

# Heapsort, Mergesort og nedre grænse for sortering

Mads Ωhm Larsen

6. april 2008

## 1 Hobsortering

### 1.1 Definition

En binær hob er et næsten komplet<sup>1</sup> binært træ. Hver knude i træet er et element i en liste. I en **max-heap** skal alle forældre være større end børnene.

### 1.2 Operationer

- **MAX-HEAPIFY** —  $O(\lg n)$  – Gør hobben til en max hob
- **BUILD-MAX-HEAP** —  $O(n)$  – Bygger hobben
- **HEAPSORT** —  $O(n \lg n)$  – Sorter hobben “in place”

### 1.3 Bygning og sortering

Vi bygger en max-hob ved at kigge på det binære træ nedefra. Tjekker for hver knude om det er større end begge dets børn. Er det ikke, bliver det skiftet ud med det største af de to. Derefter bliver det tjekket igen (**MAX-HEAPIFY**). Vi sammenligner altså  $O(n)$ , og derfor får vi en “bygningstid” på  $O(n)$ .

Vi sortere ved at skifte det største tal (roden) ud med det sidste. Herefter smider vi det ind bagerst i en liste, og sletter det fra hoben. Vi kører så **MAX-HEAPIFY** og starter forfra. Dette tager  $O(n \lg n)$  fordi vi laver  $n - 1$  kald til **MAX-HEAPIFY**, der jo tager  $O(\lg n)$ .

## 2 Flettesortering

### 2.1 Definition på flettesortering

Merge-sort er nok den mest basale *del-og-hersk* algoritme. Den deler en liste i to lige store dele og løser problemet rekursivt. Dvs. den deler til problemet er så småt at det er trivielt at løse (1 element tilbage i listen). Så fletter den igen ved at sammelige elementerne.

---

<sup>1</sup>Den er fyldt helt ud, måske bortset fra det nederste lag

## 2.2 Tidskompleksitet

Kan skrives som en ligning:

$$T(n) = \begin{cases} \Theta(1) = 0 & \text{hvis } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{hvis } n > 1 \end{cases}$$

Ved brug af *Master Theorem* ved vi at hvis vi har  $T(n) = aT(n/b) + f(n)$  så er det for denne ligning lig

$$\begin{aligned} f(n) = \Theta(n^{\log_b a}) &\Rightarrow T(n) = \Theta(n^{\log_b a} \lg n) \\ &\Rightarrow \Theta(n^{\log_2 2} \lg n) = \Theta(n \lg n) \end{aligned}$$

Selvfølgelig behøver vi ikke *Master Theorem*, men kan klare os med substitutionsmetoden: Vi starter med et gæt, som man kan komme fremtil ved hjælp af et rekursionstræ:  $T(n) \leq cn \lg n$ . Herefter kan vi gå i gang med at bevise det:

$$\begin{aligned} T(n) &\leq 2 \left( \frac{cn \lg \left( \frac{n}{2} \right)}{2} \right) + dn \\ &= cn \lg \left( \frac{n}{2} \right) + dn \\ &= cn \lg n - cn \lg 2 + dn \\ &= cn \lg n - cn + dn \\ &\leq cn \lg n \quad \text{for } c \geq d \end{aligned}$$

Induktion kræver nu et basistilfælde for hvilket det vi har fundet gælder. Man vælger selvfølgelig altid det letteste, og vi kigger derfor på  $T(1)$ .  $T(1)$  er i følge det vi har fundet  $c \cdot 1 \lg 1 = 0$ , dette er også tilfældet ifølge vores rekursionsligning da  $T(1) = 0$ .

## 3 Nedre grænse for sortering

Vi vil bevise at der er en nedre grænse på sorterings algoritme der bruger sammenligninger. Hvis vi har et beslutningstræ med højde  $h$  og  $l$  blade for  $n$  elementer. Med  $n$  elementer kan der forekomme  $n!$  permutationer, altså vil der være  $n!$  blade på vores beslutningstræ. Vi ved desuden at der maksimum kan være  $2^h$  blade i et binært træ. Dette betyder at

$$\begin{aligned} n! &\leq l \leq 2^h \\ \lg(n!) &\leq \lg(l) \leq h \\ \Omega(n \lg n) &\leq h \end{aligned}$$