Föreläsning
Datastrukturer (DAT036)

Nils Anders Danielsson

2013-12-04

Idag

- Frågor? Är något oklart inför tentan?
- Ny läsårsindelning.
- Sammanfattning.

Ny läsårsindelning

- https://www.chalmers.se/insidan/sites/ppd/utvecklingprojekt/downloadFile/attachedFile_1_f0/C_2013-1378_beslut_m_bilaga.pdf.
- Har ni några förslag på hur den extra tiden kan användas?
- Schemalagd undervisning 7-9/1?
  Sammanfattning av kursen?
  Övningspass med gamla tentor?
- Labbdeadline(s) efter jul?

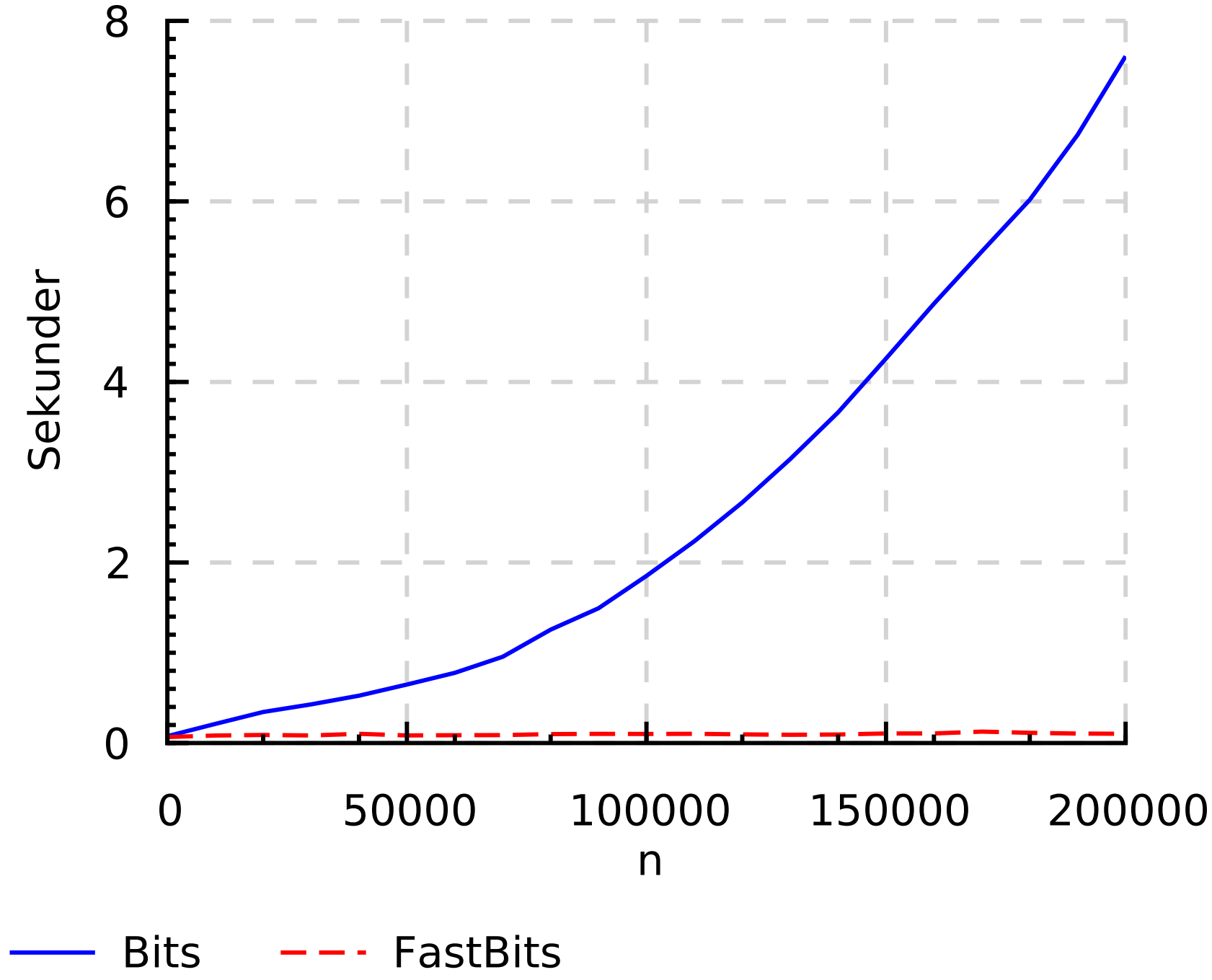
Exempel från föreläsning 1

Dåligt val av datastruktur

```
public class Bits {

    public static void main(String[] args) {

        int n = Integer.parseInt(args[0]);

        String bits = new String();

        for (int i = 0; i < n; i++) {
            bits += i & 1;
        }

        System.out.print(bits);

    }
}
```

Bättre val av datastruktur

```
public class FastBits {

    public static void main(String[] args) {

        int n = Integer.parseInt(args[0]);

        StringBuffer bits = new StringBuffer();

        for (int i = 0; i < n; i++) {
            bits.append(i & 1);
        }

        System.out.print(bits);

    }
}
```


8
6
4
2
0
Sekunder
0
50000
100000
150000
200000
n
Bits
FastBits

Hur har det gått?

Jag hoppas att ni nu (snart) har verktyg för att:

- Snabba upp era program på motsvarande sätt.
- Förutse ungefär hur snabbt era (enklare) program kommer att gå.

Ta något lite mer komplicerat program som ni har skrivit i en annan kurs. Räcker verktygen ni fått i den här kursen till för att kunna analysera det programmet? Om inte, vad saknas?

- Svara kortfattat i Pingo.

Sammanfattning

Modeller

Vi har använt förenklade modeller av verkligheten för att analysera våra datastrukturer och algoritmer.

- Uniform kostnadsmodell.
- Logaritmisk kostnadsmodell.

Modellerna har begränsningar.

Ordonotation

Ordonotation gör det möjligt att dölja irrelevanta – men även relevanta – detaljer.

- Definition (O, Ω, Θ).
- Regler.

Tidskomplexitetsanalys

Vi har analyserat många algoritmers tidskomplexitet.

En enkel analys kan se ut så här:

- Vi känner till tidskomplexiteten hos vissa standardkomponenter.
- Vi räknar sedan hur många gånger standardkomponenterna körs.
  - ”En gång per varv i loopen.”
  - ”En gång per nod.”
  - ”En gång per kant.”

Pseudokod

- Blandning av programmeringsspråk, matematisk notation och naturligt språk.
- Mål: Fokusera på det viktiga, undvik onödiga detaljer.
- Ibland kan det vara bra med fler detaljer, ibland färre.

Abstrakt datatyp/datastruktur

- Abstrakt datatyp: Matematisk abstraktion.
- Datastruktur: Implementation.

Några (klasser av) abstrakta datatyper

- Listor.
- Stackar.
- FIFO-köer.
- Prioritetsköer.
- Mängder.
- Disjunkta mängder.
- Avbildningar (”maps”).
- Grafer.

Tips: Återanvänd kod

- Ofta behöver man inte implementera datastrukturer själv.
- Om man någon gång behöver det kan man ändå dra nytta av existerande standarddatastrukturer.

2012/08:2

Uppgiften är att konstruera en datastruktur som representerar ”dubbelriktade maps” (bijektioner mellan ändliga mängder):
`insert(`s`,`t`)`, `target(`s`)`, `source(`t`)`.

```
public class Bijection<S, T> {

  private Map<S, T> to; private Map<T, S> from;

  public Bijection() { to   = new HashMap<S, T>();
                       from = new HashMap<T, S>(); }

  public void insert(S s, T t) {
    if (to.containsKey(s) || from.containsKey(t)) {
      throw new IllegalArgumentException();
    }
    to.put(s, t); from.put(t, s);
  }

  public T target(S s) { return to.get(s); }

  public S source(T t) { return from.get(t); }
}
```

Listor

ADT med olika implementationer.

Länkade listor:

- Långsam indexering.
- Snabb insättning (exkl sökning) i mitten.
- Vaktposter? Enkellänkade, dubbellänkade?
- Pekarjonglering. Testa! Invarianter.

Dynamiska arrayer:

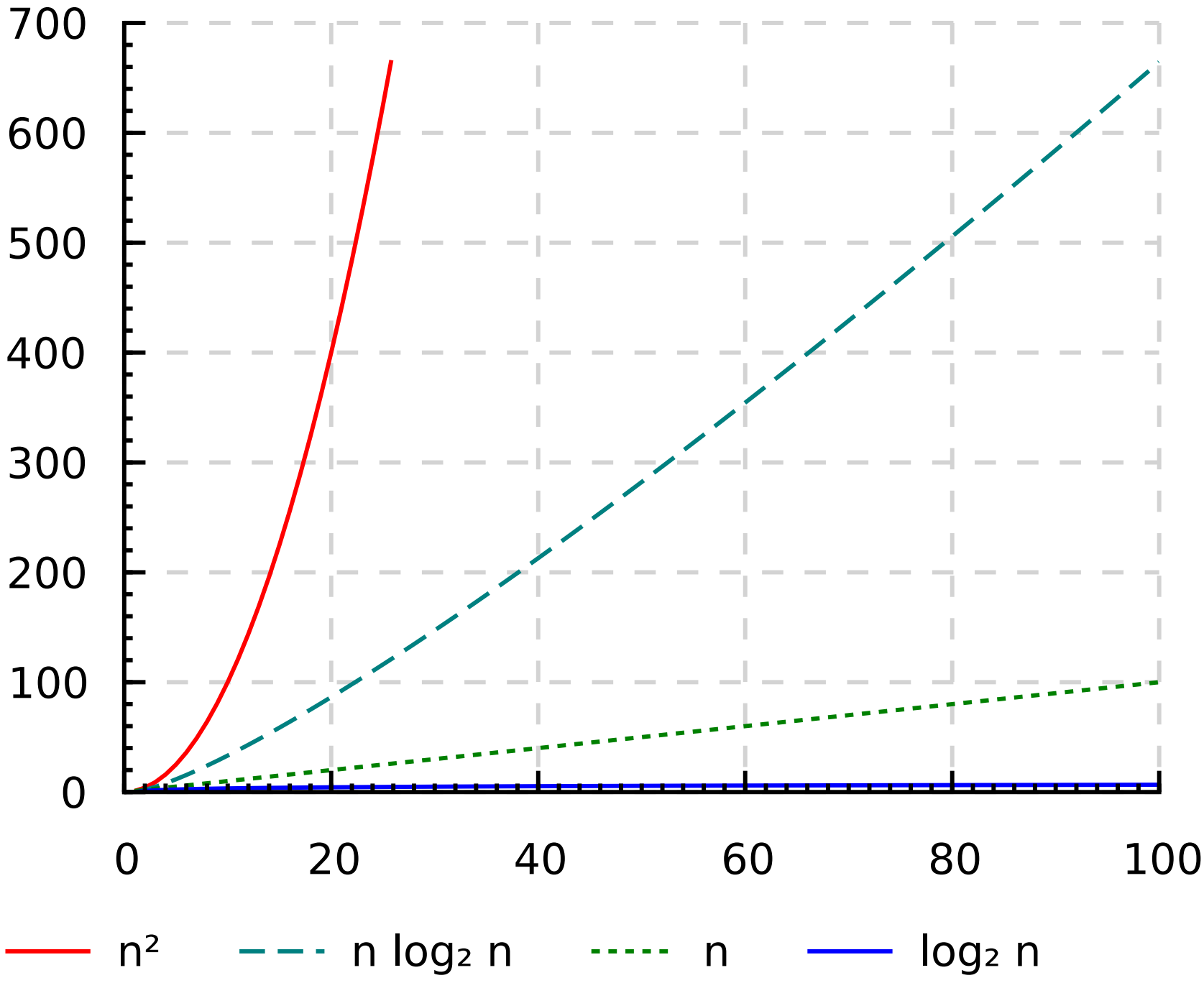
- Snabb indexering.
- Snabb insättning sist.
- Amorterad tidskomplexitet (bokföringsmetoden).

Persistenta datastrukturer

- Efter förändring finns gamla kopian kvar.
- `sort :: [Integer] -> [Integer]`, inte `sort :: [Integer] -> void`.
- Nackdel:
  Vissa operationer kan vara mindre effektiva.
- Fördel: Kan vara lättare att förstå.
- Fördel: Kan vara lättare att testa.

Sortering

- Undre gräns: $\Omega(n \log n)$.
- Insättningssortering: $O(n^2)$.
- Mergesort: $O(n \log n)$.
- Heapsort: $O(n \log n)$.
- Splaysort: $O(n \log n)$.
- Quicksort: $O(n \log n)$ (medel).
- Hinksortering: $O(N + n)$ (N: antalet hinkar).
- Radixsortering: $O(d(N + n))$
  (N: "talbasen", d: största antalet "siffror").
- Stabil? Adaptiv? In-place?


700
600
500
400
300
200
100
0
0
20
40
60
80
100
n²
n log₂ n
n
log₂ n

Stackar, FIFO-köer

- Enkla implementationer med (olika sorters) listor.
- Cirkulära arrayer.
- Två listor. Persistent. $O(1)$ (amorterat) vid enkeltrådad användning.

Prioritetsköer

- Binära heapar:
  - Heapordnade kompletta binära träd.
  - Trädet representeras ofta av en array.
  - Bubbla upp/ned.
- Braunheapar:
  - Heapordnade binära träd.
  - $0 \leq |\text{höger barn}| - |\text{vänster barn}| \leq 1$.
- Binomialheapar:
  - Effektiv `merge`.
  - Baserade på positionellt talsystem: $5 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2$.
  - Amorterad tidskomplexitet.

Mängder, avbildningar

Hashtabeller:

- Hashfunktioner.
- Kollisioner.
- Kedjning, öppen adressering.

Mängder, avbildningar

Olika sorters sökträd:

- Obalanserade binära sökträd:
  - Långsamma vid insättning av sorterad sekvens.
- AVL-träd:
  - Höjdbalanserade.
  - Rotationer.
- Splayträd:
  - Ej balanserade, ”splaying”.
  - Amorterad tidskomplexitet.
- Rödsvarta träd, B-träd.

Mängder, avbildningar

Skipplistor:

- Insättning: Randomiserad algoritm.
- Förväntad tidskomplexitet.

Grafer

- ADT: $G = (V, E)$.
- Datastrukturer: Grannlistor, grannmatriser.
- Algoritmer:
  - Topologisk sortering.
  - Kortaste vägen:
    Bredden först-sökning, Dijkstra.
  - Minsta uppspännande träd: Prim, Kruskal
    (inkl disjunkt mängd-datastruktur).
  - Djupet först-sökning, uppspännande skog.
  - Starkt sammanhängande komponenter.

Till sist

- Övningstillfället imorgon:
  Berätta vad ni vill jobba med nästa vecka.
- Svara gärna på kursenkäten.

Lycka till!