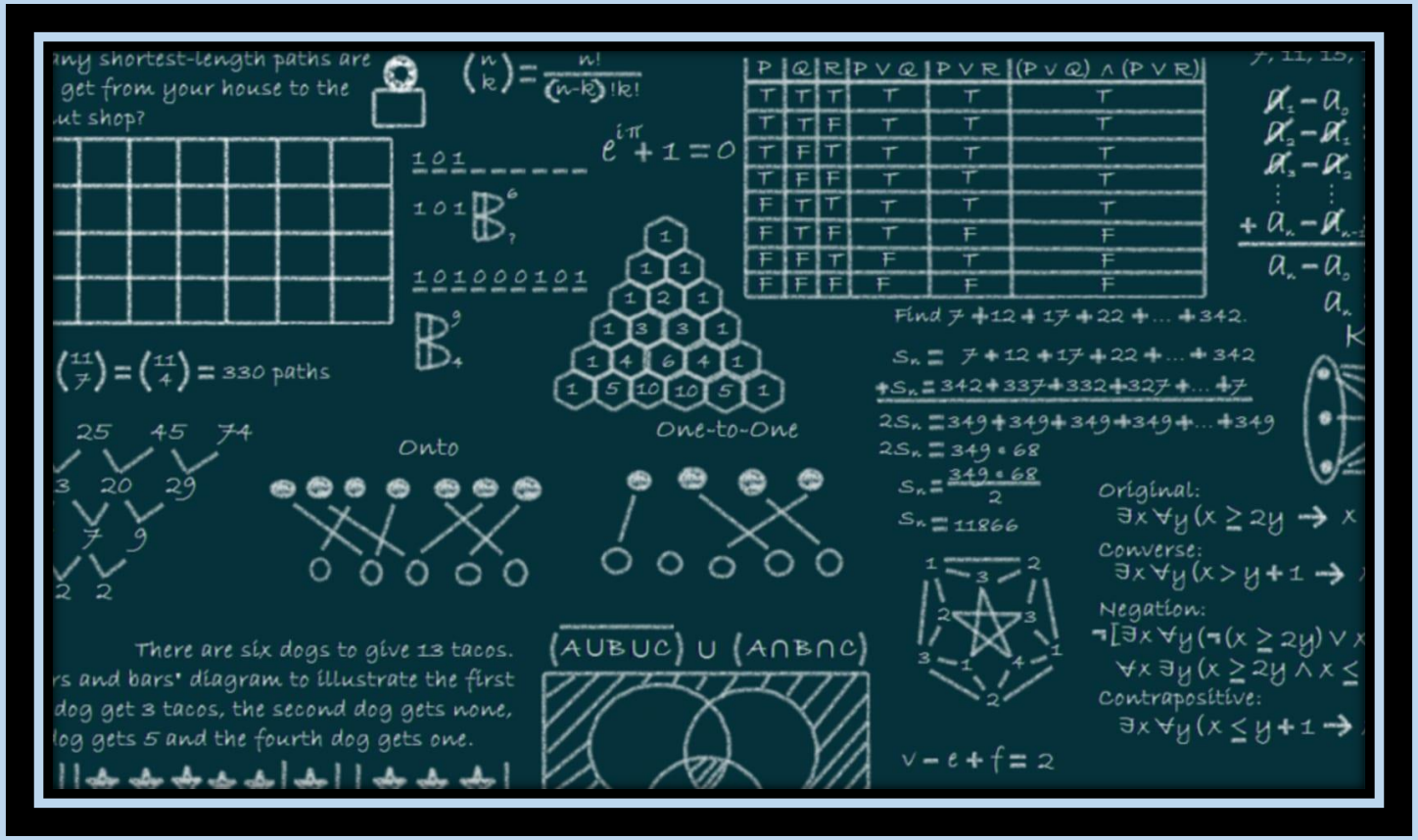



# FÖRELÄSNING 3

Datastrukturer och algoritmer
KYH – 2022 HT

Andreas Nilsson Ström

# Repetition

# Agenda

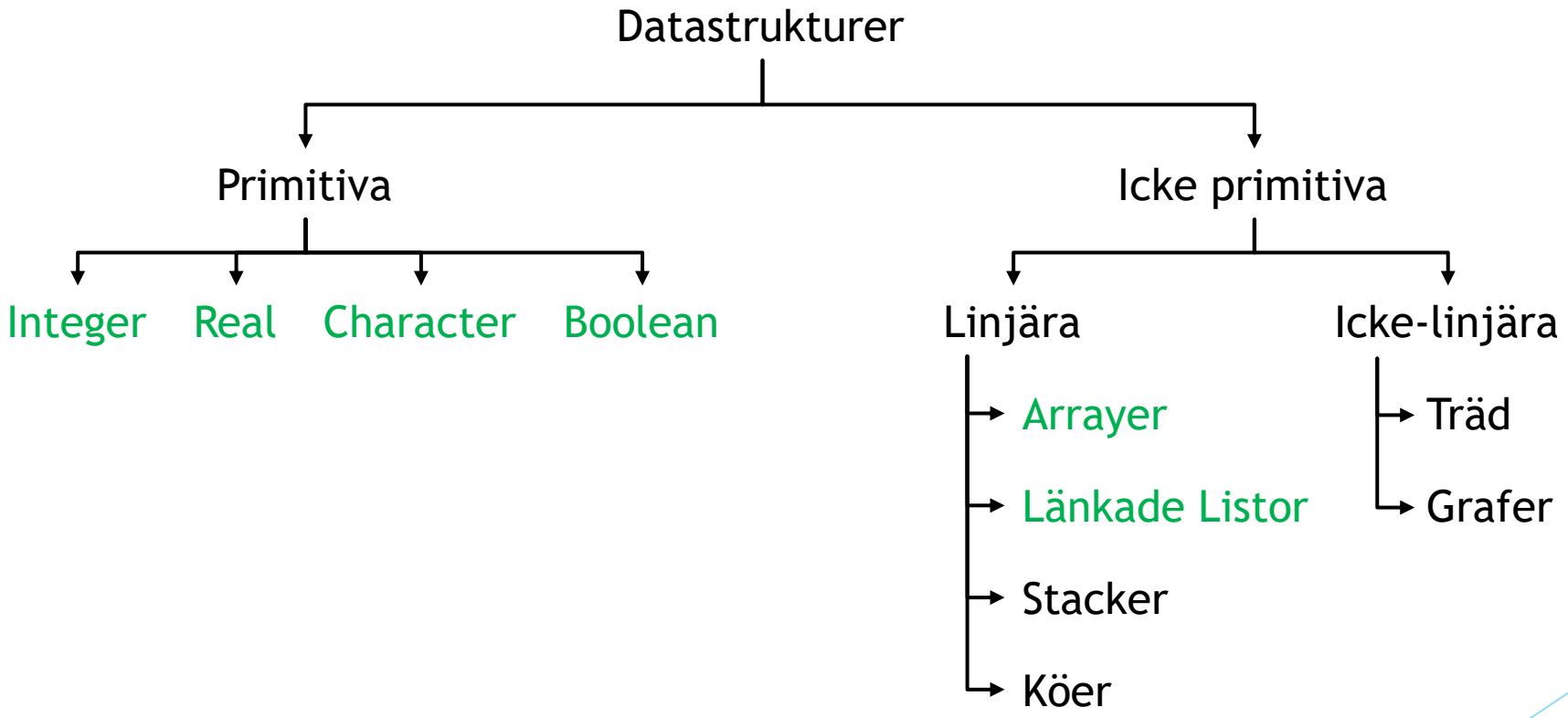
- Flera nya varianter och termer! (Det sista om länkade listor! … ?)
  - Lite mer praktiskt, vad kan vi göra med dessa saker
  - Översikt över varianter och detaljer
- Vilka sätt har vi att jämföra datastrukturers körtid?
  - Hur pratar vi om det?
- Översikt över nästa vecka
- Labbande och övningar

# Agenda

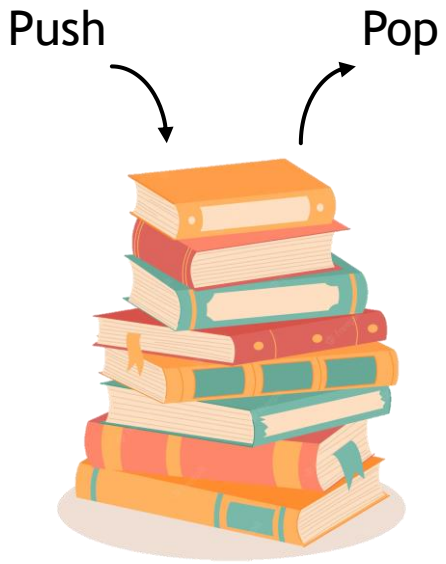
- Flera nya varianter och termer! (Det sista om länkade listor! … ?)
  - Lite mer praktiskt, vad kan vi göra med dessa saker
  - Översikt över varianter och detaljer
- Vilka sätt har vi att jämföra datastrukturers körtid?
  - Hur pratar vi om det?
- Översikt över nästa vecka
- Labbande och övningar

# Nya varianter

# Exempel på datastrukturer



# Stack




- Stack – **LIFO** ("**last in first out**")
- (Svenska: "Hög"?)
  - Push (Kallas även append, add, add_first, …)
  - Pop (Kallas även remove, remove_first, …)

**Stack:**

```
push(x)     # Lägger x på toppen av stacken
pop()       # Tar bort och ger tillbaka elementet från toppen

is_empty() # True om stacken är tom
peek()      # Ger tillbaka elementet från toppen, utan att ta bort
size()      # Ger antalet element i stacken
```

# Queue (Kö)


- Queue - **FIFO** ("**first in first out**")
  - Enqueue
  - Dequeue

**Queue:**

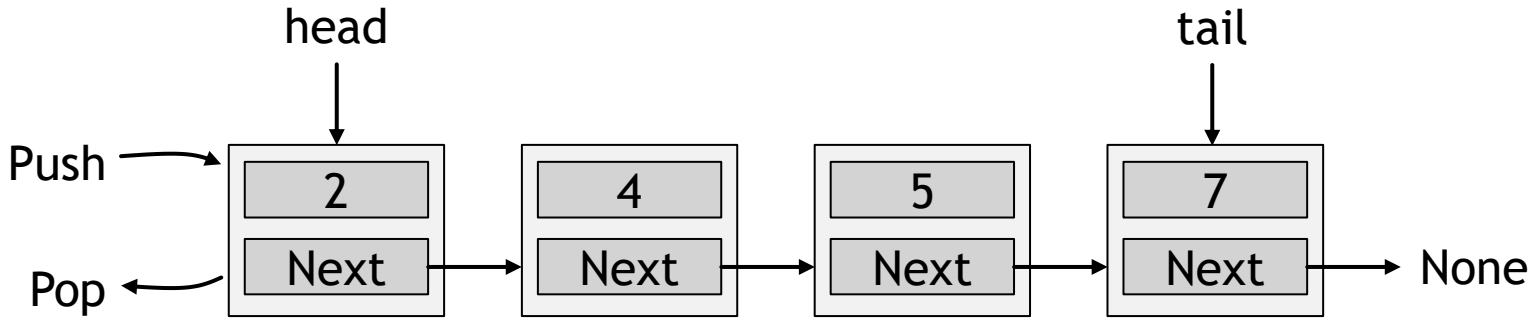
```
enqueue(x) # Lägger till x sist i kön
dequeue()  # Tar bort och ger tillbaka elementet först i kön

is_empty() # True om kön är tom
peek()     # Ger tillbaka första elementet, utan att ta bort
size()     # Ger antalet element i kön
```

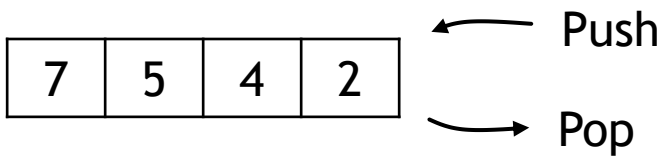
# Demo:

- Stack och kö – Implementering med hjälp av en länkad lista

# Underliggande datastrukturen?

- Notera att jag aldrig sa något om vad vi ska ha för datastruktur när jag introducerade Stack och Queue.
- I mitt demo använde jag en Linked List, men vi kan använda vilken annan linjär datastruktur som helst.



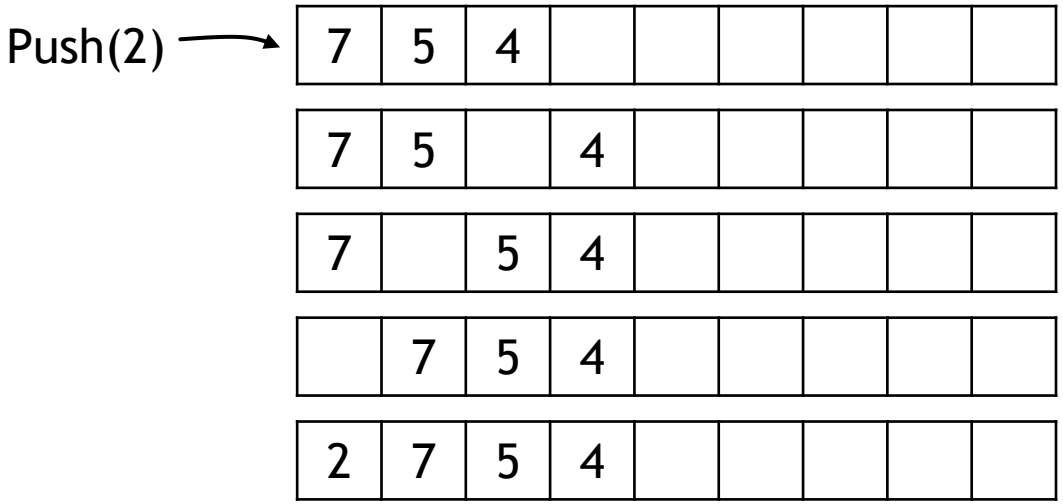
# Övning

- Implementera klassen Stack med en vanlig Python-lista
  - push(x)
  - pop()
  - is_empty()
  - peek()
  - size()

- Tips: __init__() skapar en tom lista, sen arbetar vi med den listan i metoderna

# Lägga till i början av en Array

- Dålig idé, gör inte detta. Lägg till i slutet! Varför?
- Push behöver flytta undan alla värden först innan det nya läggs till.




- Men om det är en kö då? Då tar ju dequeue bort första elementet och flyttar allt.. Ja. Inte bra! Bättre med en länkad lista kanske?

# Bag (Påse/säck)

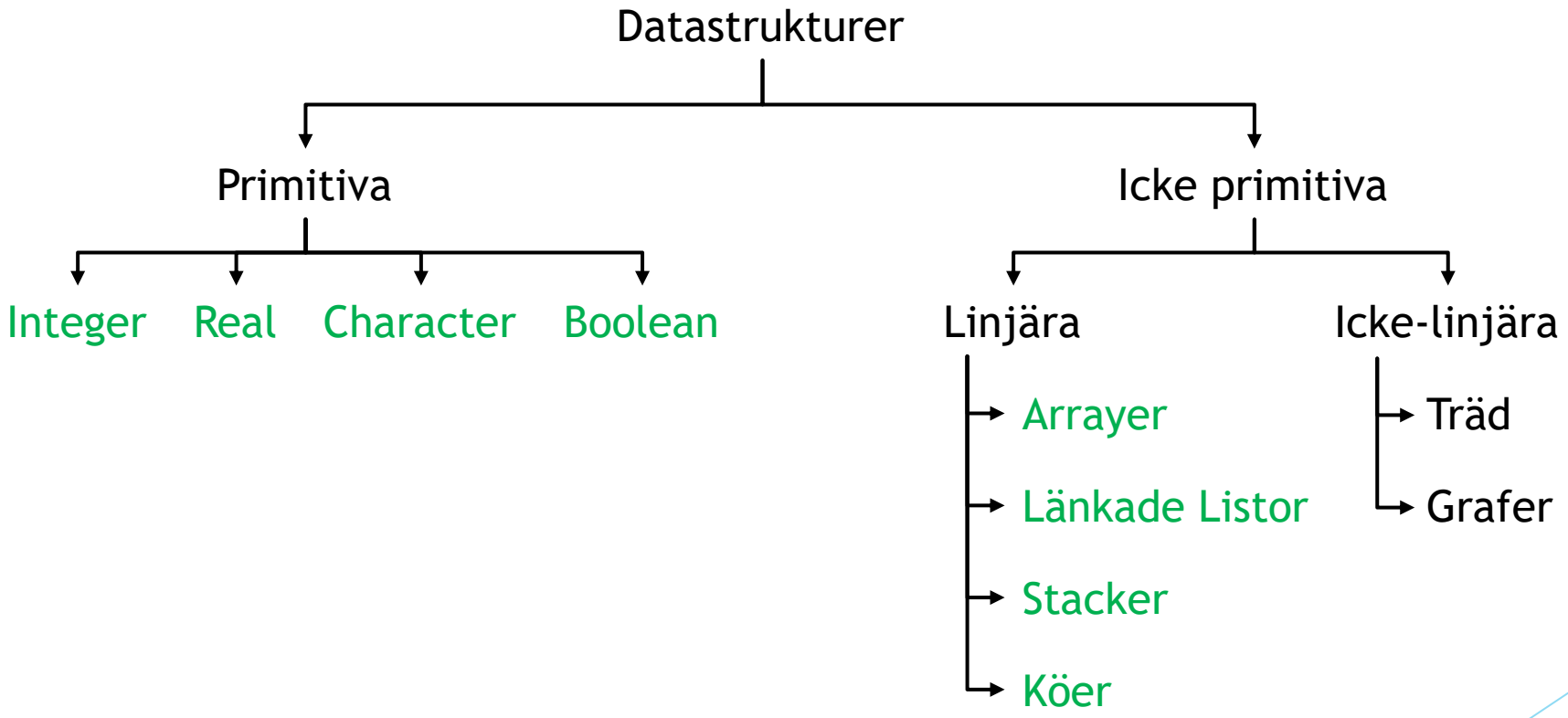

- Lik de andra, men kan bara lägga till
  - Add

**Bag:**

```
add(x)      # Lägger x i påsen

is_empty() # True om påsen är tom
size()      # Ger antalet element i stacken
```

- Vad är den bra till? Ex: Om vi bara vill lägga till saker, sen loopa över dem.
  Till exempel bra för att räkna statistik.
  Medelvärdet, högsta värdet, lägsta värdet, etc.

# Exempel på datastrukturer



# Agenda

- Flera nya varianter och termer! (Det sista om länkade listor! … ?)
  - Lite mer praktiskt, vad kan vi göra med dessa saker
  - Översikt över varianter och detaljer
* Vilka sätt har vi att jämföra datastrukturers körtid?
  - Hur pratar vi om det?
- Översikt över nästa vecka
- Labbande och övningar

# Komplexitet

# Vad är komplexitet? (Inom beräkningar)

"Computational Complexity" på engelska

Komplexiteten av en algoritm:

- Hur mycket **resurser** använder en algoritm i förhållande till egenskaperna på sin input/data?

**Resurser**:

- Tid
- Utrymme (Minne, Lagring)
- Andra typer av resurser
  - Antal jämförelser
  - Antal uppslagningar i en lista
  - Antal matematiska operationer

# Vad är komplexitet? (Inom beräkningar)

"Computational Complexity" på engelska

Komplexiteten av en algoritm:

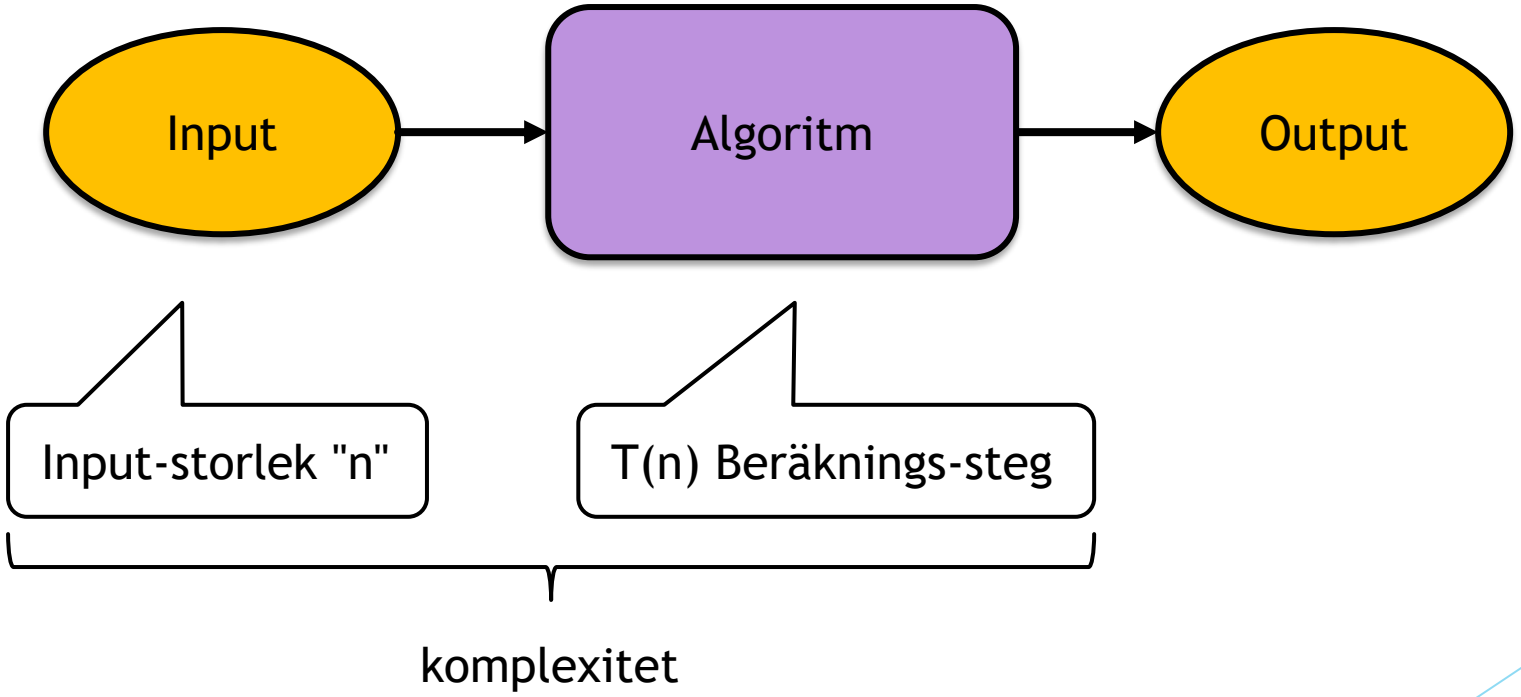
- Hur mycket resurser använder en algoritm i förhållande till **egenskaperna** på sin indata?

**Egenskaper på indata:**

- Ofta någon form av storlek
  - Antalet noder i minnet
  - Längen på input-strängen

# Vad är komplexitet? (Inom beräkningar)

Exempel för tid (T):



# Tids-komplexitet

Den exakta körtiden för en algoritm beror på..

- Teoretiskt sett: Hur problemet är löst
- Men i praktiken: Mjukvarans och hårdvarans detaljer som kör den

Saker som kan påverka, exempel:

- Hur översätts koden till maskinspråk?
- Processor (hastighet, effektivitet i körning)
- Minnesarkitektur
- Hur hanteras allt av operativsystemet

Extremt svårt att förutspå och modellera!
Uppmätta körtider går oftast inte att upprepa

# Tids-komplexitet?

Den exakta körtiden för en algoritm beror på..

- Teoretiskt sett: Hur problemet är löst
- Men i praktiken: Mjukvarans och hårdvarans detaljer som kör den

**Summa summarum:**

- Exakt uppmätt tids-komplexitet är inte relevant att prata om (om man inte jobbar med forskning..)
- Vad som har betydelse är algoritmens beteende är när input växer!
  - "**Asymtotisk komplexitet**" – Hur växer kurvan på en graf när input växer?
  - Det är oftast det här folk menar när vi pratar om komplexitet

# Exempel: Räkna bokstäver

- Beräkna antalet "a" i strängen "alfa beta gamma delta"

- Vi börjar på första bokstaven och arbetar oss åt höger, bokstav för bokstav
- Träffar vi på ett "a" så lägger vi på 1 på vår räknare
- Vi behöver alltså gå igenom hela strängen för att veta svaret!

alfa beta gamma delta

# Exempel: Hitta första matchande

- Hitta första "h" i "hello world!"

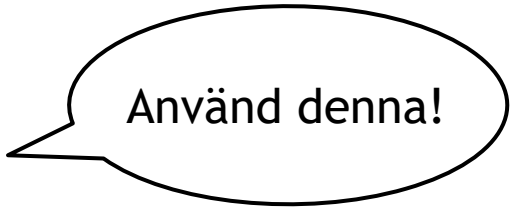
- Vi börjar på första bokstaven och arbetar oss åt höger, bokstav för bokstav
- Träffar vi på ett "h" så returnerar vi True, annars False
- Bästa fall: Vi hittar "h" på första platsen
- Värsta fallet: Vi behöver gå igenom hela strängen för att veta svaret!

hello world

# Körtiden varierar

Vi måste tänka på hur input kan påverka tiden!

- Bästa fallet (Ingen bryr sig)
  - **Minimala** resursanvändningen över alla möjliga inputs
- Värsta fallet (Standard att prata om)
  - **Maximala** resursanvändningen över alla möjliga input
- Medelfallet
  - Medelanvändningen av resurser över alla möjliga inputs
  - Kräver förmodligen en del matematik (Sannolikhetsfördelningar etc.. Det är en annan kurs)



# Big O Notation

- Ett sätt att beskriva hur beteendet är när problemet växer
- Skrivs O(…) där … beror på hur beteendet är (Baseras på input)
  - (Uttalas "Ordo", kallas även "Ordo-notation")
- Beskriver om beteendet är till exempel
  - Linjärt: O(n)
  - Konstant: O(1)
  - Kvadratiskt: O(n^2)

# Komplexitet

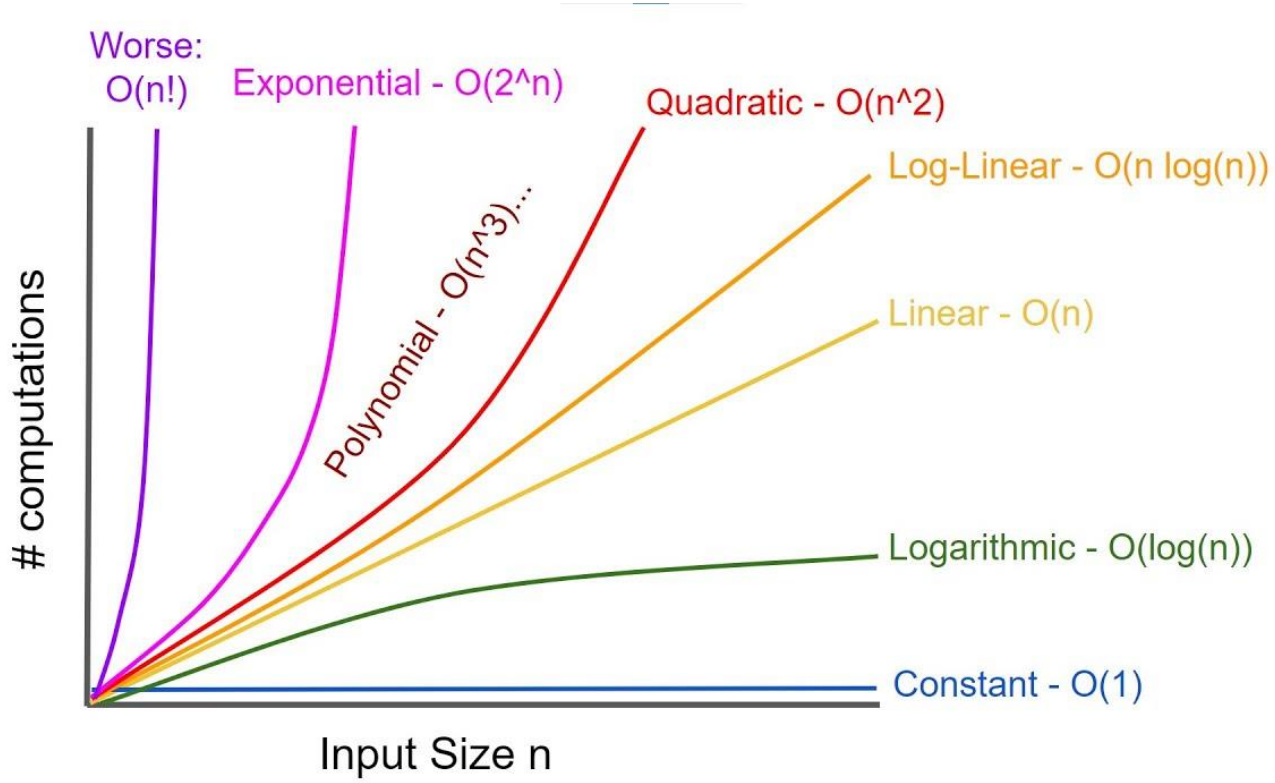



Bild: Källa

# Exempel

- Kolla om bokstav finns i sträng?
  - Behöver i värsta fall gå igenom hela strängen: O(n) – Linjärt
- Ta värdet i början av en lista?
  - Beror inte på listans storlek alls: O(1) – Konstant
- Kolla om strängen innehåller dubbletter?
  - Vi behöver jämföra varje bokstav med alla andra bokstäver
  - En loop i en loop: O(n^2) - Kvadratiskt

# Komplexitet och Big O-notation

- Vi återkommer till detta mer när vi kollar på andra algoritmer och datastrukturer under kursens gång

# Tankar…

- Att bara slänga fler eller starkare processorer på ett problem är inte alltid rätt lösning. Det är inte ett bra alternativ till att ha optimerade algoritmer och ha valt rätt datastruktur för problemet.
- Ibland räcker det med "tillräckligt bra". Om vi vet att datan vi ska hantera alltid är väldigt liten så spelar det kanske mindre roll vilken lösning som väljs. Tidsåtgången är minimal oavsett.
- Glöm inte att vi kan behöva kolla på hur olika resurser används, inte bra komplexitet
  - Minne, diskutrymme, tid, bandbredd, …

# Frågestund

- Vad är komplexiteten av era implementeringar av Stack med Pythons listor?

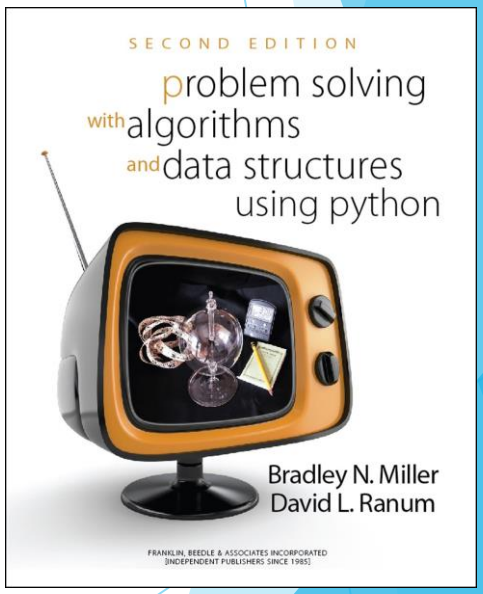
# Nästa vecka

- Måndag
  - Abstrakta datastrukturer
  - Deque
  - Sorterade listor
  - Intro till Labb 2
- Onsdag
  - Rekursion
  - Sortering och sökning
- Fredag
  - Deadline Labb 2

# Mer läsning

Bok

- **Problem Solving with Algorithms and Data Structures Using Python**
- Länk
- Kapitel: 3.2, 3.3, 3.5, 3.10, 3.12, 3.13
- (Repetition av tidigare dagar: Kapitel 1, Kapitel 2)




Extramaterial

- Youtube: Algorithms Explained: Computational Complexity

# Till nästa gång…

- Övningar: Hackerrank
  - https://www.hackerrank.com/domains/data-structures?filters%5Bsubdomains%5D%5B%5D=linked-lists
  - Till och med "Compare two linked lists"
- Mer läsning:
  - https://www.happycoders.eu/algorithms/array-vs-linked-list/