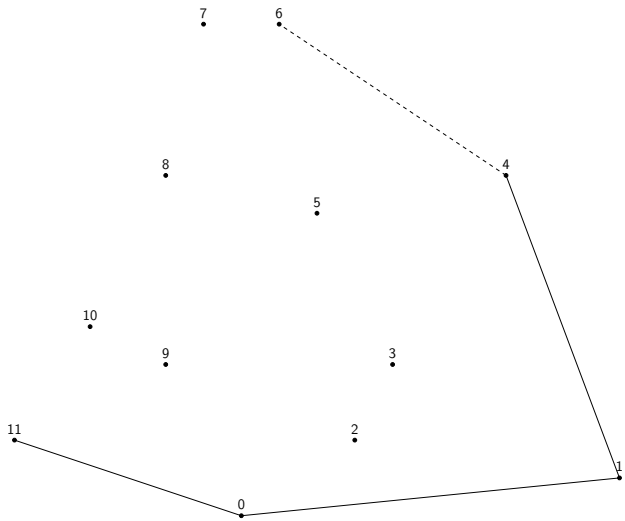


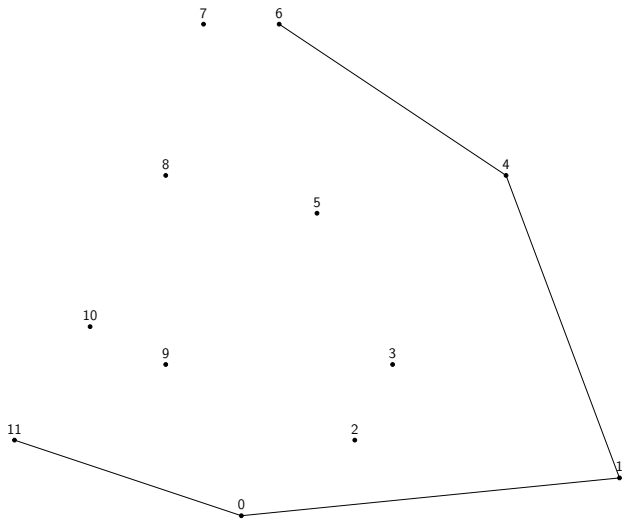


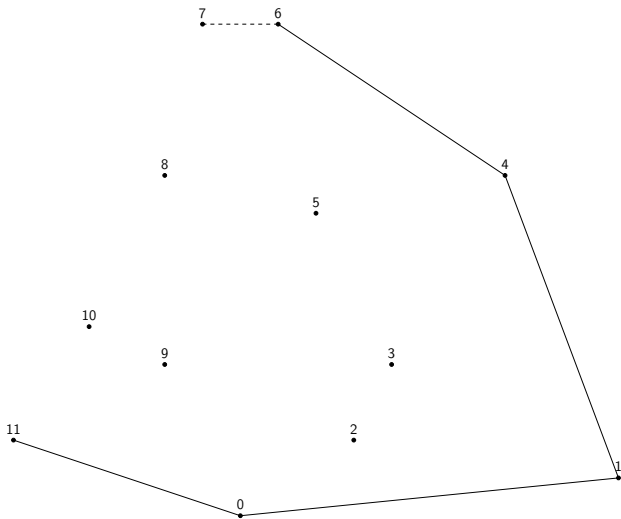


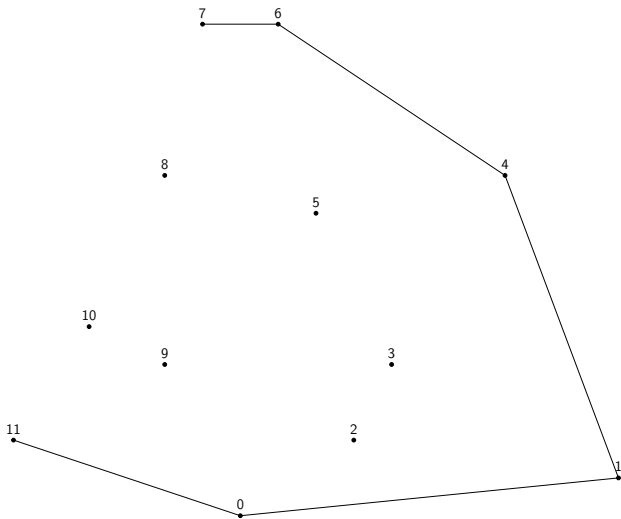


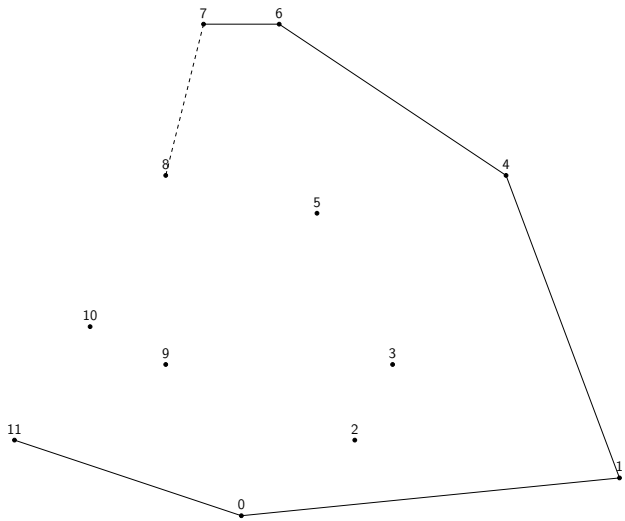


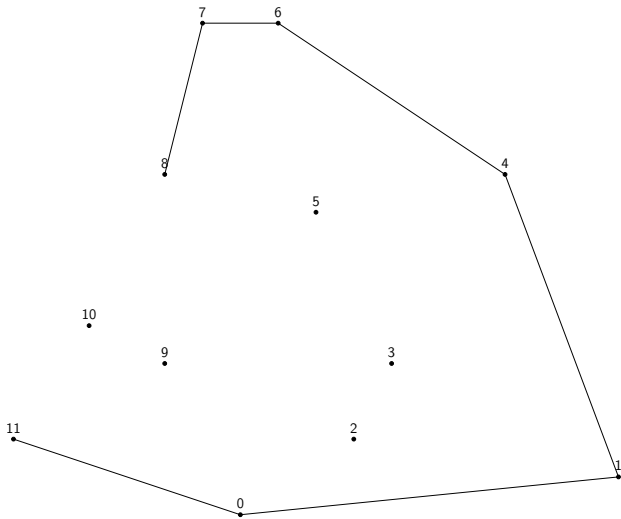


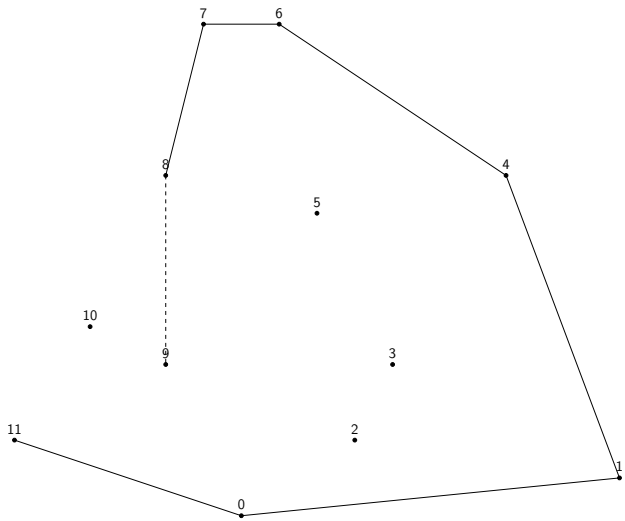


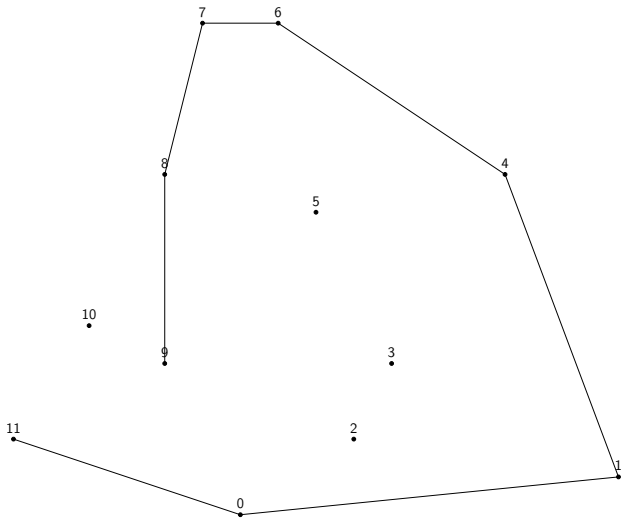


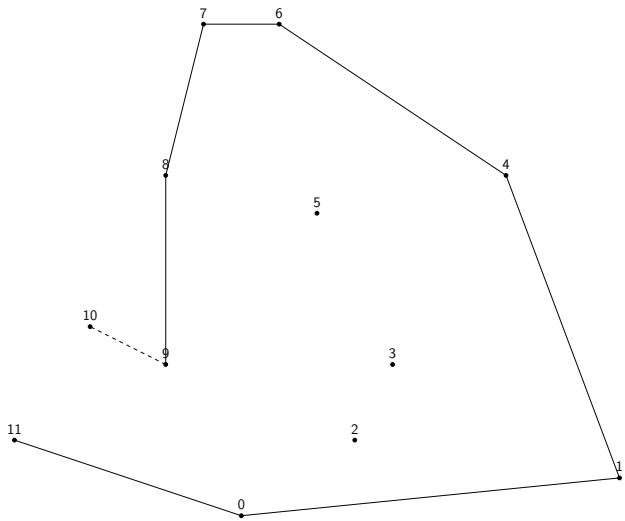


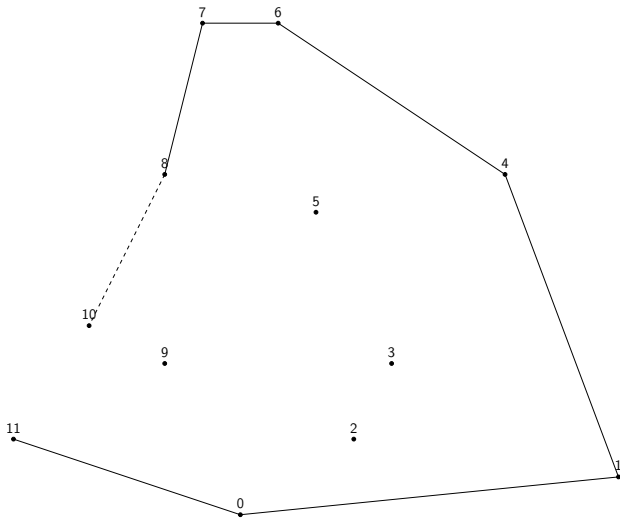


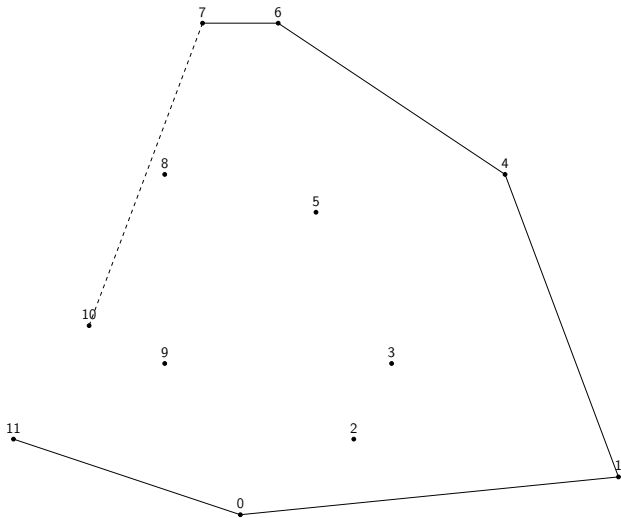


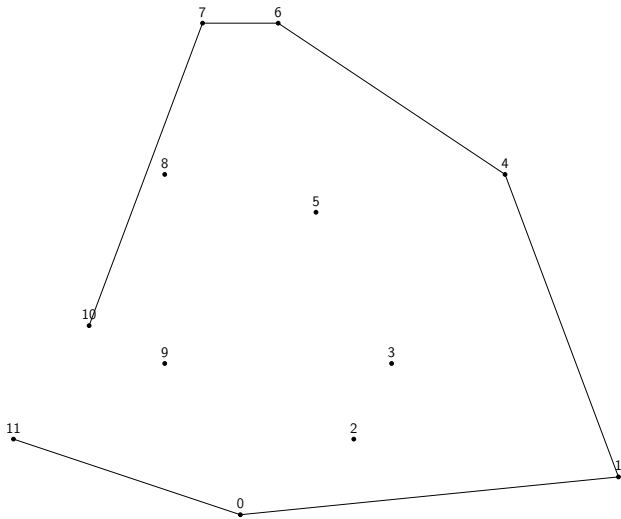


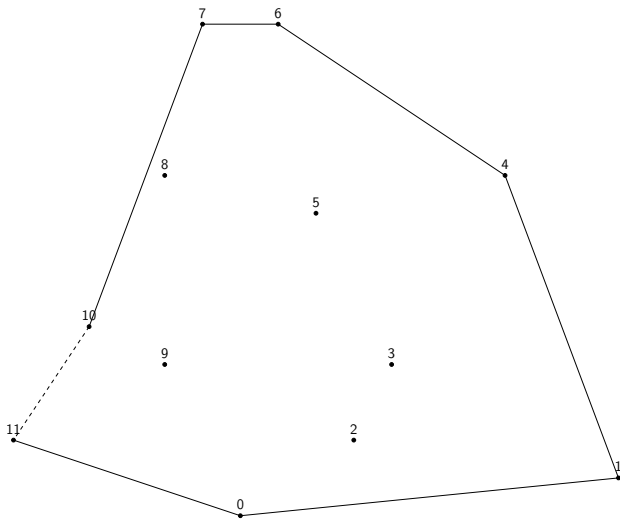


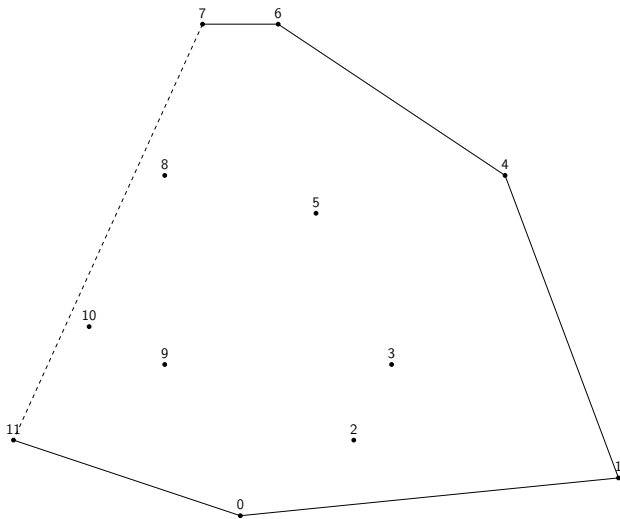


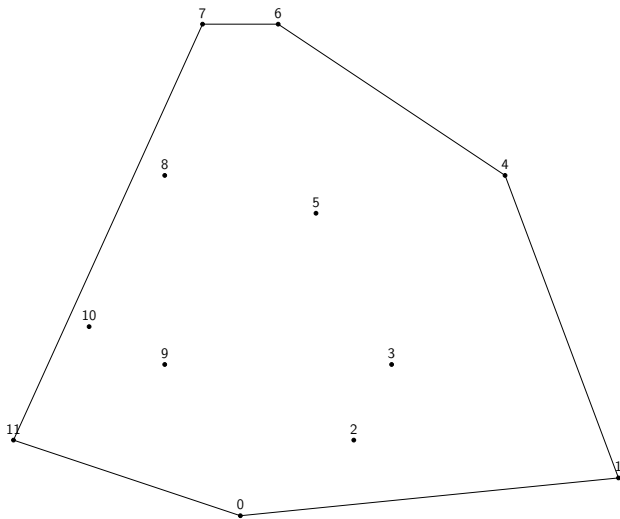












```

15 pt piv;
16 int cmp(const void* p1, const void* p2)
17 {
18     pt a = *(pt*)p1, b = *(pt*)p2;
19     if (fabs(carg(a - piv) - carg(b - piv)) > EPS)
20         return carg(a - piv) < carg(b - piv) ? -1 : 1;
21     if (fabs(cabs(a - piv) - cabs(b - piv)) < EPS) return 0;
22     return cabs(a - piv) < cabs(b - piv) ? -1 : 1;
23 }
24
25 int convex_hull(pt* p, pt* h, int n)
26 { // Eftir á er h kúpti hjúpur p. Skilar stærð h.
27     int i, j = 0, mn = 0;
28     for (i = 1; i < n; i++)
29         if (cimag(p[i] - p[mn]) < 0.0 || fabs(cimag(p[i] - p[mn])) < EPS
30             && creal(p[i] - p[mn]) < 0.0) mn = i;
31     pt t = p[mn]; p[mn] = p[0]; p[0] = t; piv = p[0];
32     qsort(p + 1, n - 1, sizeof *p, cmp);
33     for (i = 1; i < n && cabs(p[0] - p[i]) < EPS; i++);
34     if (i == n) h[j++] = p[0];
35     else if (i == n - 1) h[j++] = p[0], h[j++] = p[n - 1];
36     if (i >= n - 1) return j;
37     h[j++] = p[n - 1], h[j++] = p[0], h[j++] = p[i];
38     if (ccw(h[0], h[1], h[2]) == 0)
39         return j - (cabs(h[0] - h[1]) < EPS ? 2 : 1);
40     for (i++; i < n; i++)
41         (cabs(h[j - 1] - p[i]) > EPS && ccw(h[j - 2], h[j - 1], p[i]) == 1)
42             ? (h[j++] = p[i++]) : j--;
43     return j;
44 }

```

- ▶ Það er ljóst að í lok reikniritsins lýsir hlaðinn kúptum marghyrningi.
- ▶ Það er þó aðeins meira mál að sýna að þetta sé í raun kúpti hjúpur punktasafnsins.
- ▶ Við látum það ógert í þessum fyrirlestri.
- ▶ Ef punktasafnið inniheldur n punkta þá er reikniritið $\mathcal{O}(n \log n)$ út af því við þurfum að raða punktunum.

