



NTNU

Det skapende universitet

TDT4109 Informasjonsteknologi grunnkurs: **Tema: Algoritmer i praksis**

Børge Haugset

www.ntnu.no

1

2

Læringsmål og pensum

- Mål
 - Lære å forstå og kunne programmere algoritmer for søk og sortering.
 - Lære å forstå og kunne bruke rekursjon.



NTNU
Det skapende universitet

www.ntnu.no

2

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	1000000000000000000	1E+27	#NUM!	30	29897352854
10000000000	100000000000000000000	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	465069932842310
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	7640434618240930000000000
1000000000000000000000000	1E+48	1E+72	#NUM!	80	79726274277296700000000000
10000000000000000000000000	1E+50	1E+75	#NUM!	83	8304820237218410000000000000
100000000000000000000000000	1E+52	1E+78	#NUM!	86	86370130467071400000000000000
1000000000000000000000000000	1E+54	1E+81	#NUM!	90	896920585619588000000000000000
10000000000000000000000000000	1E+56	1E+84	#NUM!	93	9301398665684620000000000000000
100000000000000000000000000000	1E+58	1E+87	#NUM!	96	96335914751733500000000000000000

Sekvensielt søk (Sequential eller linear search)

Theory Book IT Grunnkurs
Algorithms
Kapittel 5.4

Sekvensielle søk

- Problemstilling: Finn element i en sekvens
 - Vi har en *sekvens* av elementer, som vi ikke vet noe om (rekkefølge, eller hvilke mulige verdier)
 - Ett *element* vi vil vite om er med eller ikke i *sekvens*
 - Variant: Finn indeksen til første forekomst til elementet i sekvensen
- Hva er en algoritme (detaljert trinn for trinn beskrivelse av oppskrift) for å gjøre dette?

Oppgave: Sekvensielt søk



Skriv Python-koden for å gjøre et sekvensielt søk:

- Lag en funksjon som tar inn en *sekvens* og en variabel *element* 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*
- Du skal *ikke* bruke fancy ting som 'if element in sekvens' – poenget er å ikke gjøre det.

Om sekvensielt søk

- *Brute force*: Sjekker alle verdier inntil ønsket verdi er funnet.
- Fordeler:
 - Enkel logikk, lett å implementere riktig
 - Fungerer på vilkårlig sekvens (usortert), trenger bare å iterere gjennom alle elementene.
- Ulemper?
- 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 i gjennomsnitt?
 - I gjennomsnitt $n/2$, tester, så tidsbruk proporsjonal med antall elementer i lista: $\Theta(n)$
- Når er dette et problem?

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 (wikimedia)

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 pseudokoden vist under.
- Pseudokoden 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

Karakteristikk av innstikkssorteringsalgoritmen

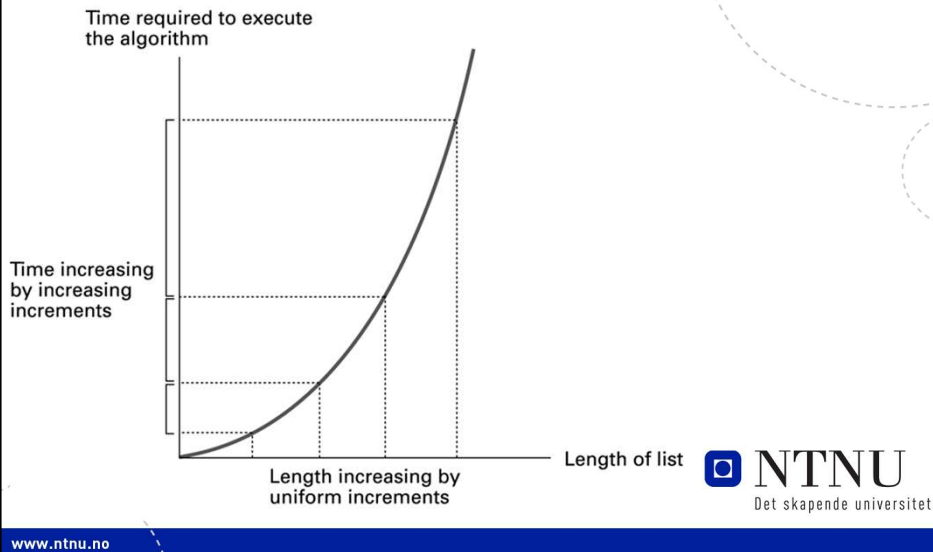
- Hva ytelsen på denne algoritmen i best-case?
 - Best-case er en nesten sortert liste der bare ett element må flyttes en plass.
 - Fører til n sammenlikninger og $\Theta(1)$ ombytninger
- Hva er ytelsen på denne algoritmen i worst-case?
 - Worst-case er at alle elementene må byttes om.
 - Fører til $\Theta(n^2)$ 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.

Noen egenskaper til innstikks-sortering

- **Enkel:** Lett å implementere, minder sjangse for feil
- **Effektiv på små lister**
- **Adaptiv:** Effektiv når data allerede er nesten sortert rekkefølge
- **Stabil:** To elementer som blir sammenlignet som like i opprinnelig liste, kommer i samme innbyrdes rekkefølge
- **In-place:** Bruker eksisterende liste pluss en variabel.
- **Online:** Kan sortere elementer fortløpende som de kommer.

15

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

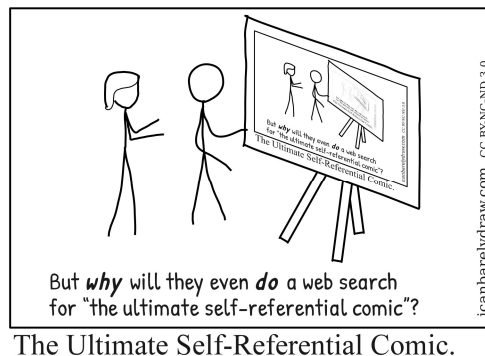


15

16

Hva skjer hvis en funksjon kaller seg selv?

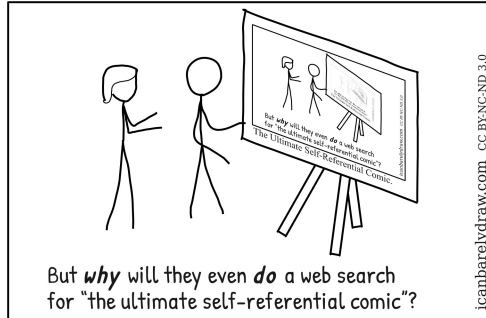
- Er det lov i Python?
- Blir det ikke kluss med variabelnavn?



16

Hva skjer hvis en funksjon kaller seg selv?

- Er det lov i Python?
- Blir det ikke kluss med variabelnavn?
- Det er lov!
- Nytt sett med *lokale* variable med samme navn
- Som andre funksjonskall, bortsett fra... Hva må vi passe på?



The Ultimate Self-Referential Comic.

Rekursjon

Starting out with Python:
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 til å kalkulere fakultet

- Fakultet til et tall beregnes på følgende måte:
 - Hvis $n=0$ så er $n!=1$
 - Hvis $n>0$ så er $n!=1 \times 2 \times 3 \times \dots \times n$
- 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 $\text{fakultet}(n) = n \times \text{fakultet}(n-1)$
 - Dette er alt vi trenger for å lage en rekursiv funksjon for å beregne fakultet

Rekursjon for kalkulering av fakultet

Hva skjer når vi kaller den rekursive funksjonen for beregning av fakultet med 4 som argument:

```
return 4 * fac(3)
      return 3 * fac(2)
            return 2 * fac(1)
                  return 1 * fac(0)
                        1
```

- 1 vil være den første returnerte verdien, men denne returneres til fac(1).
- Resultatet av fac(1) vil bli returnert til fac(2) osv.
- Returverdien av funksjonen er IKKE returverdien for hele uttrykket

Oppgave: fakultet



- Lag funksjonen fakultet som tar inn et tall og beregner fakultet til tallet ved hjelp av rekursjon.
- Hvis man beregner f.eks. $3! = 3 \cdot 2 \cdot 1$. $4! = 3 \cdot 2 \cdot 1$.
- Mer generelt blir $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$
- Pseudokode for funksjonen fakultet med parameter n

```
Hvis n er 0:
    returner 1
Hvis ikke:
    returner n multiplisert med kall av funksjonen med n-1
```

fakultet.py

Rekursjon gir ofte elegante løsninger

Vi trenger bare å formulere og kode to ting:

- Base-case(ne): Disse er ofte veldig enkle:
 - F.eks. $0! = 1$
- Reduksjon til en mindre case.
- Python vil sette opp nye variable i det rekursive kallet som får det nye (lavere) verdiene sendt som parametre.
- NB! Hvor muterbare data som lister, må vi passe på om det skjer endringer på listen eller ikke, og om det er ok eller ikke.

Binærsøk (Binary Search Algorithm)

Theory Book IT Grunnkurs
Kapittel: Algorithm, 5.5

Binærsøkealgoritmen

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

Illustrasjon av binærsøk (1)

1	2	3	9	11	13	17	25	57	90	57
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	

Suppose the data that is to be searched is **57**

Illustrasjon av binærsøk (2)



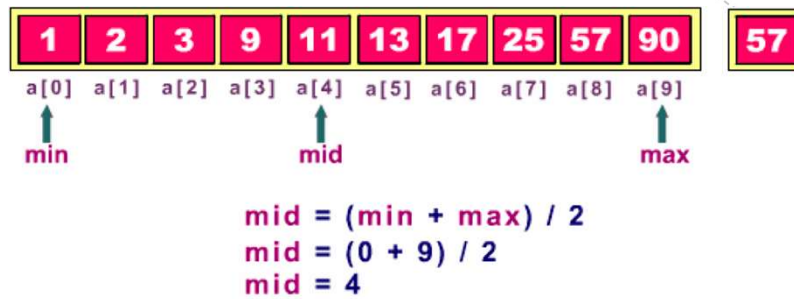
The upper and lower bounds are **min** and **max** respectively

Illustrasjon av binærsøk (3)

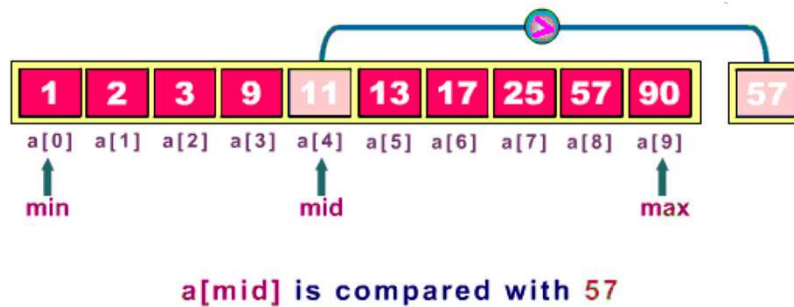


Calculate **mid** (index of variable with which **57** is to be compared)

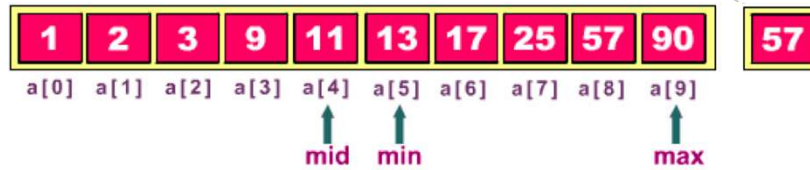
Illustrasjon av binærsøk (4)



Illustrasjon av binærsøk (5)

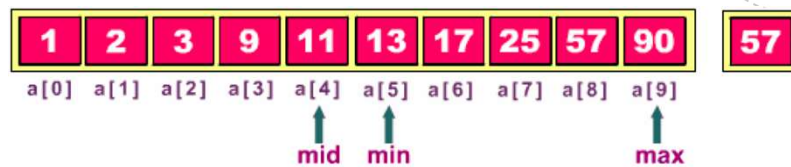


Illustrasjon av binærsøk (6)



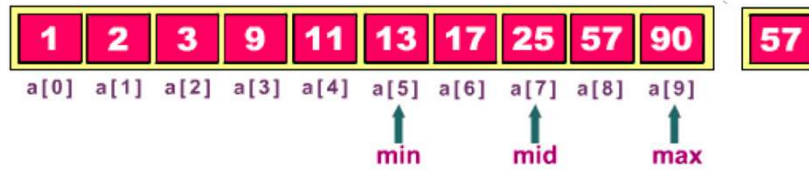
As 11 is less than 57, lower bound min is pointed to $a[5]$

Illustrasjon av binærsøk (7)



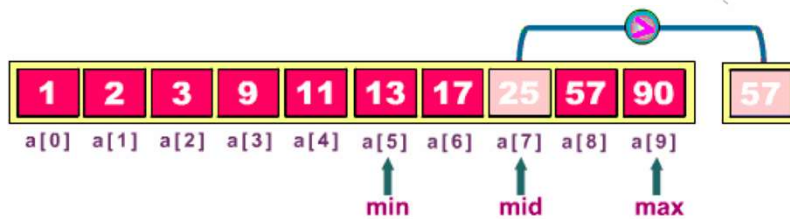
Again mid is calculated

Illustrasjon av binærsøk (8)



$$\begin{aligned} \text{mid} &= (\text{min} + \text{max}) / 2 \\ \text{mid} &= (5 + 9) / 2 \\ \text{mid} &= 7 \end{aligned}$$

Illustrasjon av binærsøk (9)



$a[\text{mid}]$ is compared with 57

Illustrasjon av binærsøk (10)



As **25** is less than **57**, lower bound **min** is pointed to **$a[8]$**

Illustrasjon av binærsøk (11)



Again **mid** is calculated

Illustrasjon av binærsøk (12)



$$\begin{aligned} mid &= (min + max) / 2 \\ mid &= (8 + 9) / 2 \\ mid &= 8 \end{aligned}$$

Illustrasjon av binærsøk (13)



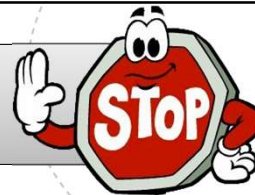
$a[mid]$ is compared with 57

Illustrasjon av binærsøk (14)



Search is successful as **57** is found

Oppgave: binærsøk



Prøv å oversett pseudokoden for binærsøk til Python:

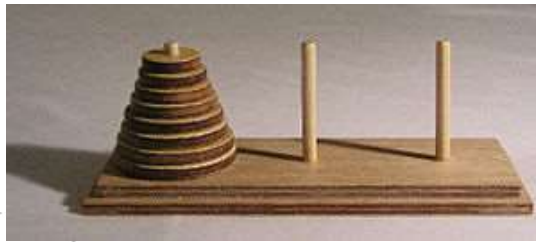
```
bin_search(liste, verdi, min, max)
    hvis max < min Returner ikke funnet verdi
    hvis ikke
        midtpunkt = midtpunkt(min,max)
        hvis verdi < liste[midtpunkt]
            returner bin_search(liste,verdi,min,midtpunkt-1)
        hvis verdi > liste[midtpunkt]
            returner bin_search(liste,verdi,midtpunkt+1,max)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

Husk:

Når vi designer en rekursiv algoritme, så trenger vi ikke tenke på alle trinnene. Det vi må fokusere på, er

- Håndtere «base-case» (som ikke kan reduseres)
- Formulere en løsning basert på løsningen til et mindre problem (typisk lavere verdi av n)

Eksempel: Tårnene i Hanoi (se i boka, Wikipedia,

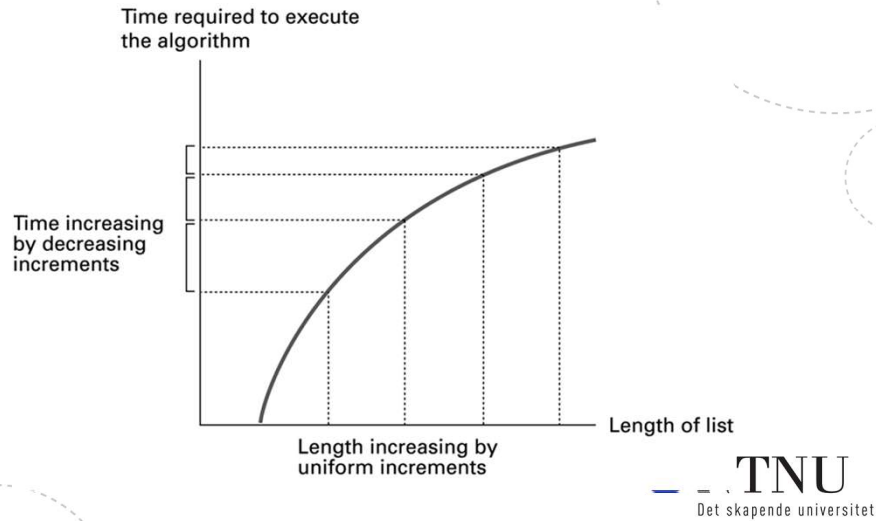


Karakteristikk av binærsøk

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

43

Graf over worst-case analyse av binært søk



43

44

Oppsummering

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

44