

# Inlämningsuppgift 1 - 1DV005

Rasmus Svensson\*, Daniel Holmkvist<sup>†</sup>

25 oktober 2017

## Innehåll

<b>1</b>	<b>Scratchuppgifter</b>	<b>2</b>
1.1	Uppgift 1: Euklides algoritm . . . . .	2
1.2	Uppgift 2: Binärsökning . . . . .	3
1.3	Uppgift 3,4: Slumptal & Urvalssortering . . . . .	4
1.4	Uppgift 5: Fibonaccis talföljd . . . . .	6
<b>2</b>	<b>Tidrapport</b>	<b>7</b>

---

\*E-mail: [rasmus.sjobol@gmail.com](mailto:rasmus.sjobol@gmail.com)

<sup>†</sup>E-mail: [dh222kd@student.lnu.se](mailto:dh222kd@student.lnu.se)

# 1 Scratchuppgifter

## 1.1 Uppgift 1: Euklides algoritm

Vår implementation börjar med att fråga användaren om två tal  $a, b$  (här används  $a$ ). Då vill vi att  $a \geq b$  byter vi värde på dem ifall detta ej uppfylls. Sedan använder vi (pseudokod):

WHILE  $b \neq 0$

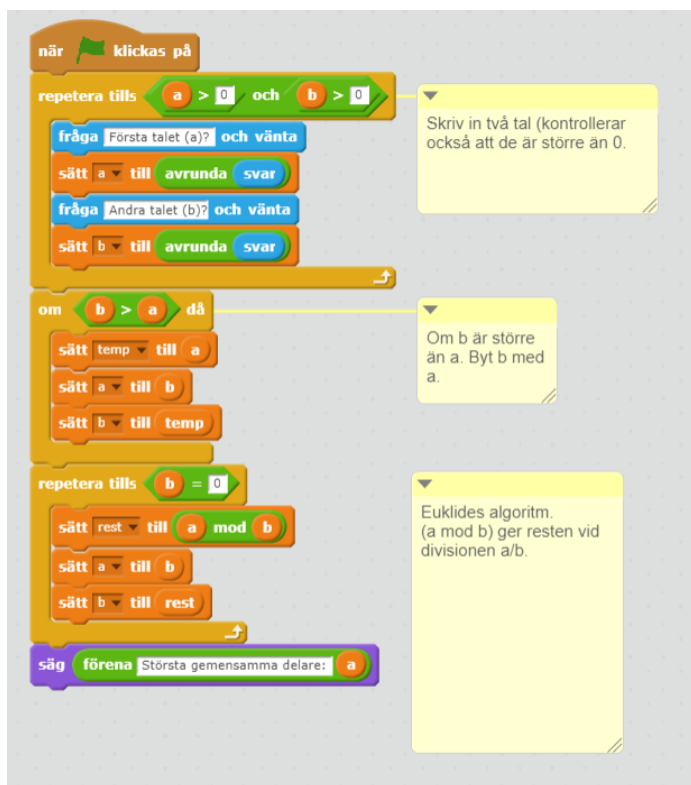
rest =  $a \bmod b$

$a = b$

$b = \text{rest}$

Först används modulus för att beräkna resten, sedan genomförs Euklides algoritm tills resten från divisionen mellan  $a$  och  $b$  är 0. Euklides algoritm säger att när resten är 0 är svaret funnet och är  $a$ , så där slutas loopen, och svaret skrivs ut.

Figur 1: Hela implementationen av Euklides algoritm



Länk till projektet:

<https://scratch.mit.edu/projects/178445830/#editor>

## 1.2 Uppgift 2: Binärsökning

På denna uppgiften har vi en sorterad lista på 10 olika heltal. Vi implementerade algoritmen rekursivt. Binärsökning jämför det mellersta elementet i en lista med det som eftersöks (här kallat nyckeln). Då finns tre fall:

Fall 1. Värdet på mittenelementet i listan = nyckeln. Detta medför att elementet som söks har funnits, och vi kan skriva ut dess index.

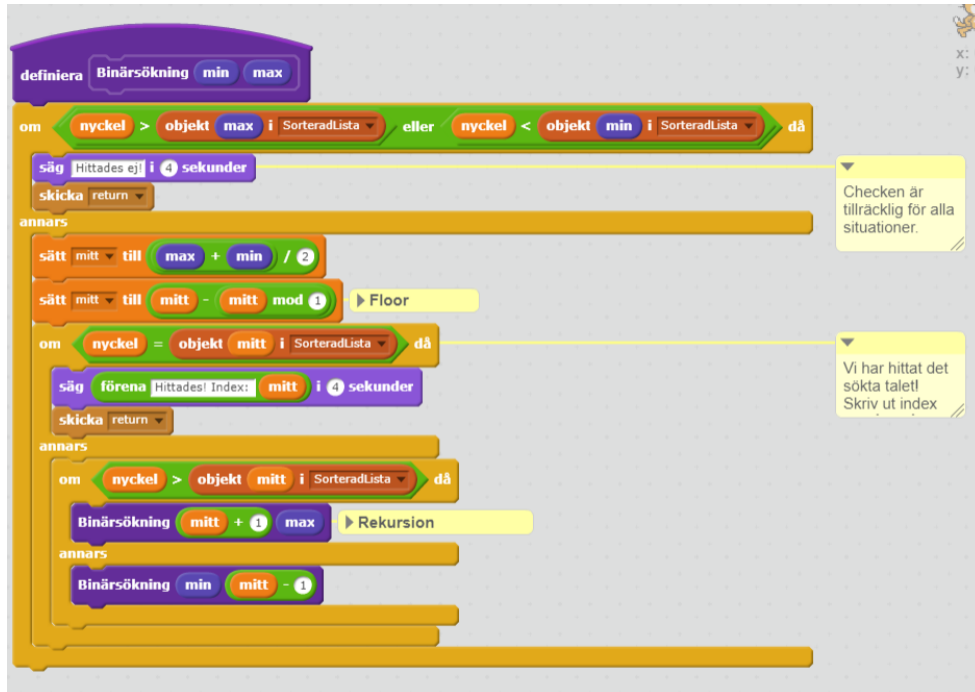
Fall 2. Värdet på mittenelementet < nyckeln. Eftersom listan är sorterad måste elementets som söks ha ett större indexvärde än det vi nu kollar på, således räcker det att söka igenom den halvan av listan. Mittenelementet kan också uteslutas då det redan är känt från 1) att det inte är det värde vi söker.

Fall 3. Värdet på mittenelementet > nyckeln. Med samma resonemang som i fall 2 kan vi söka igenom endast den halvan som har ett mindre index än mittenelementet.

Om värdet som söks ej hittades kan samma metod kallas igen, fast med nya gränser för de index som söks. De möjliga index en nyckel kan ha halveras således varje genomkörning. Indexet på det mellersta elementet inom gränserna  $max, min$  (heltal), fås genom  $(max + min)/2$ . Detta värde kan emellertid vara ett decimaltal, således avrundas det (här: alltid neråt, då det gör det enklare att följa koden). Scratchimplementationen för binärsökning finns att se som figur 2.

Om värdet inte finns med i listan kommer, vid något steg av genomsökningen värdet ligga utanför gränserna som söks, och då kan vi avbryta sökningen.

Figur 2: Implementation av binärsökning



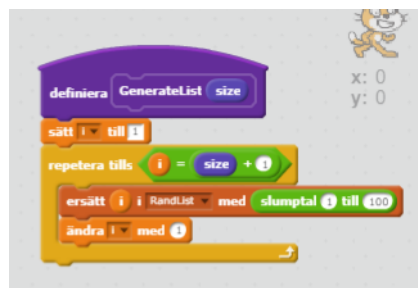
Länk till projektet:

<https://scratch.mit.edu/projects/178458263/#editor>

### 1.3 Uppgift 3,4: Slumptal & Urvalssortering

För att lägga in slumpmässigt valda tal i listan används:

Figur 3: Lista av slumpmässiga tal

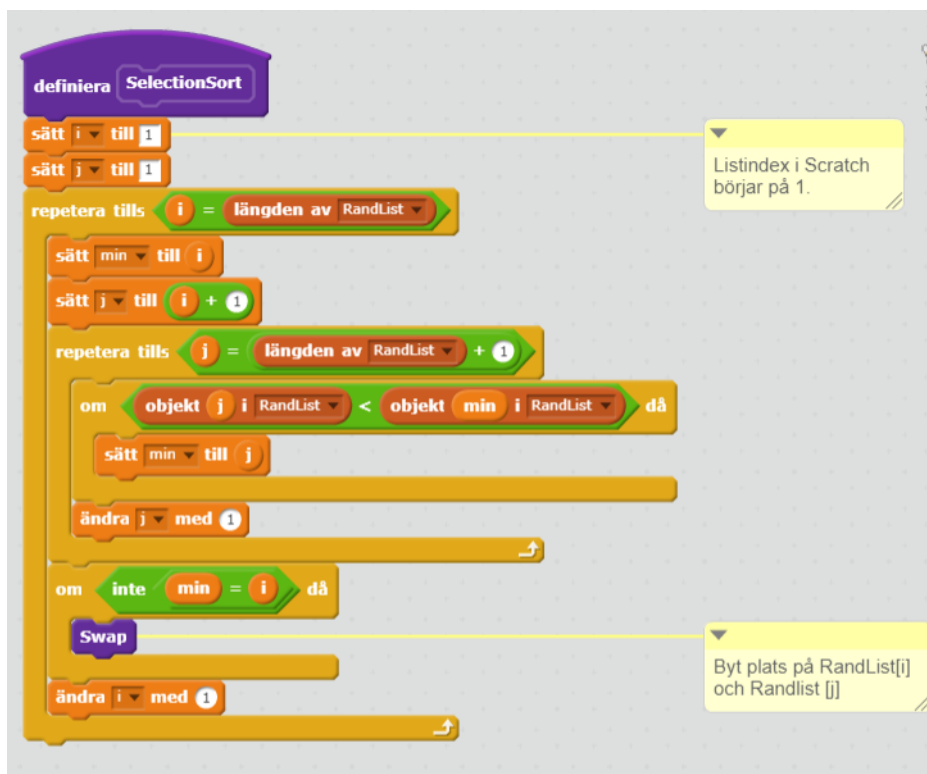


I figur 3 läggs ett tal mellan 1-100 på första platsen, sedan andra, tills vi nått slutpunkten. Detta görs här genom en scratch-implementation av en

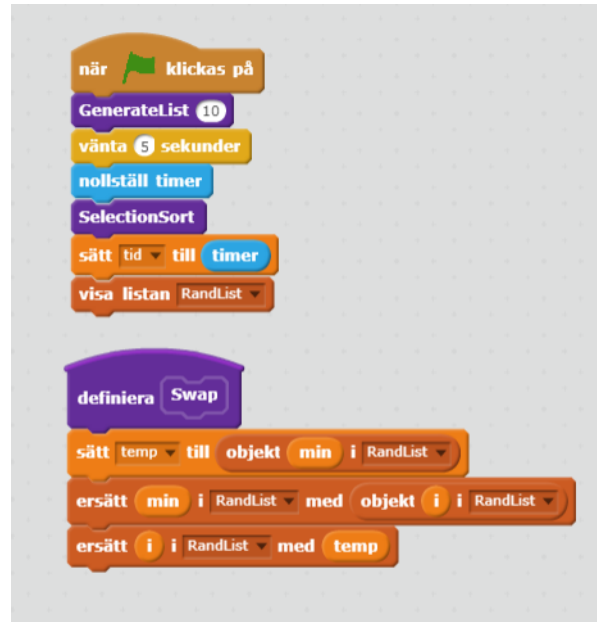
for-loop, och användandet av scratch:s “ersätt” block.

Urvalssortering (eng: selection sort) hittar först det minsta talet, och sätter det först i listan, sedan det näst minsta, etc. Därför används två loopar, en för vilken position vi letar efter och en för att finna det minsta talet relevant för positionen. Om det finns ett relevant tal mindre än talet som står på platsen byts dessa genom en swap metod vilken byter plats på talen(se: figur 5).

Figur 4: Urvalsortering



Figur 5: Swap

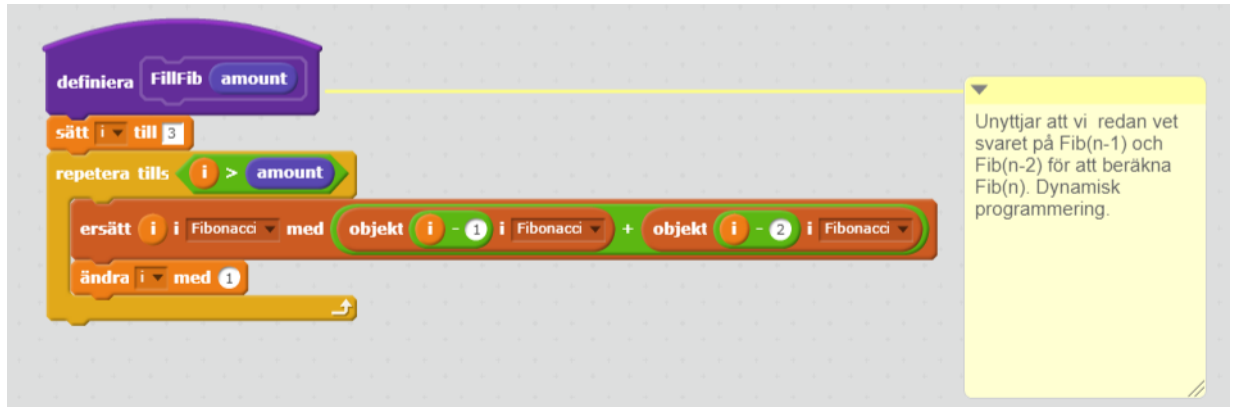


Länk till projektet:  
<https://scratch.mit.edu/projects/178469197/#editor>

## 1.4 Uppgift 5: Fibonaccis talföljd

Då fibonaccitalen bygger på varandra kan vi använda att vi vet  $\text{Fib}(n-1)$  och  $\text{Fib}(n-2)$  vid beräkning av  $\text{Fib}(n)$ . På grund av att vi utgår ifrån talen 0,1 kommer detta alltid att vara tillräckligt. Vi kan således lägga ihop dessa och lägga in dem på rätt plats i listan.

Figur 6: Fibonaccitalen



Länk till projektet:  
<https://scratch.mit.edu/projects/178477056/#editor>

## 2 Tidrapport

9/10 -2017 1h /person. 10/10 - 2017 1h / person.