Systemutveckling Ramverk

25 HP

Kursupplägget

Föreläsning			Innehåll (FM)	Övningar (EM)	Inlämning
1	mån 25 mar	D	Introduktion Typescript (Basic Types, Type Inferrence, Variable Declarations, Iterators and Generators)	TypeScripts Hemsida TypeScript in 5 minutes + övning	
2	ons 27 mar	V	Typescript forts. (Functions, Classes, Interfaces, Modules)		
3	fre 29 mar	V	Introduktion React (SPA, Virtuell DOM, Kap. 1-6)	Reacts Hemsida Main Concepts inklusive övningar i CodePen Kodövning (RP)	
4	mån 1 apr	D	React Playground (1-initial-setup + 2-layout)		Inlämning 1 ges ut
5	ons 3 apr	V	React forts. (Kap. 7-12)		
6	fre 5 apr	V	Create React App TS Intro	Övning ”Todo App”	Handledning
7	tis 9 apr	D	React Playground (3-navigation-with-state) React Playground (4-navigation-with-routes)	Kodövning (RP)	
8	ons 10 apr	V	React Playground (5-code-splitting) React Playground (6-error-boundary)	Kodövning (RP)	
9	fre 12 apr	V	React Playground (7-portals)	Muntlig Presentation Kodövning (RP)	Inlämning 1 in Inlämning 2 ut
10	mån 15 apr	D	React Playground (8-code-split-app-start)	Kodövning (RP)	
11	tis 16 apr	D	React Playground (9-api-lib-axios)	Kodövning (RP)	Handledning
12	tors 18 apr	V	React Playground (10-context)	Kodövning (RP)	
13	tis 23 apr	D	Algoritmer och välja Ramverk	Jobba med inlämning	Handledning
14	tors 25 apr	V	Tentaplugg		Inlämning 2 lämnas in
15	fre 26 apr	V	TENTAMEN		

Algoritmer

Ett sätt instruktioner för att lösa en klass av problem

Algoritmer - Vad är en algoritm?

”In mathematics and computer science, an algorithm is an unambiguous specification of how to solve a class of problems.” **- Wikipedia**

Begreppet har sitt ursprung inom matematiken och en av de tidigaste algoritmerna vi känner till gjordes av den grekiske matematikern Euclid of Alexandria. Han tog fram ett sett av instruktioner för att kunna beräkna den största gemensamma nämnaren för två heltal.

Algoritmer utgår alltid ifrån ett starttillstånd och med ett begränsat antal instruktioner återfinns sluttillståndet.

Algoritmer implementeras och exikveras vanligtvis inom datavetenskapen med hjälp av funktioner och/eller klasser i moderna programmeringspråk.

Algoritmer - En överblick

Det finns otroligt många typer av algoritmer och dom grupperas vanligtvis utifrån problemet som de försöker lösa. Nedan är några vanliga algoritmtyper inom datavetenskapen:

Sök-algoritmer

Sorterings-algoritmer

Maskininlärnings-algoritmer

Kompremerings-algoritmer

Navigerings-algoritmer

Nätverks-algoritmer

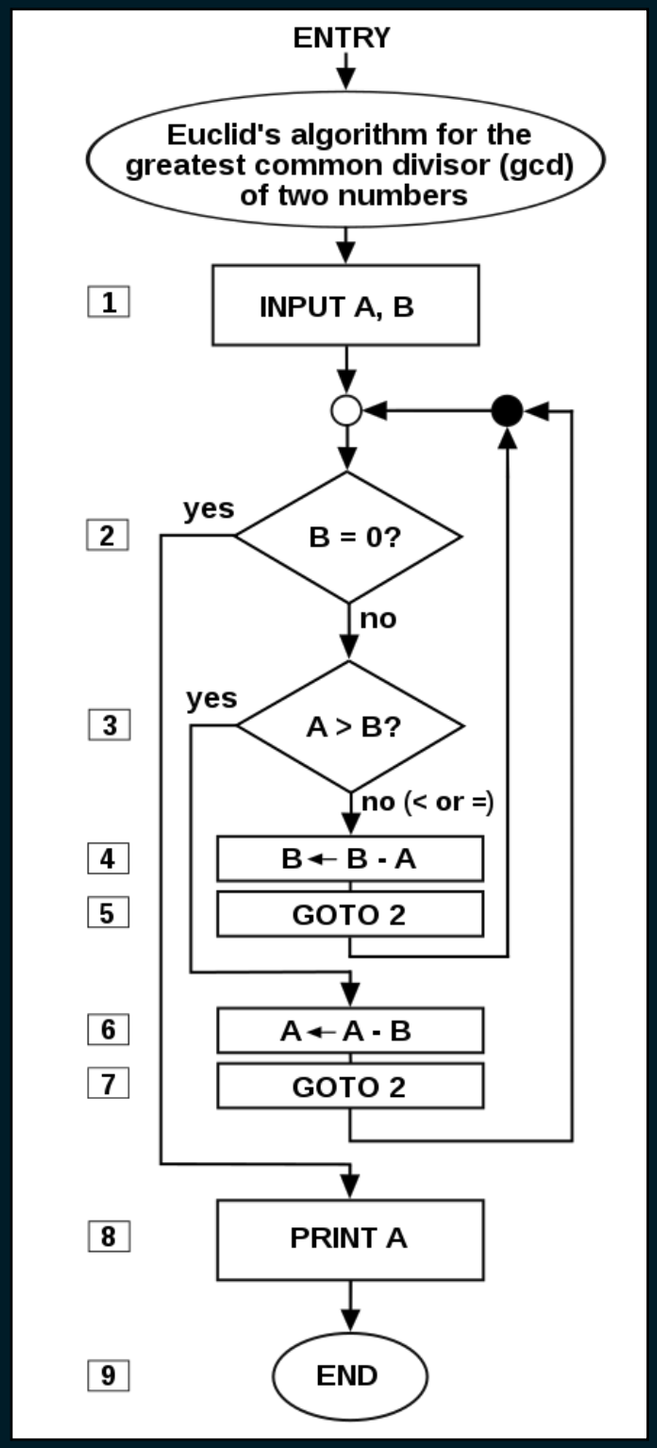
Databas-algoritmer

Kvant-algoritmer

Spel-algoritmer

https://en.wikipedia.org/wiki/List_of_algorithms

Algoritmer - Euclides algoritm



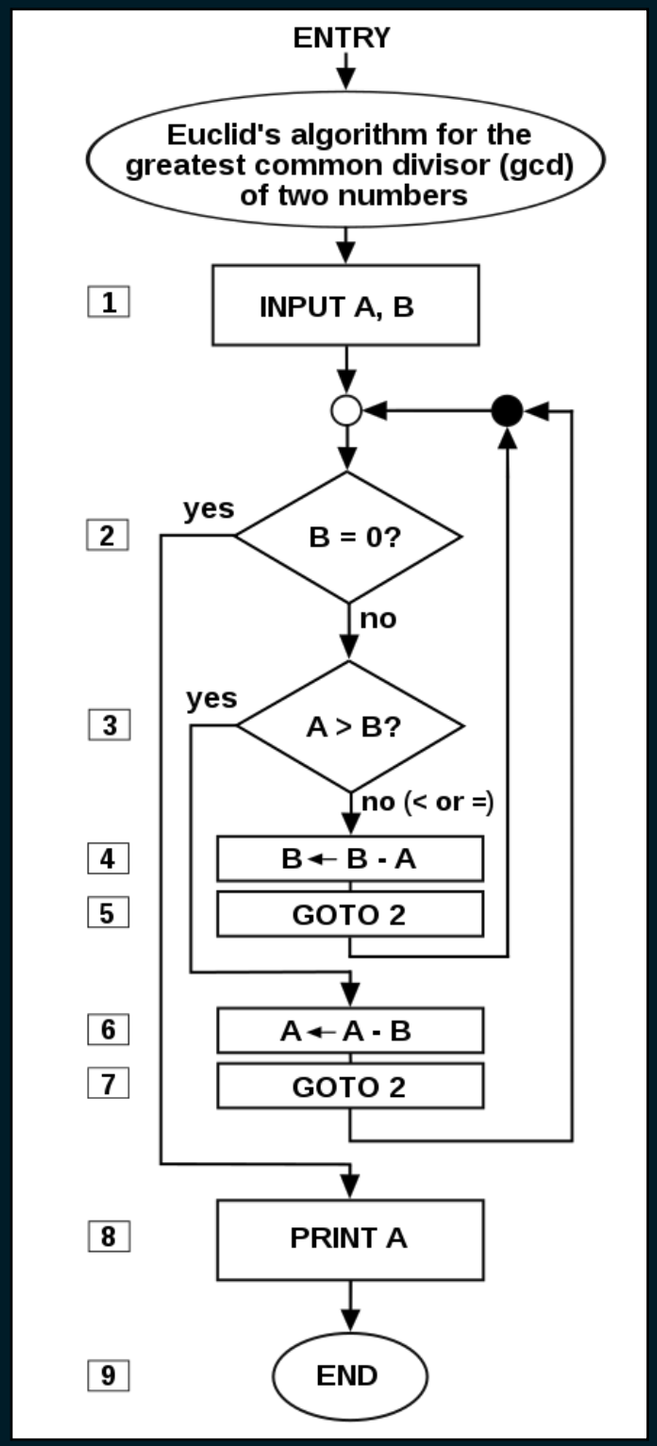
A	B
3	**6**
3	3
3	**0**

A	B
4	**12**
4	8
4	4
4	**0**

A	B
9	**3**
6	3
3	3
3	**0**

A	B
2	**5**
2	3
2	1
1	1
1	**0**

Algoritmer - Euclides algoritm



Kodövning

Implementera Euclides algoritm i Javascript.

Skapa en funktion som tar in två tal och returnerar den största gemensamma nämnaren för dessa två tal.

Använd console.log för att skriva ut svaret och kör filen med `node` i terminalfönstret.

Algoritmer - Problemklassificering

P ≠ NP

💰**Bounty: 1 miljon dollar om du löser problemet!**

Algoritmer - Problemklassificering

P (Polynomial Time)

För att ett problem skall ases var ett P-problem skall lösningstiden skala nära linjärt med storleken av problemetes input. Det är alltså problem som är lösbara även för stora input värden.

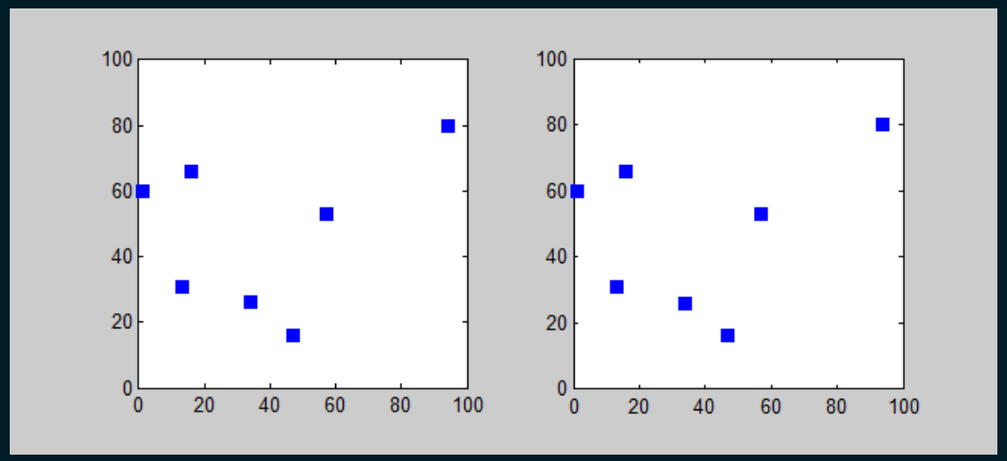
KLASSISKA P-PROBLEM
Greatest Common Divisor
Convex Hull
Sorting

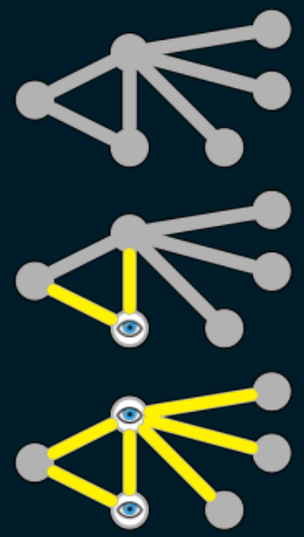
Algoritmer - Problemklassificering

NP (Non-deterministic Polynomial Time)

Problem som inte går att lösa i Polynomial tid klassas som NP-problem. Det innebär att en lösning för ett problem skalar för mycket med input-storleken för att vara lösbart för stora värden.

KLASSISKA NP-PROBLEM
Towers of Hanoi
Travel Salesman Problem
Vertex Cover





Algoritmer - Problemklassificering

NP vs **NP-Hard** vs **NP-Complete**

”P (Polynomial Time) : As name itself suggests, these are the problems which can be solved in polynomial time.

NP (Non-deterministic-polynomial Time) : These are the decision problems which can be verified in polynomial time. That means, if I claim that there is a polynomial time solution for a particular problem, you ask me to prove it. Then, I will give you a proof which you can easily verify in polynomial time. These kind of problems are called NP problems. Note that, here we are not talking about whether there is a polynomial time solution for this problem or not. But we are talking about verifying the solution to a given problem in polynomial time.

NP-Hard : These are at least as hard as the hardest problems in NP. If we can solve these problems in polynomial time, we can solve any NP problem that can possibly exist. Note that these problems are not necessarily NP problems. That means, we may/may-not verify the solution to these problems in polynomial time.

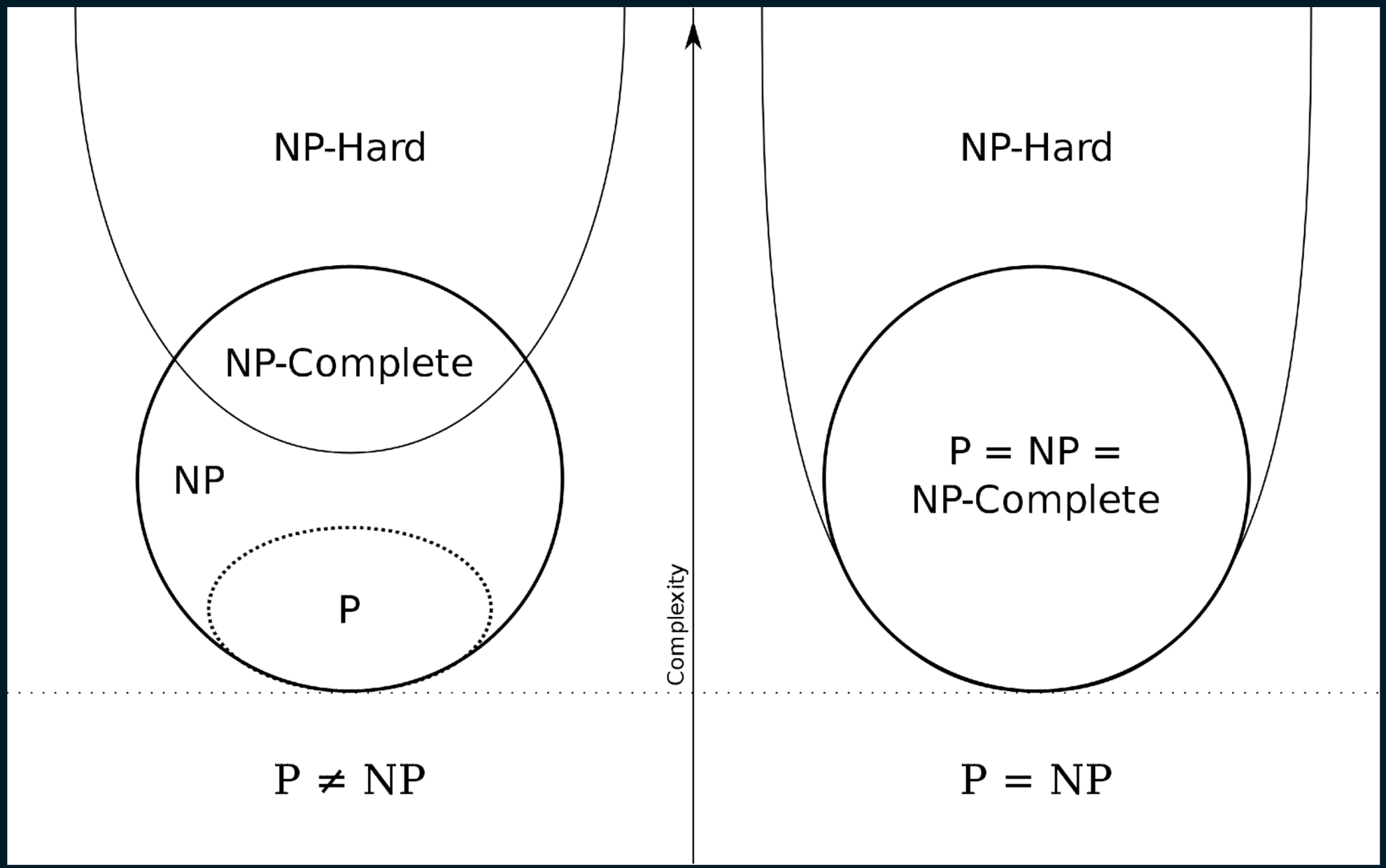
NP-Complete : These are the problems which are both NP and NP-Hard. That means, if we can solve these problems, we can solve any other NP problem and the solutions to these problems can be verified in polynomial time.”

- Stack Overflow

https://stackoverflow.com/questions/1857244/what-are-the-differences-between-np-np-complete-and-np-hard

Algoritmer - Problemklassificering

NP vs **NP-Hard** vs **NP-Complete**



Algoritmer - Komplexitet

Big O-notation

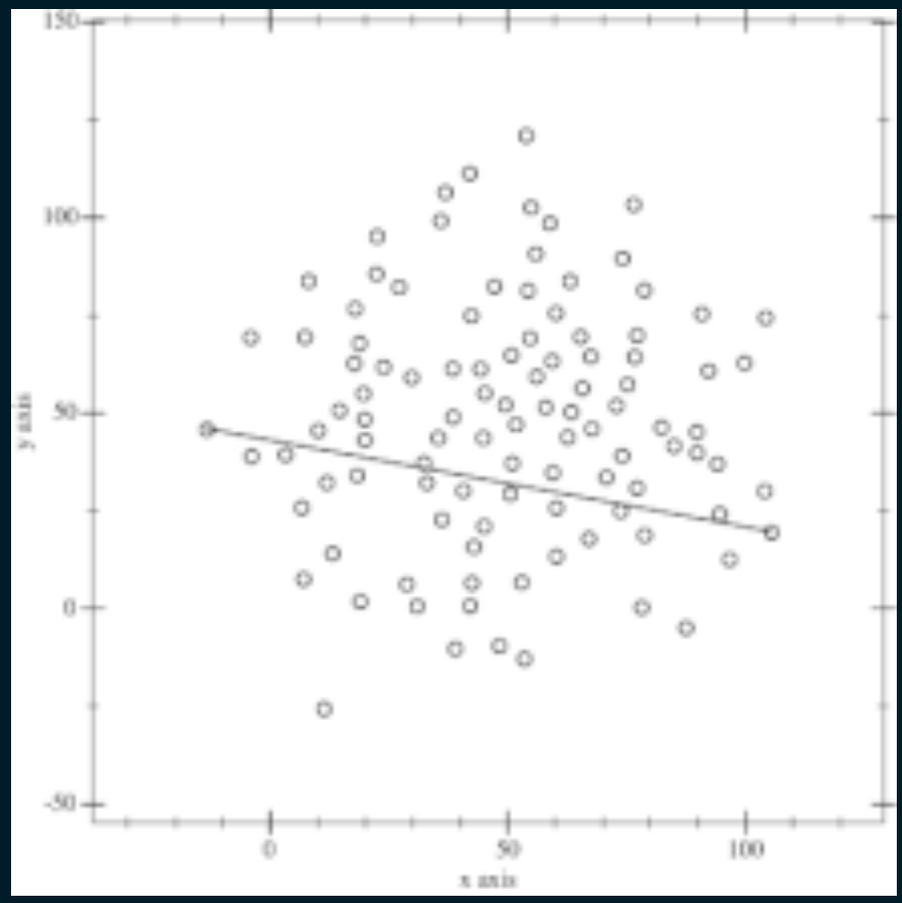
För att kunna avgöra hur effektiv en algoritm är används notationen ”Ordo” och skrivs tex som $O(n)$, $O(n^2)$ eller $O(n!)$ där n är antalet element som algoritmen utgår ifrån.

Notationen beskriver hur antalet uträkningar skalar med hur stor input-värdet är till algoritmen, tex hur många element det finns i en lista, eller antalet punkter på en graf.

Algoritmer - Komplexitet

Konvexa höljet

Ett problem som går ut på att hitta de punkterna i en graf som tillsammans omsluter alla andra punker. Problemet kan lösas med flera algoritmer.



Brute Force algoritmen, där man för varje punkter jämför mot alla andra punker, har en komplixitet $O(n^2)$.

Quick Hull algoritmen, löser problemet genom att dra en baslinje mellan ändpunkterna för attt sedan bygga vidare tills alla punker är på rätt sida av linjerna. Algoritmen har en komplexitet på O(n*log(n)).

Algoritmer - Komplexitet

Konvexa höljet

En jämförelse mellan Brute Force algoritmen och Quick Hull algoritmen —>

antal punkter	Time BF	Time QH
3	0.00069 ms	0.00071 ms
5	0.00123 ms	0.00093 ms
10	0.00611 ms	0.00120 ms
20	0.07582 ms	0.00166 ms

O(n2) vs **O(n*log(n))**

Algoritmer - Komplexitet

Best Case - Average Case - Worst Case

När man pratar om komplexiteten för en algoritm syftar man alltid på Average Case lösningen. Den genomsnittliga tiden för alla olika input's.

Ibland kan det även vara viktigt att ta hänsyn till Best Case &| Worst Case för algoritmen. Detta inträffar bara för vissa input's, tex Best Case för en sorteringsalgoritm är i många fall när input-listan redan är sorterad.

Även resurskomplexiteten (minnes-användningen på datorn) **är viktigt att tänka på utifall att hårdvaran som algorimen skall köras på har lite minne. Algoritmer som vanligvis tar upp mycket minne är de som använder en recursiv implementation.**

Algoritmer - Komplexitet

Sorteringsalgoritmer

Name	Best	Average	Worst	Memory	Stable
Quicksort	$n \log n$ variation is n	$n \log n$	n^2	$\log n$ on average, worst case space complexity is n; Sedgewick variation is $\log n$ worst case.	Typical in-place sort is not stable; stable versions exist.
Merge sort	$n \log n$	$n \log n$	$n \log n$	n A hybrid block merge sort is $O(1)$ mem.	Yes
In-place merge sort	—	—	$n \log^2 n$ See above, for hybrid, that is $n \log n$	1	Yes
Heapsort	n If all keys are distinct, $n \log n$	$n \log n$	$n \log n$	1	No
Insertion sort	n	n^2	n^2	1	Yes
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No
Selection sort	n^2	n^2	n^2	1	No
Timsort	n	$n \log n$	$n \log n$	n	Yes
Cubesort	n	$n \log n$	$n \log n$	n	Yes
Shell sort	$n \log n$	Depends on gap sequence	Depends on gap sequence; best known is $n^{4/3}$	1	No
Bubble sort	n	n^2	n^2	1	Yes
Binary tree sort	$n \log n$	$n \log n$	$n \log n$(balanced)	n	Yes
Cycle sort	n^2	n^2	n^2	1	No
Library sort	n	$n \log n$	n^2	n	Yes
Patience sorting	n	—	$n \log n$	n	No
Smoothsort	n	$n \log n$	$n \log n$	1	No
Strand sort	n	n^2	n^2	n	Yes
Tournament sort	$n \log n$	$n \log n$	$n \log n$	n[9]	No
Cocktail sort	n	n^2	n^2	1	Yes
Comb sort	$n \log n$	n^2	n^2	1	No
Gnome sort	n	n^2	n^2	1	Yes
UnShuffle Sort[10]	n	kn	kn	In-place for linked lists. `n * sizeof(link)` for array. n+1 for array?	No
Franceschini's method[11]	—	$n \log n$	$n \log n$	1	Yes
Block sort	n	$n \log n$	$n \log n$	1	Yes
Odd–even sort	n	n^2	n^2	1	Yes

Algoritmer - Recursion

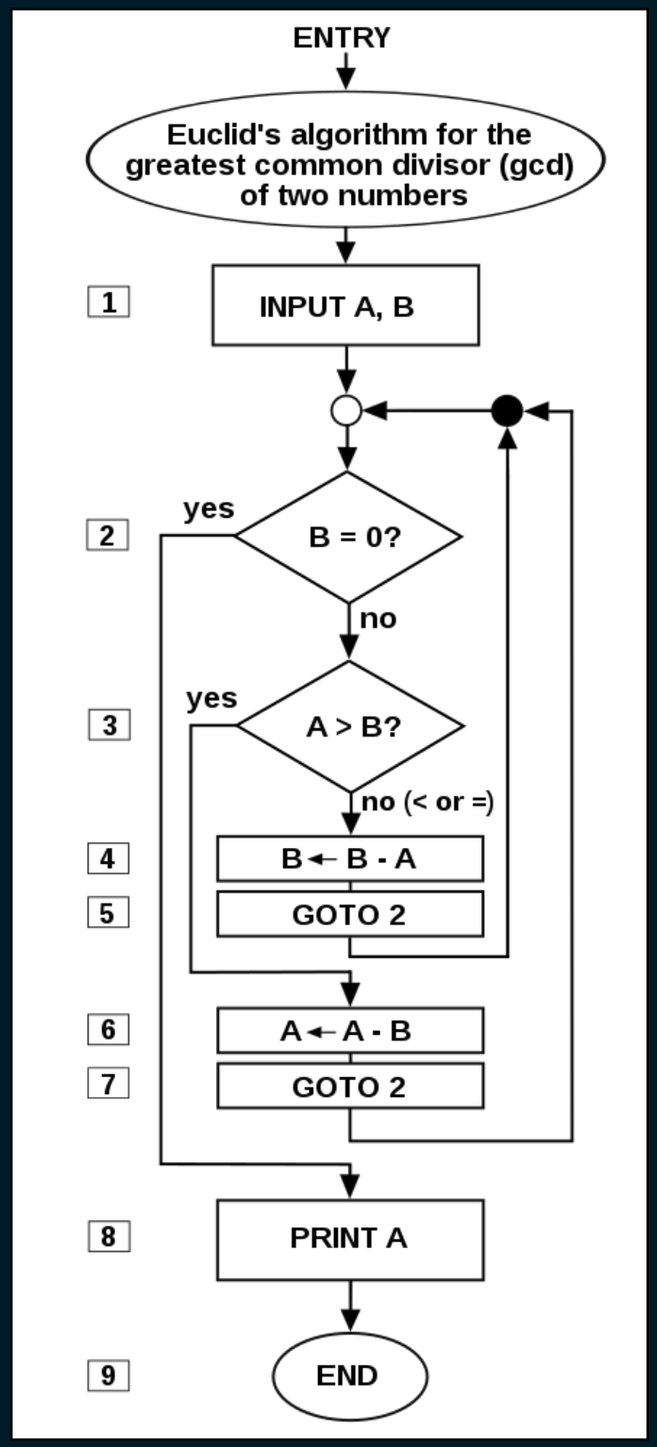
Recusion är en implementationsteknik snarare än en algoritm i sig och innebär att en funktion har ett anrop till sig själv vilket skapar en loop.

För att recursions-loopen inte skall hålla på i all evighet behövs ett basfall som när sant returnerar ett värde. Om recusionskoden har implementerats rätt kommer värdet att skickas tillbaks genom alla funktionsinstanser till början.

```
function myRecursiveFunction(someValue) {
    // Base Case
    if (condition) {
        return someValue;
    }

    // Recursive call
    return recursiveFunction(someValue + 1);
}
```

Algoritmer - Euclides algoritm



Kodövning

Implementera Euclides algoritm i Javascript.

Skapa en funktion som tar in två tal och returnerar den största gemensamma nämnaren för dessa två tal.

Använd console.log för att skriva ut svaret och kör filen med `node` i terminalfönstret.

Men denna gången skall funktionen vara recursiv, dvs anropa sig själv för att komma fram till lösningen på problemet.

Algoritmer - Korrekthet

Det sista som kan vara bra att känna till är hur man vet att en algoritm faktsikt fungerar.

För att säkerställa det behöver man ta fram ett matematiska bevis för algoritmen men det ligger utanför den här kursen.

Välja Ramverk

Vi måste använda ReactSuperCharged för det kom nyss ut!

Välja Ramverk

Innan vi börjar fundera på vilka ramverk eller bibliotek vi skall använda oss av i vårat projekt behöver vi först undersöka vilka behov och krav vi har.

När vi har behoven och kraven på plats är första frågan vi bör fundera på va vi kan skriva själva och vad vi behöver leta ramverk för.

Om vi kommer fram till att ett ramverk behövs, påbörjas nästa steg. Det innebär en djupare undersökning om vilka ramverk som kan lösa behoven och vilket av dessa ramverk som bör väljas.

Det finns ingen snabb guide i hur man väljer ramverk utan det är något som kommer med erfaranhet.

Välja Ramverk

Några saker värda att tänka på om vi kommit fram till att ett ramverk faktiskt behövs:

Hur många använder sig av ramverket?
Vem är utgivaren av ramverket?
Underhålls ramverk fortfarande?
Finns det någon uttalad framtidsplan för ramverket?
Är ramverket open source eller finns det andra likvärdiga ramverk som bättre löser vårt krav och på så vis är mer kostnadseffektivt i längden?

🚌**-låt oss ta en GitHub tour tillsammans…**

https://github.com/

Nästa lektion

Handlening & Tentaplugg

Läsanvisningar

P vs NP

https://en.wikipedia.org/wiki/P_versus_NP_problem

Sorteringsalgoritmer

https://en.wikipedia.org/wiki/Sorting_algorithm

Resten av dagen

Handlening & Inlämningsuppgift

Tack