TDDI16 Datastrukturer och algoritmer

23 Augusti 2010, kl 8-12

1. [2p]

Analysera tidskomplexiteten för följande program. Ge en exakt lösning. Antag att n är antalet element i listan.

```
procedure Sums( A[1..n]: list of integers);
var i,j, value : integer;
var B[1..n] : list of integers;
begin
    for i := 1 to n do begin
        B[i] := 0;
    end
    for i := 1 to n do begin
        for j:= i to n do begin
            B[i] := B[i] + A[j];
        end
    end
end
```

2. [2 + 2 = 4p]

(a) Definiera ADT *Queue* (modell + operationerna - förklara vad operationerna gör).

(b) Beskriv implementationen av ADT *Queue* med circulär array. Beskriv datastrukturen. Visa hur en tom kö ser ut. Visa hur en full kö ser ut. Ange tidskomplexitet för varje operation.

3. [2 + 2 = 4p]

(a) Skriv en algoritm i pseudokod som använder ADT *Tree* för att räkna det största antalet barn i en nod i ett givet träd. (Använd operationer så som de är definierade för ADT Tree.)

(Exempel: Antag rot A; A har 2 barn: B och C; B har 3 barn: D, E och F; C, D, E och F har inga barn. Då ska algoritmen returnera: 3.)

(b) Förklara implementationen av ADT *Tree* som använder en 'parent'-array för att lagra information om vilken nod som är förälder till en nod. Ange tidskomplexitet för varje operation för ADT *Tree* vid användning av denna implementation.

4. [2 + 1 = 3p]

(a) Antag en hashtabell av storlek 5 och nedanstående operationer. Antag stängd hashing och linjär sondering som kollisionshanteringsstrategi. Visa hur tabellen ser ut efter VARJE operation. (Operationerna körs i den givna ordningen.) Använd de givna hashvärdena h(x).

(i)	*Insert*(A,'Jeff')	h('Jeff') = 3
(ii)	*Insert*(A,'Jane')	h('Jane') = 2
(iii)	*Insert*(A,'Jack')	h('Jack') = 2
(iv)	*Delete*(A,'Jane')	h('Jane') = 2

(b) Ge en fördel med linjär sondering över kvadratisk sondering. Ge en fördel med kvadratisk sondering över linjär sondering.

5. [1 + 2 = 3p]

(a) AVL-träd

Visa hur AVL-trädet i figur 1 ser ut efter operationen *Insert*(70).

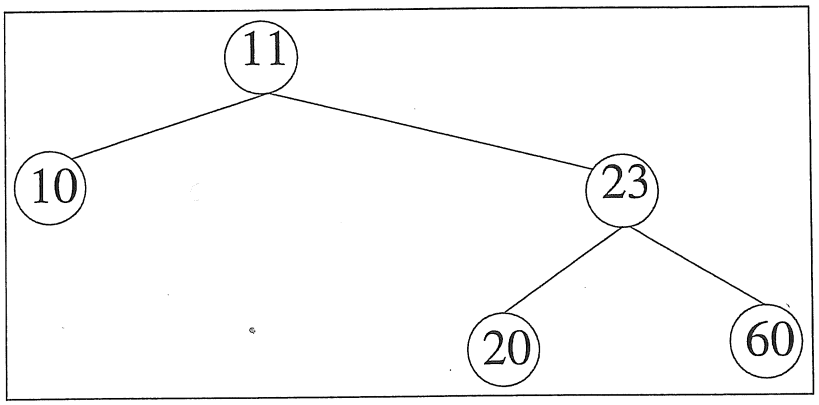

Figure 1: AVL-träd

(b) PriorityQueue

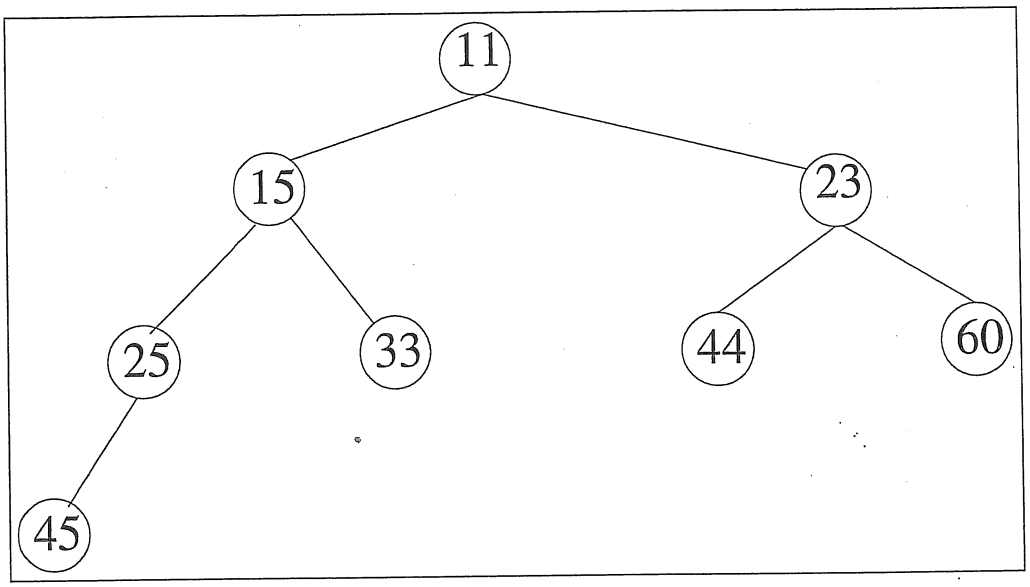

Figure 2: heap

(i) Visa hur heapen i figur 2 ser ut efter operationen $Insert(14)$.

(ii) Visa hur heapen i figur 2 ser ut efter operationen $DeleteMin$ (*utan* den tidigare $Insert$).

6. [2 + 2 + 1 + 1 = 6p]

Givet sekvensen

$$< 28, 12, 17, 24 , 33, 14, 12, 35 >$$

(a) Visa hur *MergeSort* sorterar sekvensen. Visa varje steg.

(b) Visa hur sekvensen sorteras med hjälp av 2-positions *RadixSort* med bas 10.

(c) Vilken/vilka av *QuickSort*, *MergeSort* och *HeapSort* har optimal tidskomplexitet?

(d) Är en algoritm med optimal tidskomplexitet också alltid snabbast? Förklara ditt svar.

7. [2p]

Lös följande rekursiv ekvation.

$$\begin{aligned} T(n) &= d && \text{if } n = 1 \\ &\quad 2\,T(n/2) && \text{if } n \geq 2 \end{aligned}$$