NTNU

Det skapende universitet

TDT4110 Informasjonsteknologi grunnkurs:

Tema: Algoritmer i praksis

Professor Alf Inge Wang

Læringsmål og pensum

- Mål
  - Lære å forstå og kunne programmere algoritmer for søk og sortering.
  - Lære å forstå og kunne bruke rekursjon.
- Pensum
  - Theory Book IT Grunnkurs: Algorithms, s.107-170
  - Starting out with Python:
    - 3rd edition: Chapter 12 Recursion

Algoritmeeffektivitet

- Måles som antall utførte instruksjoner
  - For eksempel antall sammenlikninger (if-setninger), antall ombytninger mellom plasser i ei liste etc.
- Ser på hva som skjer for svært mange inputs
- Læreboka bruker stor Theta-notasjon til å representerer klasser av effektivitet
  - Eks: Innstikksortering er $\Theta(n^2)$
- For algoritmer kan man også vurdere:
  - Worst-case: Det tilfelle er algoritmen vil ta lengst tid
  - Best-case : Det tilfelle algoritmen vil ta kortest tid

Funksjoners vekst

n	n^2	n^3	2^n	log(n)	nlog(n)
1	1	1	2	0	0
2	4	8	4	1	2
3	9	27	8	2	5
4	16	64	16	2	8
5	25	125	32	2	12
6	36	216	64	3	16
7	49	343	128	3	20
8	64	512	256	3	24
9	81	729	512	3	29
10	100	1000	1024	3	33
100	10000	1000000	1,2677E+30	7	664
1000	1000000	1000000000	1,072E+301	10	9966
10000	100000000	1000000000000	#NUM!	13	132877
100000	10000000000	1000000000000000	#NUM!	17	1660964
1000000	1000000000000	1000000000000000000	#NUM!	20	19931569
10000000	100000000000000	10000000000000000000	#NUM!	23	232534967
100000000	10000000000000000	1E+24	#NUM!	27	2657542476
1000000000	100000000000000000	1E+27	#NUM!	30	29897352854
10000000000	10000000000000000000	1E+30	#NUM!	33	332192809489
100000000000	1E+22	1E+33	#NUM!	37	3654120904376
1000000000000	1E+24	1E+36	#NUM!	40	39863137138648
10000000000000	1E+26	1E+39	#NUM!	43	431850652335357
100000000000000	1E+28	1E+42	#NUM!	47	4650699332842310
1000000000000000	1E+30	1E+45	#NUM!	50	49828921423310400
10000000000000000	1E+32	1E+48	#NUM!	53	531508495181978000
100000000000000000	1E+34	1E+51	#NUM!	56	5647277761308520000
1000000000000000000	1E+36	1E+54	#NUM!	60	59794705707972500000
10000000000000000000	1E+38	1E+57	#NUM!	63	631166338028599000000
100000000000000000000	1E+40	1E+60	#NUM!	66	6643856189774730000000
1000000000000000000000	1E+42	1E+63	#NUM!	70	69760489992634600000000
10000000000000000000000	1E+44	1E+66	#NUM!	73	730824180875220000000000
100000000000000000000000	1E+46	1E+69	#NUM!	76	764043461824093000000000
1000000000000000000000000	1E+48	1E+72	#NUM!	80	797262742772967000000000000
10000000000000000000000000	1E+50	1E+75	#NUM!	83	830482023721841000000000000
100000000000000000000000000	1E+52	1E+78	#NUM!	86	863701304670714000000000000
1000000000000000000000000000	1E+54	1E+81	#NUM!	90	896920585619588000000000000
10000000000000000000000000000	1E+56	1E+84	#NUM!	93	930139866568462000000000000000
100000000000000000000000000000	1E+58	1E+87	#NUM!	96	963359147517335000000000000000

Sekvensielt søk (Sequential eller linear search)

Theory Book IT Grunnkurs

Algorithms

Kapittel 5.4

Sekvensielle søk

- Sekvensielt søk, er algoritmer som søker igjennom en liste for en bestemt verdi ved å sjekke hvert eneste element i rekkefølge inntil verdien er funnet.
- Sekvensielt søk er den enkleste søkalgoritmen som bruker en brute-force tilnærming.
- Fordel med sekvensiell søk er at den fungerer på en usortert liste.

Oppgave: Sekvensielt søk

- Skriv Python-koden for å gjøre et sekvensielt søk:
  - Lag en funksjon som tar inn ei *liste* og en variabel *item* som spesifiserer hva det søkes etter
  - Gå igjennom lista og undersøk om hvert element i lista er lik variabelen *item*.
  - Hvis man finner et element i lista som har samme verdi som *item*, så skal funksjonen returnere *True*
  - Hvis ikke skal den returnere *False*

seq_search.py

Om sekvensielt søk

- Hvordan kan vi gjøre algoritmen mer effektiv?
  - Stoppe når vi finner det vi leter etter.
- Hva er best-case scenario for et sekvensielt søk?
  - Finner første element:
- Hva er worst-case scenario for et sekvensielt søk?
  - Finner siste element:
- Hva er tidsbruken for sekvensielt søk?
  - Tidsbruk proporsjonal med antall elementer i lista: $\Theta(n)$

Innstikksortering (insertion sort)

Theory Book IT Grunnkurs

Algorithms

Kapittel 5.4

Innstikksortering

- Enkel sorteringsalgoritme som sorterer ei liste, element for element
- Ganske enkel å programmere
- Effektiv for korte lister
- Kan sortere alle lister som de er
- Algoritmen minner mye om hvordan mennesker tenker, som f.eks. sortering av en kortstokk

Illustrasjon av innstikksortering

6 5 3 1 8 7 2 4

Oppgave: Insertion sort

- Lag funksjonen insertion_sort som tar inn ei liste og returnerer en sortert liste ved hjelp av psaudokoden vist under.
- Psaudokoden antar at første element i lista har indeks 0.

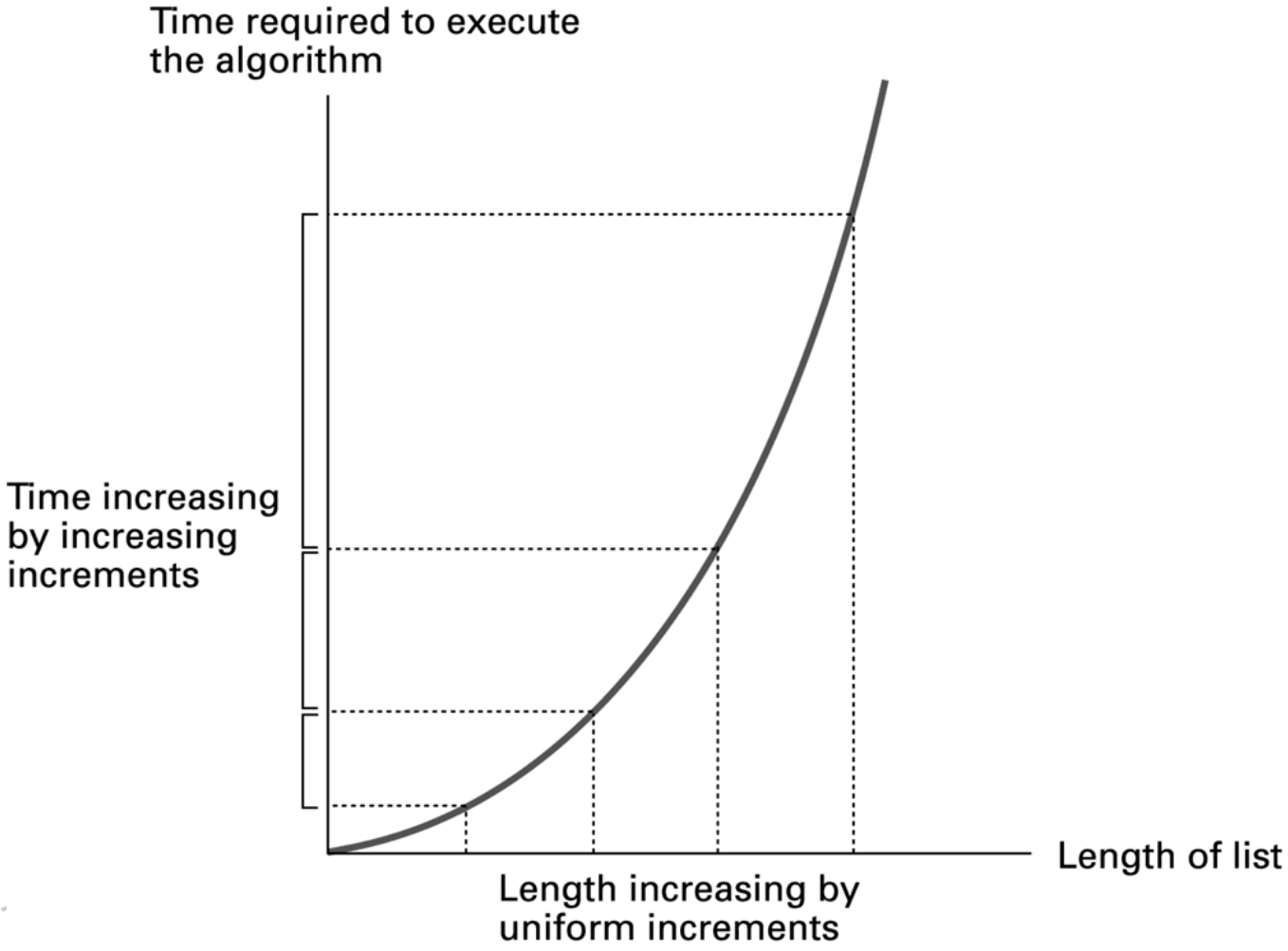
```
La i gå fra 1 til lengden av lista – 1:
  La element få verdien liste[i]
  La hull få verdien i
  Så lenge hull>0 og liste[hull-1] > element
       La liste[hull] få verdien liste[hull-1]
       La hull få verdien hull – 1
  liste[hull] = element
```

insertion_sort.py

Karakteristikker av innstikkssorteringsalgoritmen

- Hva ytelsen på denne algoritmen i best-case?
  - Best-case er en nesten sortert liste der bare ett element må flyttes en plass.
  - Fører til n sammenlikninger og Θ(1) ombytninger
- Hva er ytelsen på denne algoritmen i worst-case?
  - Worst-case er at alle elementene må byttes om.
  - Fører til $\Theta(n^2)$ sammenlikninger og ombytninger.
- Hva er en god egenskap med algoritmen?
  - Krever lite ekstra plass i minnet:
    - Lista med elementer
    - En variabel for å ta vare på den som skal byttes ut.

Graf over worst-case analysen av *innstikksortering (insert sort)*



Rekursjon

Starting out with Pyton:
Chapter 12 Recursion

Rekursjon

- Rekursiv funksjon kalles det når en funksjon kaller seg selv.
- Rekursive funksjoner brukes ofte i algoritmer for å lage elegante løsninger på problemer.
- For at en rekursiv funksjon ikke skal fortsette i det uendelige, er det viktig at argumentet til funksjonen endres slik at den en gang får en stopp verdi.
- Vi ser på et eksempel.

rekursjon.py

Rekursjon til å kalkulere fakultet

- Fakultet til et tall beregnes på følgende måte:
  - Hvis n=0 så er n!=1
  - Hvis n>0 så er n!=1 x 2 x 3 x … x n
- For å lage en algoritme for fakultet må:
  - Først finne delen som ikke er rekursiv (som ikke skal kalles på nytt):
    - Hvis n = 0 så er n!=1 (denne er ikke avhengig av tidligere ledd)
  - Resten er den rekursive delen som stegvis kan beskrives:
    - Hvis n>0 så er fakultet(n) = n x fakultet(n-1)
  - Dette er alt vi trenger for å lage en rekursiv funksjon for å beregne fakultet

fakultet.py

Binærsøk
(Binary Search Algorithm)

Theory Book IT Grunnkurs

Kapittel: Algorithm, 5.5

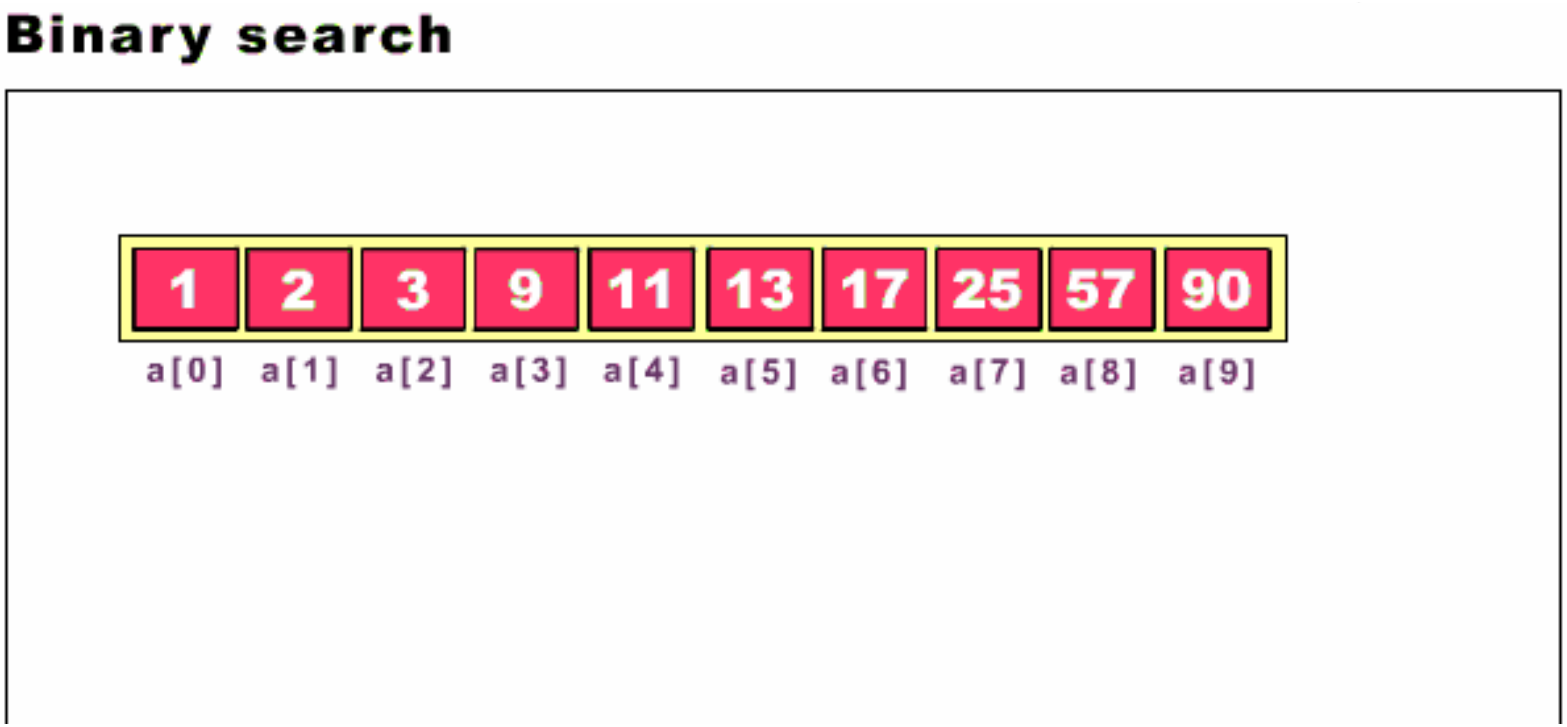
NTNU
Det skapende universitet

Binærsøkealgoritmen

- Binærsøkealgoritmen fungere på samme måte som vi kan spille gjettespillet fra uke 39.
- Vi tar en titt…

gjettespill.py

Illustrasjon av binærsøk



Algoritme for binærsøk

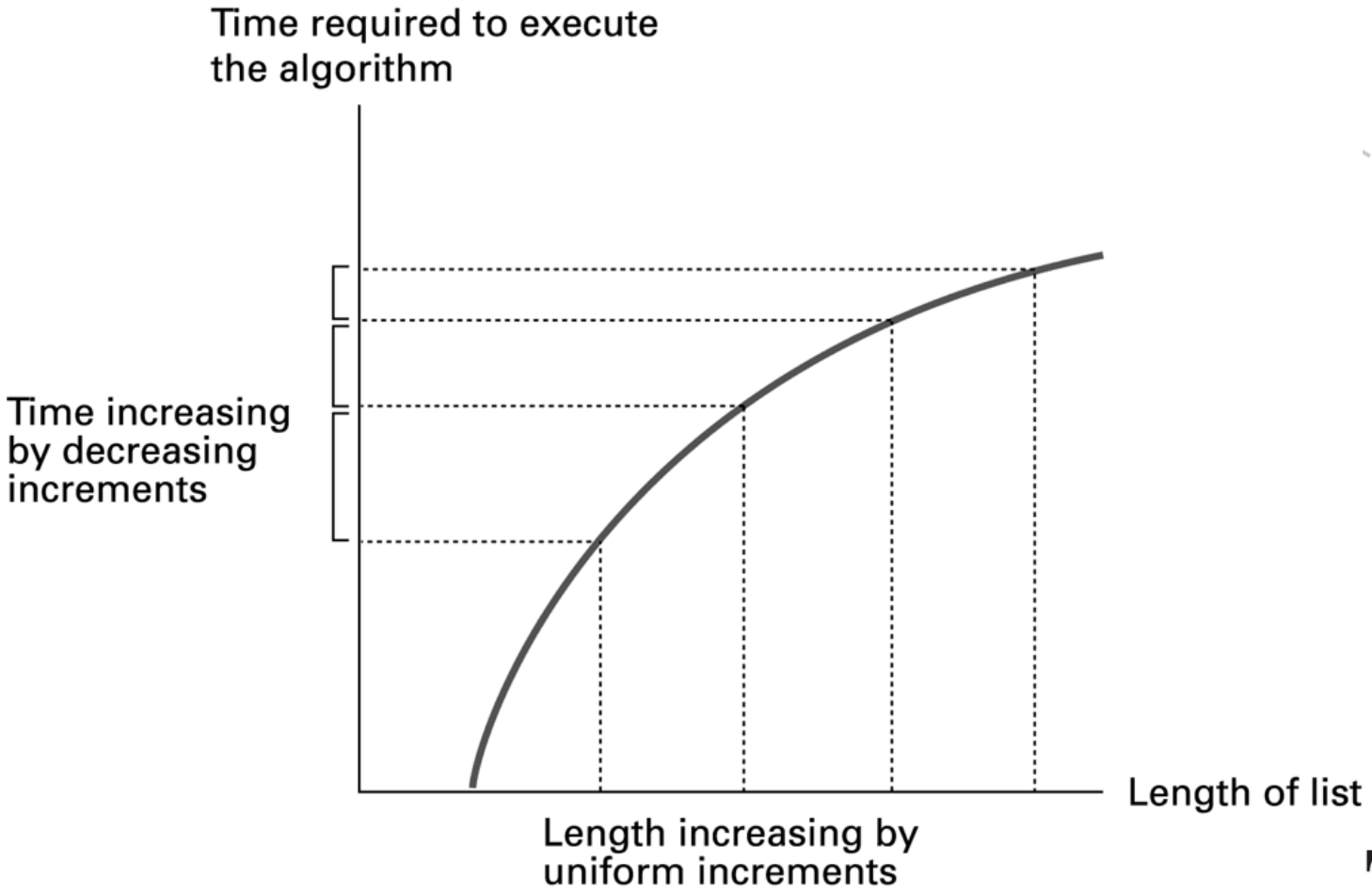
```
bin_search(liste, verdi, min, max)
  hvis max < min Returner ikke funnet verdi
  hvis ikke
    midtpunkt = midtpunkt(min,max)
    hvis verdi < liste[midtpunkt]
      returner bin_search(liste,verdi,min,midtpunkt-1)
    hvis verdi > liste[midtpunkt]
      returner bin_search(liste,verdi,midtpunkt+1,max)
    hvis ikke
      returner midtpunkt
```

binary_search.py

Karakteristikker av binærsøk

- Hva er best-case?
  - Treffer på midten første gang
  - Tidsbruk: Θ(1)
- Hva er worst-case?
  - Halverer lista helt til det er ett element igjen
  - Tidsbruk: Θ(log n)
- Kan sekvensielt søk være raskere enn binærsøk?
  - Ja hvis sekvensielt søk treffer tidlig i lista
- Begrensninger på binærsøk:
  - Lista må være sortert (pre-prosessert)

Graf over worst-case analyse av binært søk



Oppsummering

- Sekvensielle søk:
  - Enkel å implementere og fungerer på usorterte lister
  - Brute-force
  - Worst-case: $\Theta(n)$
- Innstikk sortering:
  - Enkel å implementere og fungerer som stokking av kortstokk
  - Worst-case: $\Theta(n^2)$
- Binærsøk:
  - Kan implementeres rekursivt og fungerer kun på sorterte lister
  - Worst-case: $\Theta(\log n)$