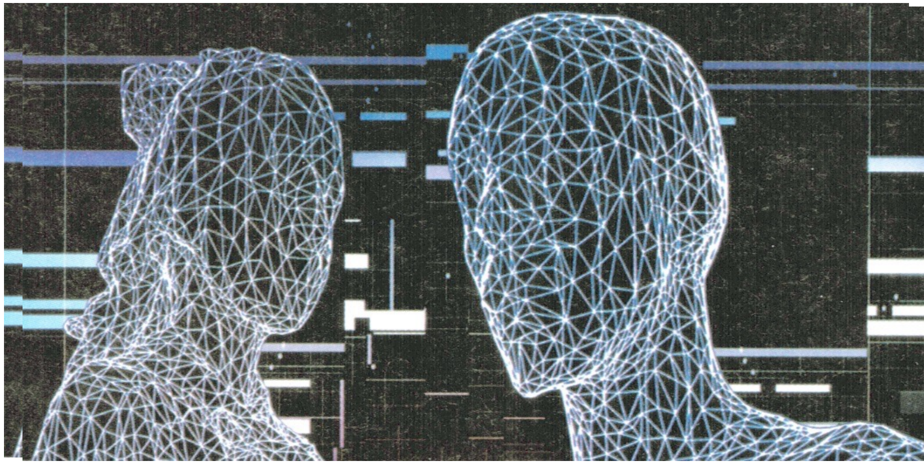
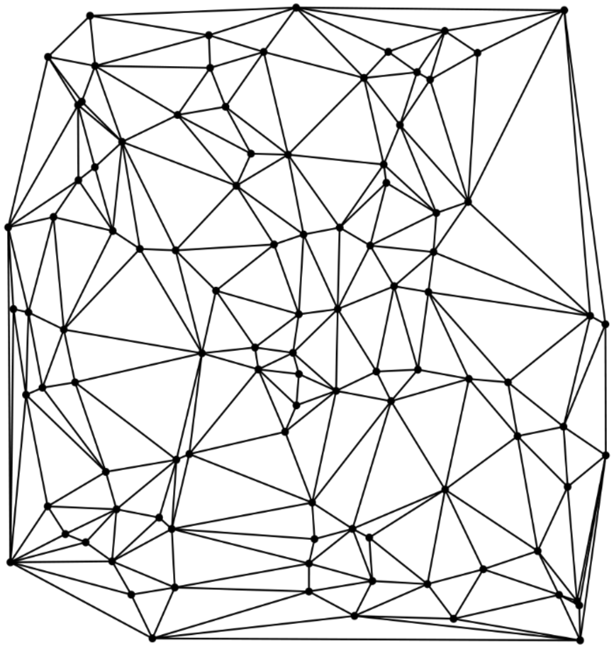
IN3030 Uke 13, v2021

Eric Jul
PT
Inst. for informatikk

Triangulering – å lage en flate ut fra noen målinger

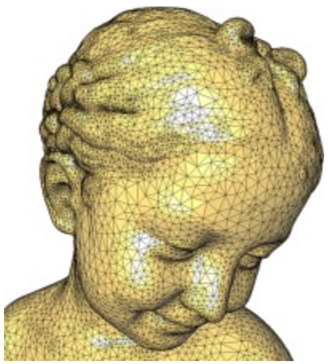
- Av og til vil vi representere noen målinger i 'naturen' og lage en kunstig, kontinuerlig flate:
  - Oljeleting – topp/bunn modell av oljeførende lag
  - Kart – fjell og daler, sjøkart
  - Grafiske figurer:
    - Personer, våpen, hus,..
- (x,y) er posisjonen, mens z er høyden
- Vi kan velge mellom :
  - Firkanter – det er vanskelige flater i en firkant (vridde)
  - Trekanter – best, definerer et rett plan
- Rette plan kan lettest glattes for å få jevne overganger til naboflater.





Hva bruker vi den konvekse innhyllinga til?

- Innhyllinga er en helt nødvendig første steg i flere-stegs algoritmer innen :
  - Spillgrafikk (modellerering av flater , mennesker, ansikter, hus, borger, terreng, .. osv) med lyssetting
  - Kartografi
    - Høydekart over landskap
    - Sjøkart
    - volumberegninger innen olje-prospektering.



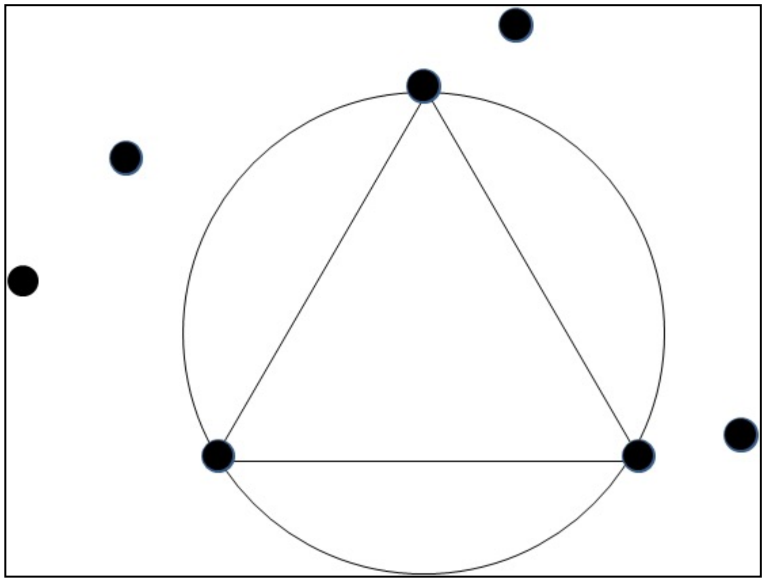
De etterfølgende figurer er laget i Geogebra . Anbefales sterkt (gratis) – last ned: http://geogebra.no/

Delaunay triangulering (1934)

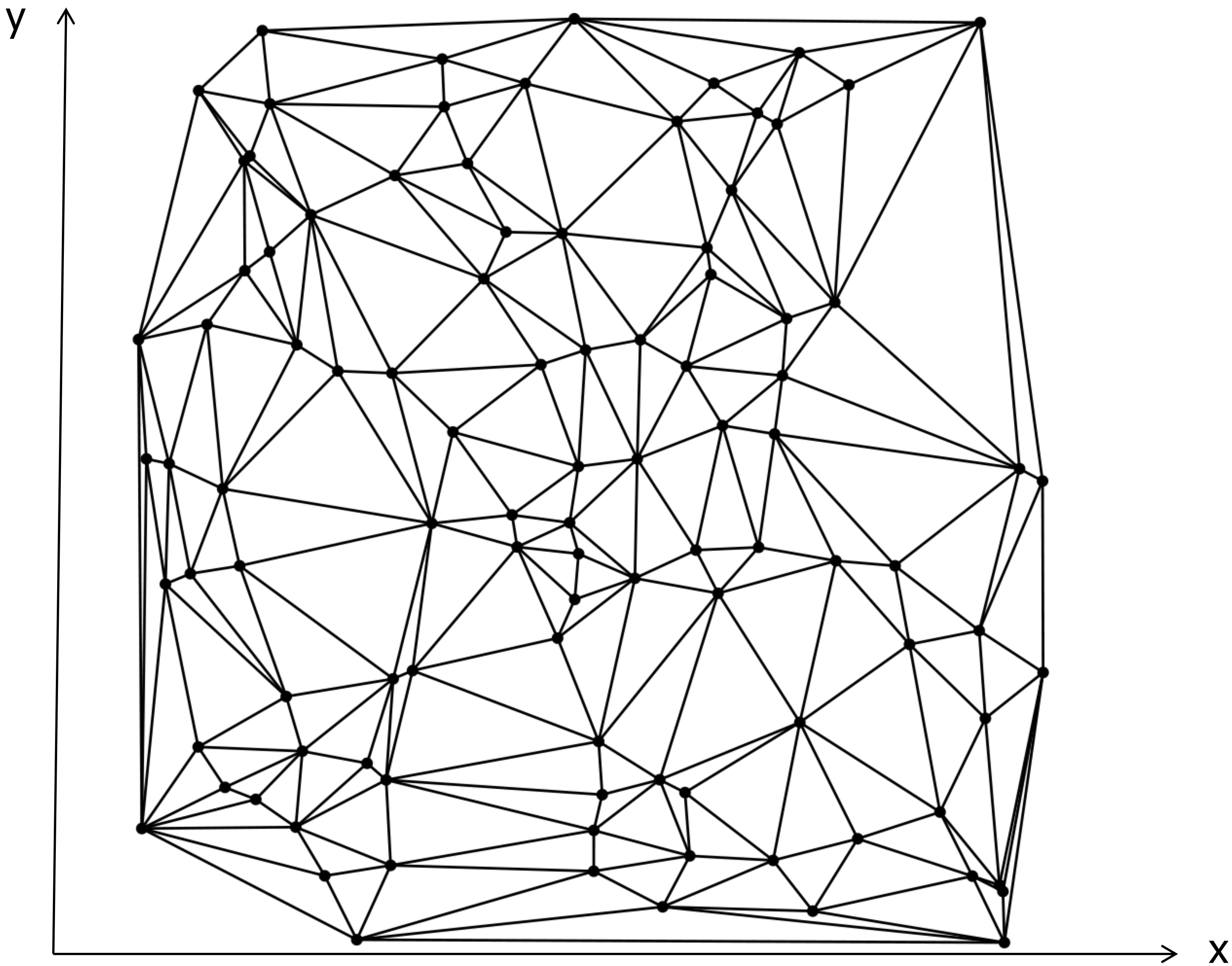
Boris Nikolaevich Delaunay 1890 – 1980, russisk fjellklatrer og matematiker
(etterkommer etter en fransk offiser som ble tatt til fange under Napoleons invasjon av Russland, 1812)

Vi har n punkter i planet
Forbind disse punktene med hverandre med et trekantnett slik at:

- Ingen linjer (trekantsider) krysser hverandre
- Man lager de 'beste' trekantene (maksimerer den minste vinkelen, dvs. færrest lange og tynne trekanter)
- **Def:** Den omskrevne sirkelen for tre de hjørnene i enhver trekant inneholder ingen av de (andre) punktene i sitt indre

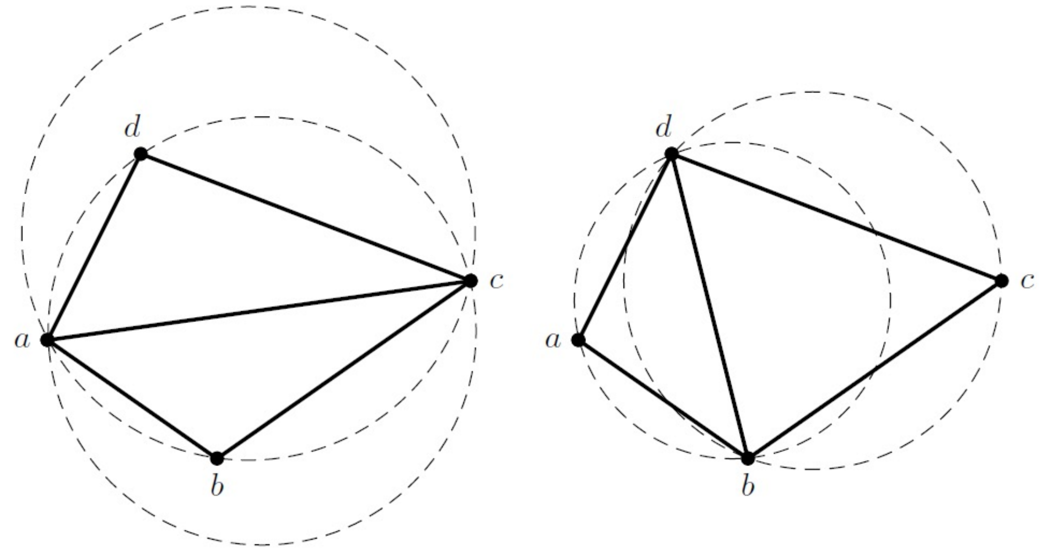


Delaunay triangulering av 100 tilfeldige punkter



Delaunay algoritmer; mange dårlige & få gode

- De aller første for å lage en DT (Delaunay Trekant) ABC:
  - Velg et punkt A, prøv alle mulige (n-1) av B_i, og igjen for hver av B_i –ene: alle mulige (n-2) valg av C_j . Test så om A B_i C_j tilfredstiller sirkel-kriteriet.
  - Å finne én DT tar da $O(n^2)$ tid og finne alle DT tar **$O(n^3)$** tid !
- I kurset INF 4130 undervises en flippingsalgoritme som i verste tilfellet er $O(n^2)$.



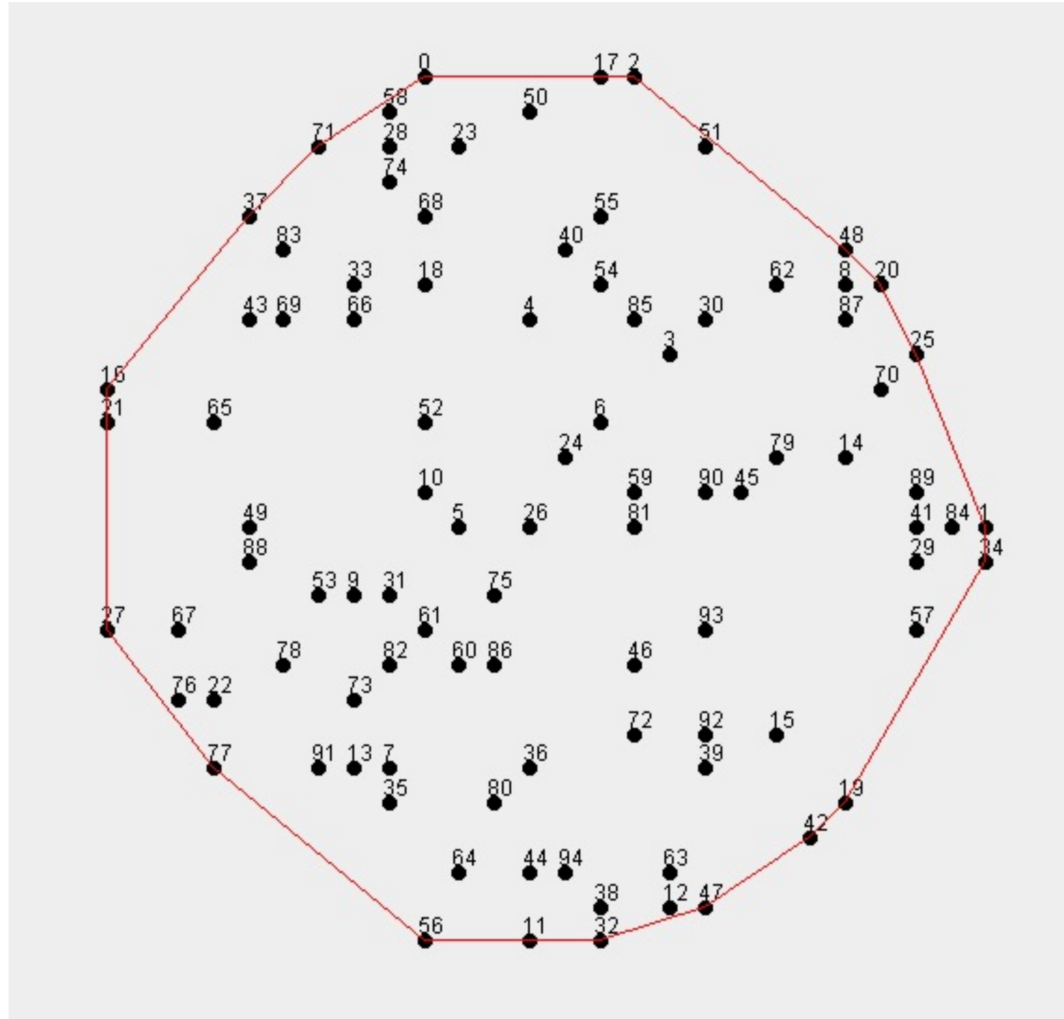
Delaunay – algoritmer her (i prinsippet O(n)):

- Det er (minst) to algoritmer som er O(n):
  - a) Konveks innhylling + sirkelutvidelser fra kjent linje AB i en DT
  - b) Konveks innhylling + Nærmeste nabo(er) + …..
- Konveks innhylling, er Oblig5
- Rask kode som løser a) og b) er ca. 2000 LOC (Lines Of Code) og greier a)+b) med minst 500 000 punkter per sekund.

Den konvekse innhyllinga til n punkter – Oblig 5

- Hva er det, definisjon
  - Hvordan ser den ut
- Hva brukes den til?
- Hvordan finner vi den?

1) Oblig 5, problemstilling



- Vi skal finne den konvekse innhyllinga til n punkter p i xy-planet.
- Her er 95 tilfeldige punkter og deres innhylling:
- en rekke med linjer fra punkt til punkt i mengden slik at alle andre punkter er på 'innsida' av denne mangekanten.
- Mangekanten er konveks, dvs alle indre vinkler er ≤ 180°
- Altså **ikke** linje 19-57 og 57-34.

. Alle punktene på en slik innhylling er med: Altså: linje 16-21 og 21-27, **ikke** linje direkte: 16-27.

Først en enkel geometrisk sats, I

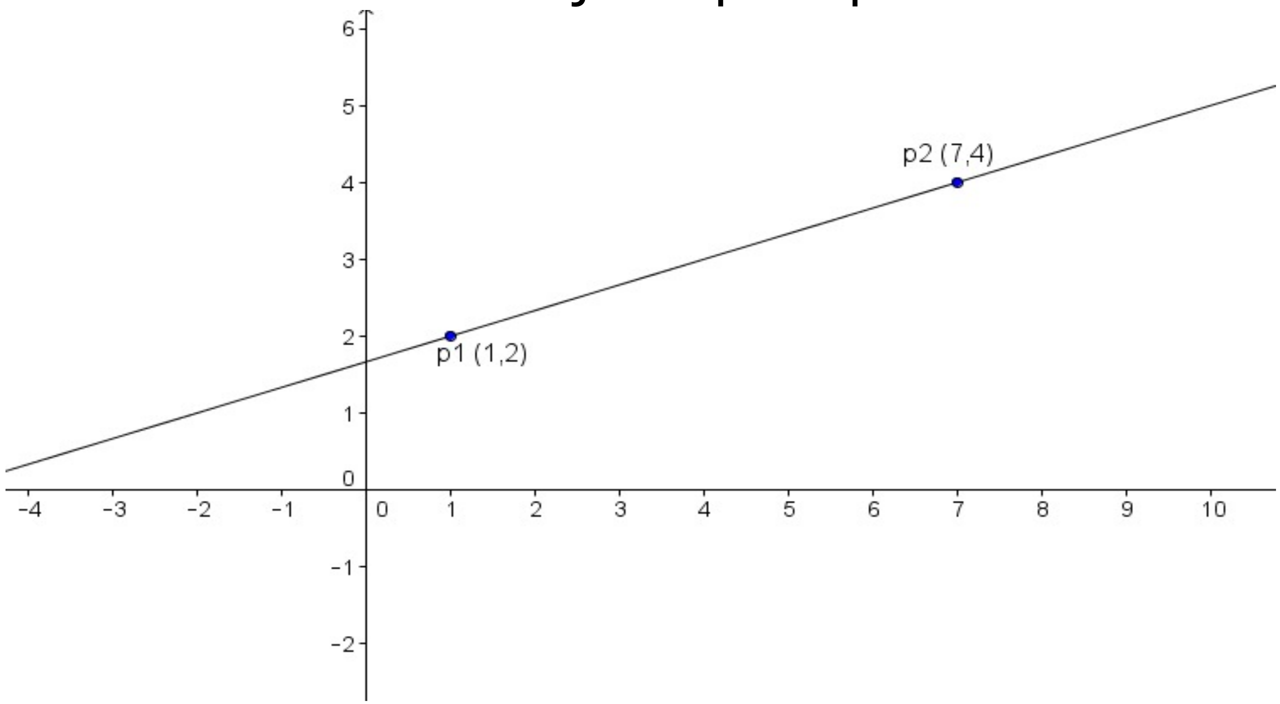
Ligningen for en linje

Enhver linje **fra** et punkt p1 (x1,y1) **til** p2(x2,y2) kan skrives på formen (trivielt forskjellig fra slik Geogebra gjør det):

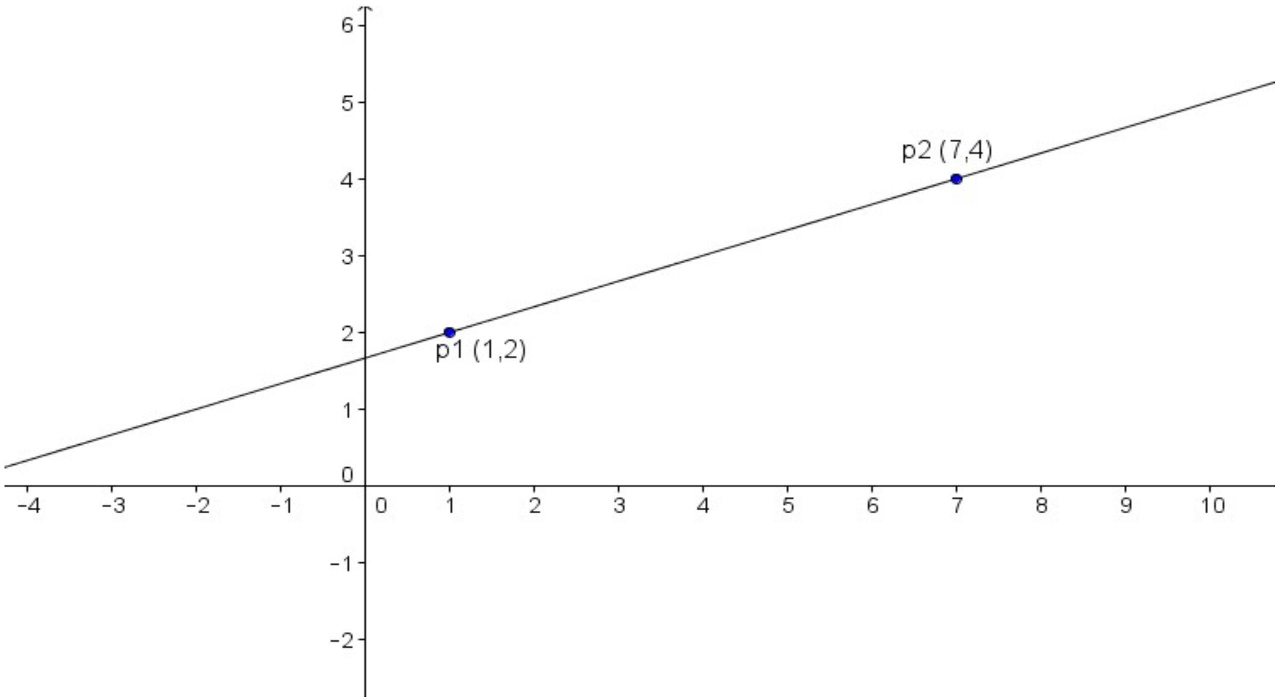
$$\boldsymbol{ax + by + c = 0}$$

Hvor: $a = y1 - y2,\ b = x2 - x1\ og\ \ c = y2 * x1 - y1 * x2$.

Merk at dette er en rettet linje *fra* p1 *til* p2.



Først en enkel geometrisk sats, II



*Figur2. En linje **fra** p1 (1,2) **til** p2 (7,4) har da linjeligningen:*

$$a = y1 - y2, \quad b = x2 - x1 \quad , \quad c = y2 * x1 - y1 * x2.$$

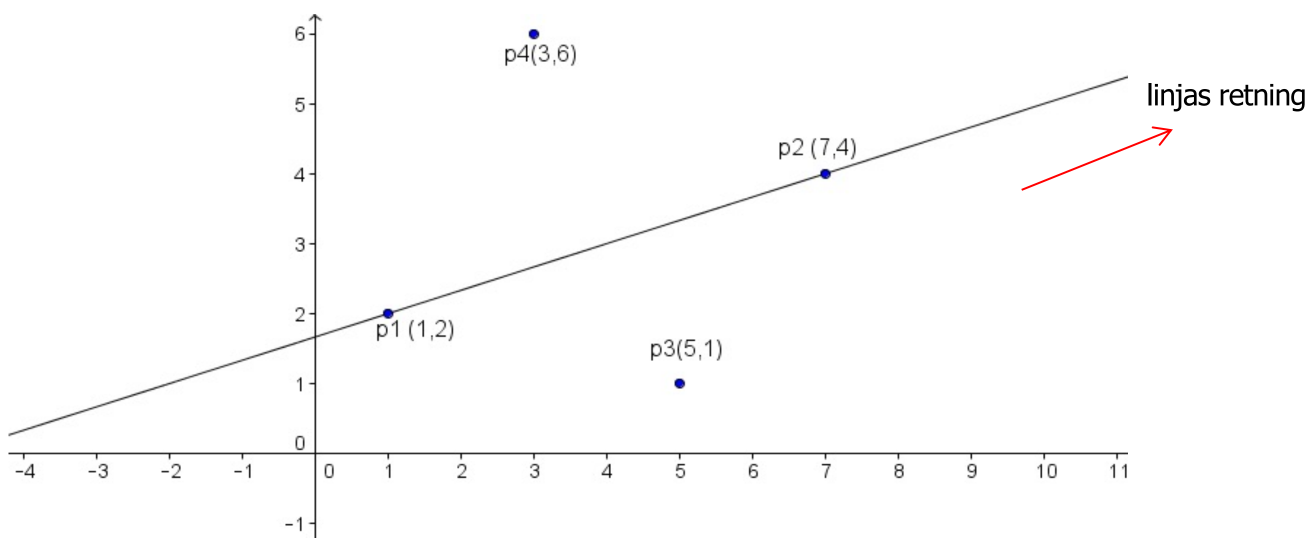
$$(2-4)x + (7-1)y + (4*1 - 2*7) = 0;\ dvs\!: \boldsymbol{-2x + 6y - 10 = 0}$$

Avstanden fra et punkt til en linje, I

- Setter vi ethvert punkt på p(px,py) linja, vil linjeligninga gi 0 som svar (per definisjon):

$$\boldsymbol{a * px + b * py + c = 0}$$

- Setter vi inn et punkt som **ikke** er på linja (p4 eller p3) vil vi få et tall som er :
  - negativt (<0) hvis punktet er til høyre for linja, sett i linjas retning: p1 til p2
  - positivt (>0) hvis punktet er til venstre for linja, sett i linjas retning: p1 til p2



Avstanden fra et punkt til en linje II

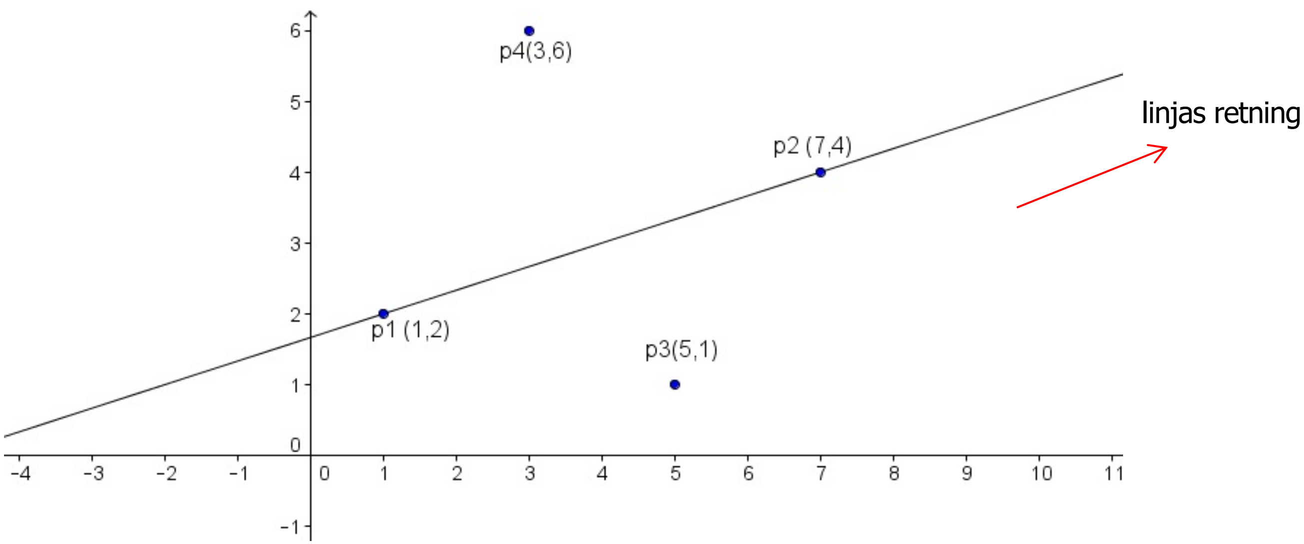
- Avstanden fra et punkt (x,y)til en linje (vinkelrett ned på linja) er :

$$d = \frac{ax + by + c}{\sqrt{a^2 + b^2}}$$

- Jo lenger fra linja punktene et desto større negative og positive tall blir det.

Setter inn **p3** (5,1) i linja p1-p2: -2x+6y-10 = 0, får vi

d= : $\frac{-2*5+6*1-10}{\sqrt{40}} = \frac{-14}{6,32} = -2,21..$



En linje deler da planet i to: Punktene til høyre og til venstre for linja (sett fra rettet linje fra p1 til p2)

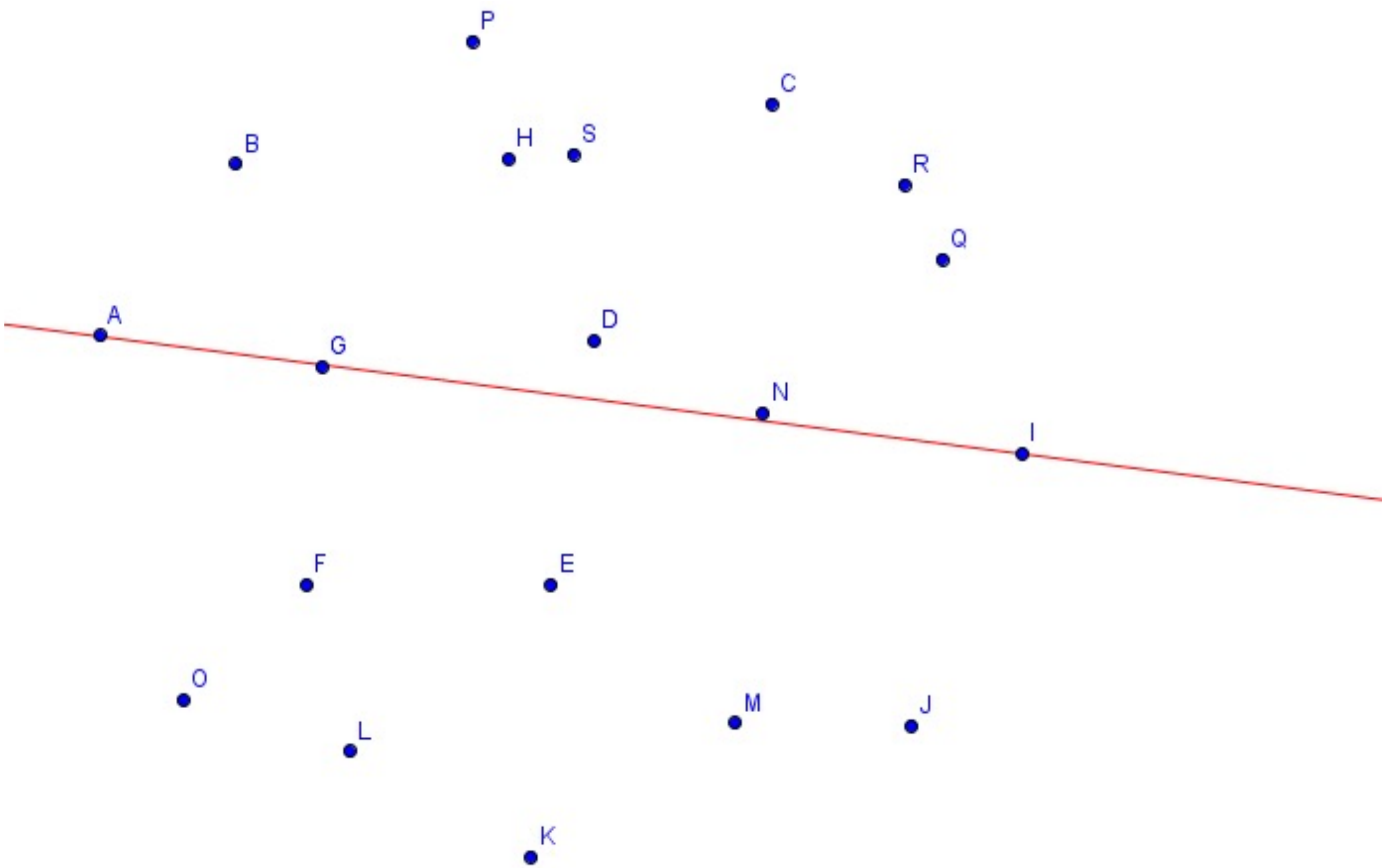
- Vi er nå interessert i de punktene som ligger lengst fra én gitt linje (til høyre for den) $\boldsymbol{ax + by + c = 0}$ i en stor punktmengde.

- Kan da avstandsformelen forenkles – gjøres raskere når vi skal måle mange ulike punkters avstand *d* til en og samme linje: $ax + by + c = 0$

$$d = \frac{ax + by + c}{\sqrt{a^2 + b^2}}$$

To observasjoner:

- Punktene med minst og størst x-verdi (A og I) ligger på den konvekse innhyllinga
- De punktetene som ligger lengst fra (positivt og negativt) enhver linje p1-p2, er to punkter på den konvekse innhyllinga.(P og K)

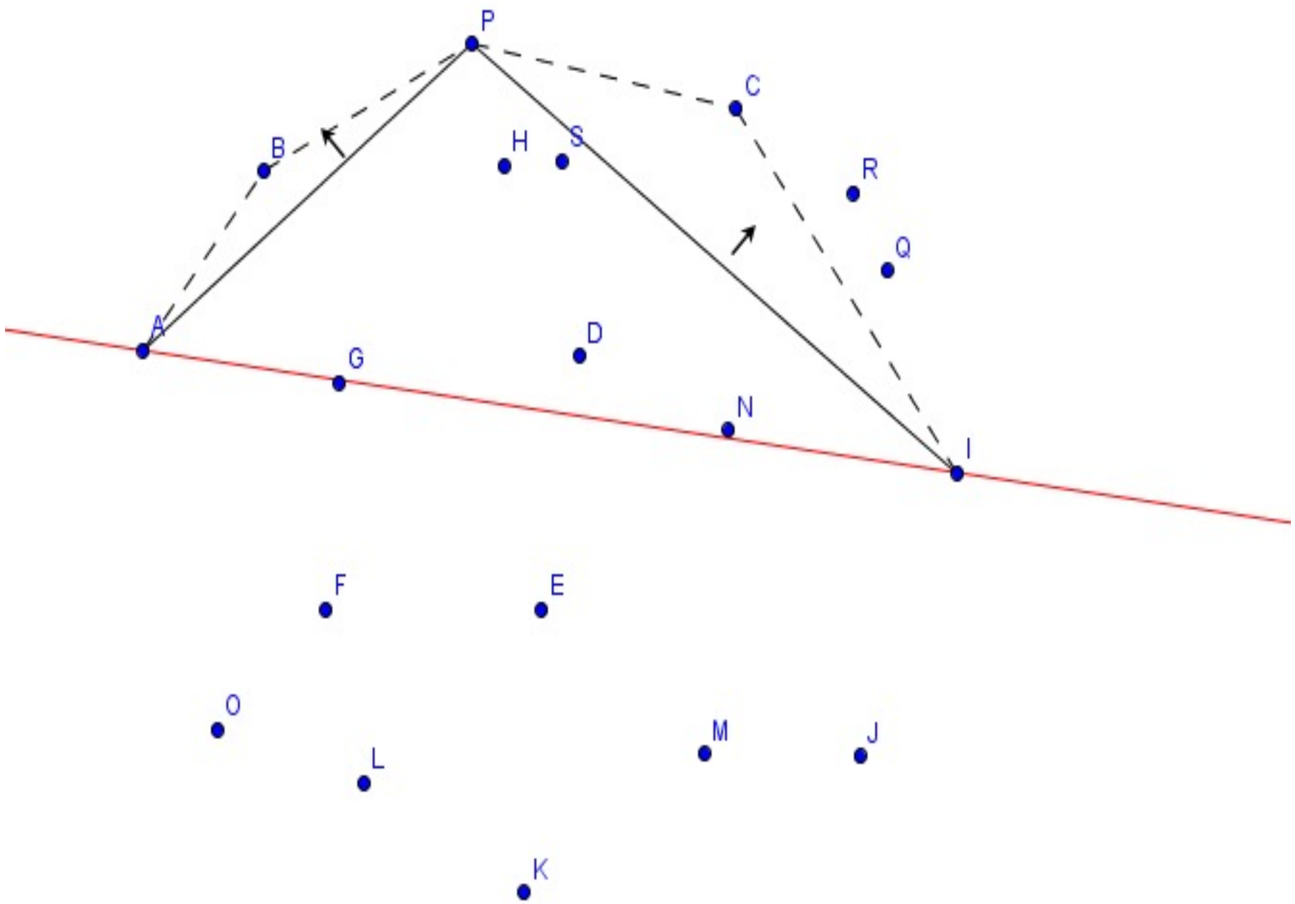


Vi skal etter starten av algoritmen bare se på det punktet som ligger i mest negativ avstand fra linja (dvs mest til-høyre for linja)

Algoritmen for å finne den konvekse innhyllinga sekvensielt

1. Trekk linja mellom de to punktene vi vet er på innhyllinga fra maxx -minx (I - A).
2. Finn punktet med størst negativ (kan være 0) avstand fra linja (i fig 4 er det P). Flere punkter samme avstand, velg vi bare ett av dem.
3. Trekk linjene fra p1 og p2 til dette nye punktet p3 på innhyllinga (neste lysark: I-P og P-A).
4. Fortsett rekursivt fra de to nye linjene og for hver av disse finn nytt punkt på innhyllinga i størst negativ avstand (≤ 0).
5. Gjenta pkt. 3 og 4 til det ikke er flere punkter på utsida av disse linjene.
6. Gjenta steg 2-5 for linja minx-maxx (A-I) og finn alle punkter på innhyllinga under denne.

Rekursiv løsning: Finn først P (mest neg. ‘avstand’ fra I-A) Trekk så I-P og finn C, Trekk så I-C , og finn R. trekk så I-R og finn Q. Finner så intet ‘over’ R-C eller C-P. Trekker P-A og finner så B over. Ferdig.



Problemer dere vil møte i den rekursive, sekvensielle løsningen I

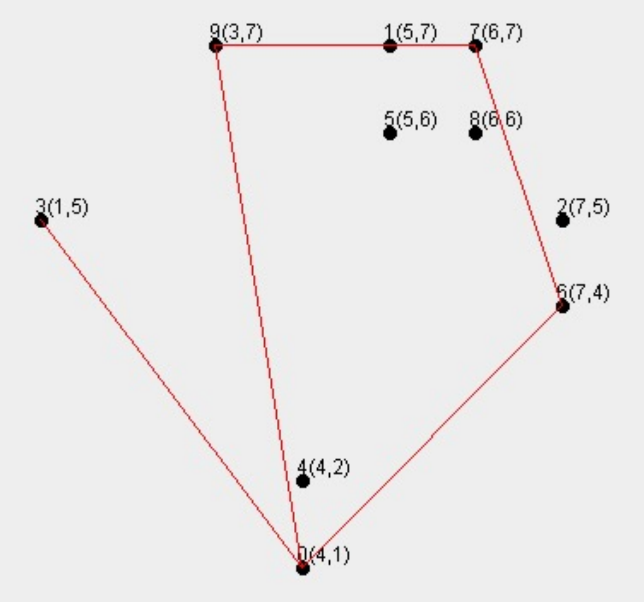
- **Hvordan representere et punkt p_i** ?
  - Med indeksen 'i' (ikke med koordinatene x og y) ?
- **Debugging** (alle gjør feil først) av et grafisk problem vil vi ha tegnet ut punktene og vårt beste forsøk på konvekse innhylling.

Klassen TegnUt (hvis n < 250)
Brukes slik fra main-tråden:

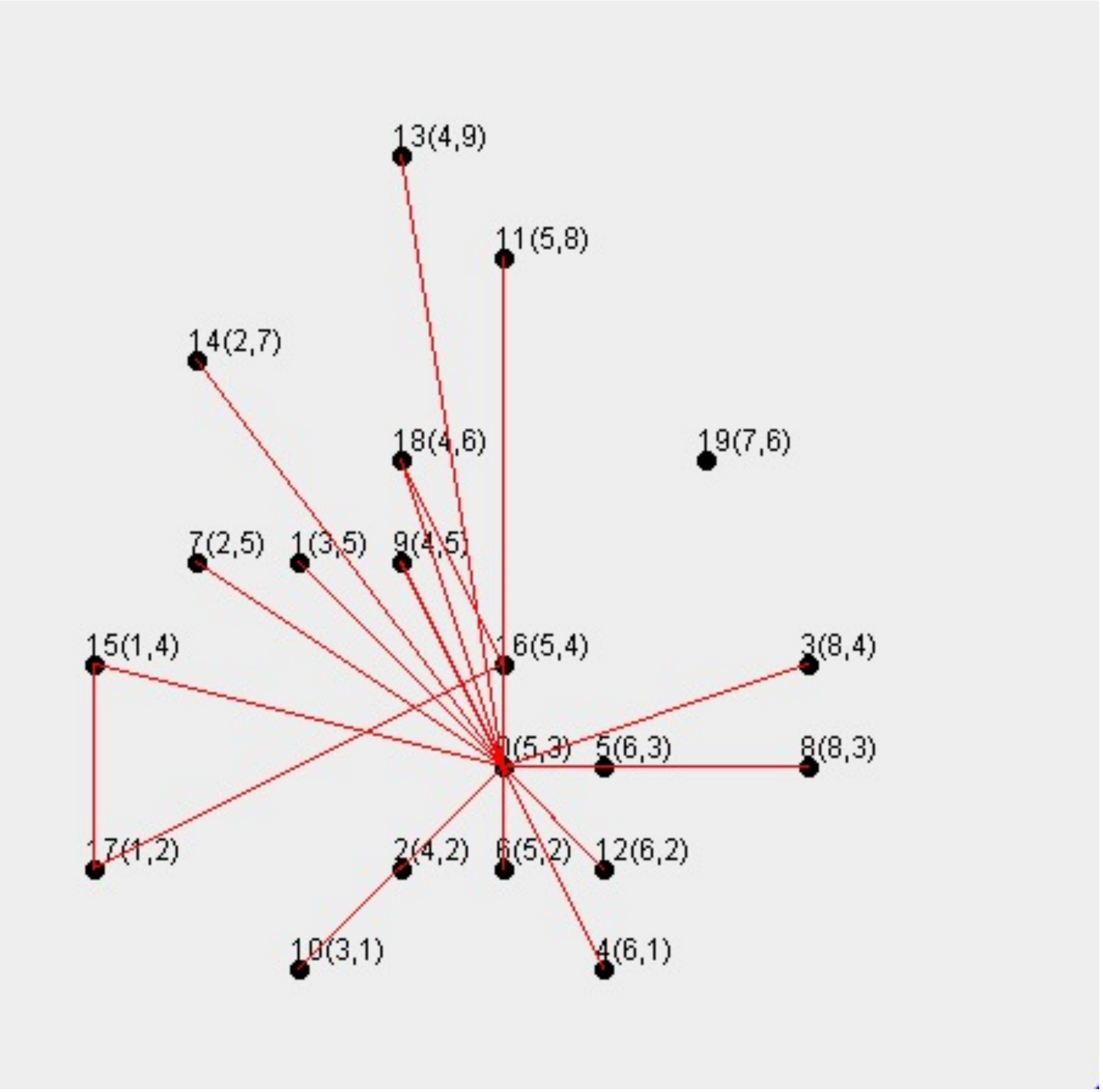
```
TegnUt tu = new TegnUt (this, koHyll);
```

a) TegnUt tegner ut punktene og innhyllinga i en IntList koHyll. Skrives trivielt om av deg hvis du bruker ArrayList. 'this' er en peker til main-objektet.
b) TegnUt antar at main-objektet er et objekt av klassen Oblig4.
c) Ikke nødvendigvis 'proff' kode i klassen TegnUt

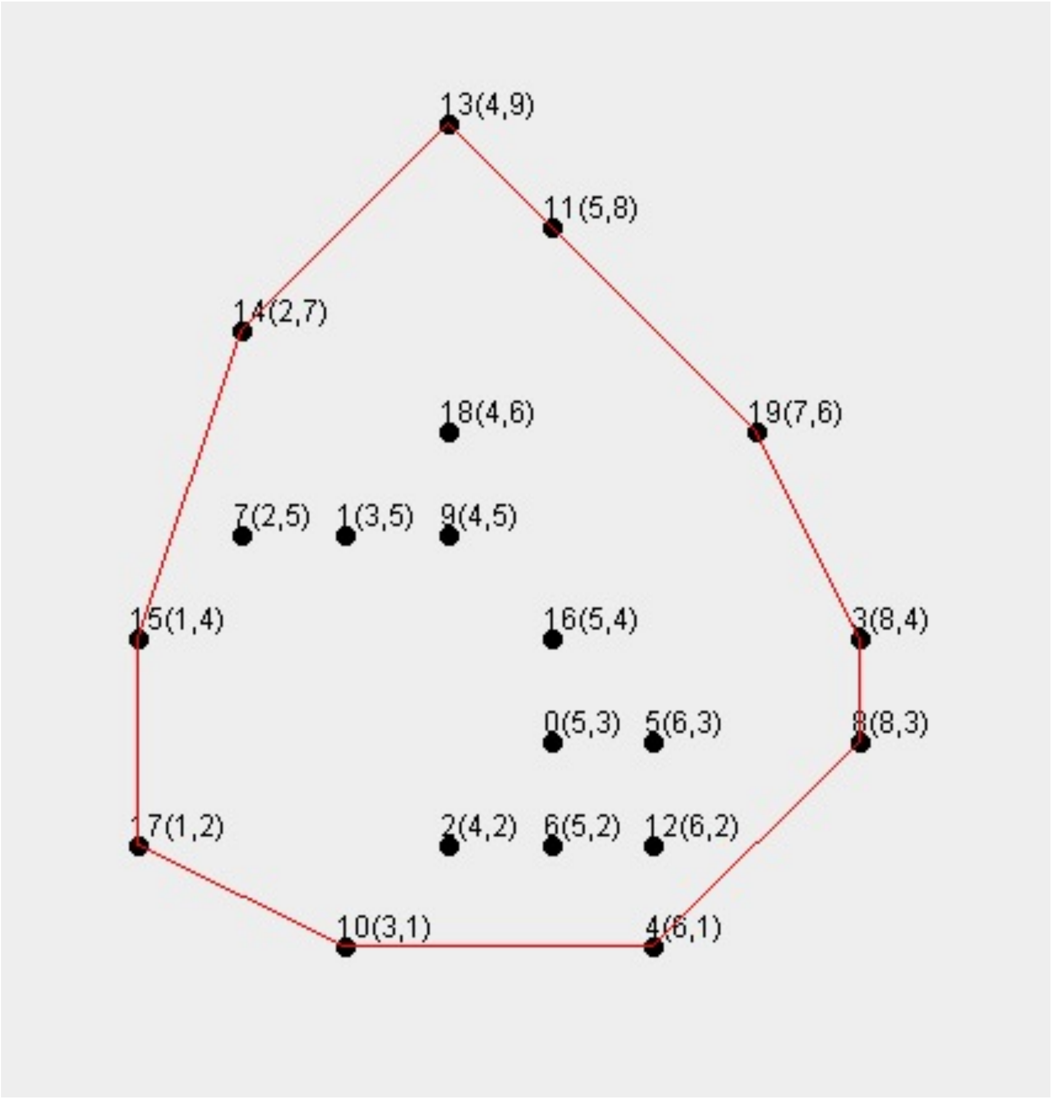
«Litt» feil:



Mye feil:


13(4,9)
11(5,8)
14(2,7)
18(4,6)
19(7,6)
7(2,5)
1(3,5)
9(4,5)
15(1,4)
16(5,4)
3(8,4)
0(5,3)
5(6,3)
8(8,3)
17(1,2)
2(4,2)
6(5,2)
12(6,2)
10(3,1)
4(6,1)

Riktig



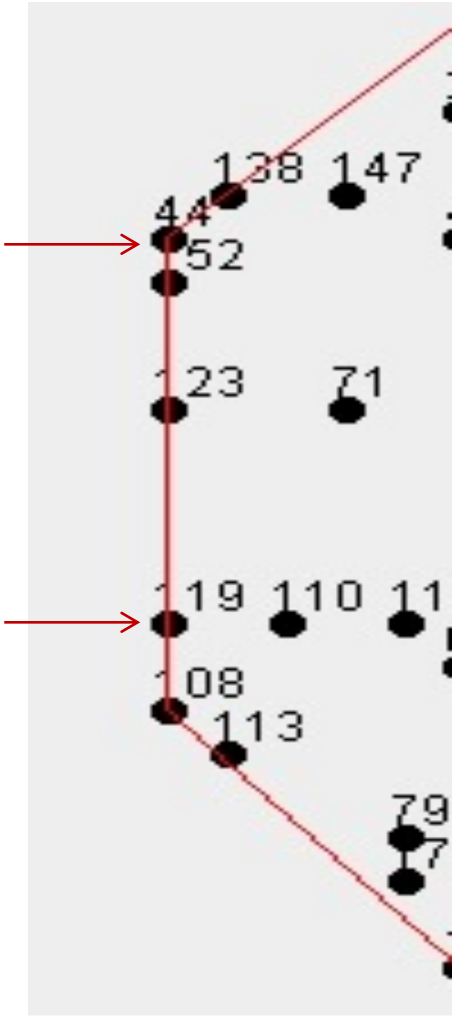
13(4,9)
11(5,8)
14(2,7)
18(4,6)
19(7,6)
7(2,5)
1(3,5)
9(4,5)
15(1,4)
16(5,4)
3(8,4)
0(5,3)
5(6,3)
8(8,3)
17(1,2)
2(4,2)
6(5,2)
12(6,2)
10(3,1)
4(6,1)

Problemer dere vil møte i den rekursive, sekvensielle løsningen II

- **Finne punktene på den konvekse innhyllinga i riktig rekkefølge**?
  - Tips: Du bruker to metoder for det sekvensielle tilfellet:
    - sekvMetode() som finner minx, maxx og starter rekursjonen. Starter rekursjonen med to kall på den rekursive metoden, først på maxx-minx, så minx-maxx:
    - sekvRek (int p1, int p2, int p3, IntList m)
      som, inneholder alle punktene som ligger på eller under linja p1-p2, og p3 er allerede funnet som det punktet med størst negativ avstand fra p1-p2.
      IntList m er en mengde punkter som liigger over (til høyre for) linje p1-p2
  - Du kan la sekvRek legge inn ett punkt: p3 i innhyllinga-lista, men hvor i koden er det ?
  - Når legges minx inn i innhyllingslista ?

Problemer dere vil møte i den rekursive, sekvensielle løsningen III

- **Få med alle punktene på innhyllinga hvor flere/mange ligger på samme linje (i avstand = 0), og få dem i riktig rekkefølge.**
- Tips:
  - Husk at når du finner at største negative avstand er = 0 må du ikke inkluderer p1 eller p2 som mulig nytt punkt (de er allerede funnet)
  - Si at du har funnet p1=44 og p2=119. Du bør da bare være interessert i å finne de punktene som ligger mellom p1 og p2 på linja (52 og 123), og da må du teste om nytt punkt p3 har både y og x-koordinater *mellom* tilsvarende koordinater for p1 og p2.
  - Da finner du ett av punktene (si: 123) med kall på sekRek over linja p1-p2 (44-119). Gjenta rekursivt (over 44-123) og 123-119) til det ikke lenger er noen punkter mellom nye p1 og p2.
  - Punktene videre nedetter linja (f.eks. 119-108) finnes av rekursjon tilsvarende som for 44 og 119.



Alternativ måte å få punktene i riktig rekkefølge

- Anta at vi har alle punktene på DKI i en uordnet IntList.
- Lag et fiktivt senterpunkt mp i mengden- for eksempel med koordinater midt mellom maxx og minx.
- Vi lager et tenkt aksekors med origo i dette punktet, og det deler de usorterte punktene på den konvekse innhyllinga i i fire kvadranter
- Første kvadrant er de hvor både x og y-verdiene >= midtpunktet.
- Regn ut linjene fra midtpunktet til hver av disse punktene (x2,y2)og sorter punktene stigende etter verdien av: –a/b = $(y2 - ymp)/(\ x2 - xmp)$ for de ulike linjeligningene, og dette er den riktige rekkefølgen.

  (Hvis $x2 - xmp$ ==0 , så er dette siste punktet)
- Tilsvarende for de andre kvadrantene

Hvordan parallellisere Oblig5 – den konvekse innhyllinga

- Presented at NEXT lecture on April 22nd, 2021

Hvorfor lage en egen IntList istedenfor ArrayList <Integer>

- Fordi det er raskere og tar mindre plass
- Også fordi vi kan legge inn problemspesifike metoder hvis vi trenger det
- Hva er forskjellen på en Integer og en int.
- Hvordan lage IntList
- (ha muligheter til å lage en array av lister,
  OK med `IntList[] a = new IntList[ant];`
  Ikke mulig med:
  `ArrayList <Integer > []b = new ArrayList<Integer >[ant];`

Forskjeller på Integer og int er bl.a størrelsen

- Integer er et objekt som brukes å holde et heltall.
  - **Hvert** Integer tar da 12 byte (object header) + 4 byte (int)
- int er en basaltype som tar 4 byte
  - En int [] array har **ett** array objekt for hver rad – dvs. ekstra 12 byte per rad + ett array-objekt for hver dimensjon (2,3,..)
  - se: http://stackoverflow.com/questions og http://www.javamex.com/tutorials/memory/object_memory_usage.shtml
  - a normal object requires **8 bytes** of "housekeeping" space;
  - **arrays require 12 bytes** (the same as a normal object, plus 4 bytes for the array length). If the number of bytes required by an object for its header and fields is not a multiple 8, then you **round up to the next multiple of 8**.
-
```
+-----------+--------------+---------------- +----------+
| mark word | class pointer | array size (opt) | padding  |
+-----------+--------------+-----------------+----------+
```
- Har vi 1000 Integer i en array er det: 1000x 16 (Integer objekter) + 16 (array overhead) + 8x1000 (pekere) = **24 016 byte**
- Har vi int [] a = new int[1000] er det 8+16 + 4*1000 = **4 024 byte**

```
class IntList{  // en litt for kort IntList, noen metoder mangler kanskje
            int [] data;
            int len =0;
            IntList( int len) {
                  data = new int [Math.max(1,len)];
            }
                IntList() {
                   data = new int [32];
               }
            void add( int elem) {
                   if (len == data.length) {
                         int [] b = new int [data.length*2];
                         //for (int i = 0; i < data.length; i++) b[i] = data[i];
                         System.arraycopy(data,0, b,0,data.length);
                         data =b;
                    }
                   data[len++] = elem;
            } // end add

            void clear(){
                  len =0;
            } // end clear

            int get ( int pos){
                  // error antar at svaret brukes til array-indeks
                  if (pos > len-1 ) return -1; else return data [pos];
            } //end get

            int size() {
                  return len;
            } //end size
} // end class IntList
```

Fra testprogrammet

```
long ILSek (int n) {
        IntList itlist = new IntList(n);
         long  j = 1;

          for (int i =0; i<n; i++)       itlist.add(i);
          for (int i =0; i<n; i++) j +=itlist.get(i);
         return j;
 }// end ILSek

 long ALSek (int n) {
        ArrayList <Integer> alist = new ArrayList<Integer>(n);
        long j = 1;

        for (int i =0; i<n; i++)      alist.add(i);
        for (int i =0; i<n; i++) j +=alist.get(i);
         return j;
  }// ens ALSek

………………………………..

long t = System.nanoTime();  // start tidtaking IntList
 sum += ILSek(n);
t = (System.nanoTime()-t);
..
t = System.nanoTime();        // start tidtaking ArrayList
 sum += ALSek(n);
t = (System.nanoTime()-t);
```

Tidsforbruk int og Integer

- Integer
  - Når vi skal lagre et heltall i en Integer, må den først pakkes-inn (boxing) i et objekt vi lager + en peker til den i arrayen av Integer-pekere
  - Når vi skal lese en heltallsverdi fra en Integer, må vi først følge en peker for å lokalisere Integer-objektet, og så lese ut int-verdien (unboxing)
- int-array
  - Lesing og skriving i en int [] er 'mye' raskere – tilsvarende som å finne og skrive en peker i Integer-arrayen.

Test på lage (add) n stk Integer i ArrayList og i IntList og lese dem (get)

Test av sekvensiell IntList mot (sekvensiell?) ArrayList for å oppbevare heltall
med 4 kjerner , og 8 tråder, Median av:3 iterasjoner

n	IntList(ms)	ArrayList(ms)	Speedup
100000000	797.44	30162.61	37.82
10000000	28.52	1269.97	44.53
1000000	2.76	7.77	2.81
100000	0.23	0.63	2.74
10000	0.02	0.07	2.77
1000	0.00	0.01	3.33
100	0.00	0.00	2.99
10	0.00	0.00	?

n	IntList(ms)	ArrayList(ms)	Speedup
500000000	6194.84	211277.47	34.11
50000000	146.46	2054.59	14.03
5000000	15.06	41.22	2.74
500000	1.13	3.44	3.05
50000	0.10	0.36	3.43
5000	0.02	0.04	1.57
500	0.00	0.00	2.50
50	0.00	0.00	2.00