ASD - Heapsort - si Costuri

In acest capitol se prezinta un alt algoritm de sortare: HeapSort.

Heapsort foloseste o structura de date, pe care o vom numi "heap" (ansamblu), structura ce permite extragerea maximului/minimului rapid (in timp logaritmic).

Heapsort va fi un algoritm de sortare bazat pe selectia maximului/minimului, cu o performanta $\mathcal{O}(n \lg n)$.

1 Heaps

O structura de date de tip heap/ansamblu (binar) este un vector ce poate fi vazut ca un arbore binar aproape complet (1). Fiecare cheie din arbore corespunde unui element din vector. Arborele este complet pe toate nivelele cu exceptia celui mai de jos, unde exista chei incepand din stanga pana la un anumit punct.
Un vector A asociat unui ansamblu este caracterizat de 2 atribute:
- A.length (numarul elementelor din vector);
- A.heap-size (numarul de elemente din "heap") (chiar daca $A[1 \cdots A.length]$ contine numere, doar elementele din $A[1 \cdots A.heap - size]$, unde $0 \leq A.heap - size \leq A.length$), sunt elemente valide in "heap".

root = A[1];
Pentru $\forall$ indice i al unui nod, se poate calcula usor indicii parintelui sau si al descendentilor sai stang si drept:
$PARENT(i)$
    return $\lfloor i/2 \rfloor$
$LEFT(i)$
    return $2i$
$RIGHT(i)$
    return $2i + 1$

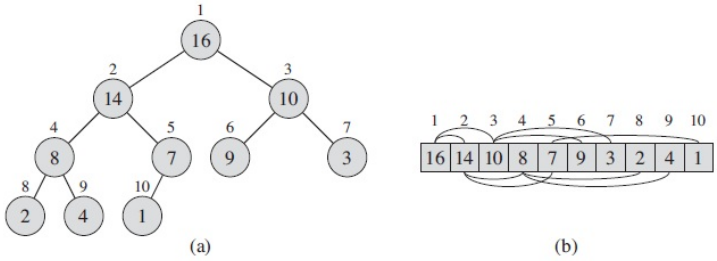

Figure 1: Un max-ansamblu vazut ca (a) arbore binar si (b) vector. In interiorul cercurilor sunt valorile fiecarui nod. Numarul de deasupra este indicele corespunzator in vector. Deasupra si sub vector sunt trasate liniile aratand relatiile tata-fiu. Parintii sunt intotdeauna la stanga descendentilor. Inaltimea arborelui este 3; nodul cu indicele 4 (avand valoarea 8) are inaltimea 1.

Sunt 2 tipuri de arbori binari partiali:
- max-heap (max-ordonat) ($\forall$ nod $i \neq$ root $\Rightarrow$ A[PARENT(i)] $\geq$ A[i] (Maximul elementelor se afla in root);
- min-heap (min-ordonat) ($\forall$ nod $i \neq$ root $\Rightarrow$ A[PARENT(i)] $\leq$ A[i] (Minimul elementelor se afla in root).
Pentru algoritmul HeapSort va fi folosita o structura de tip arbore partial max-ordonat.

Se defineste inaltimea unui nod intr-un heap, drept numarul de muchii al celei mai lungi cai directe de la nod la o frunza. Inaltimea unui ansamblu este inaltimea radacinii sale. Cum un ansamblu cu n elemente se bazeaza pe un arbore binar complet, inaltimea lui este $\Theta(\lg n)$. Se poate demonstra ca operatiile de baza pe un ansamblu ruleaza in timp cel mult proportional cu inaltimea arborelui, deci $\mathcal{O}(\lg n)$. Proceduri de baza folosite intr-un algoritm de sortare si o structura de date de tip coada cu prioritati:

- MAX-HEAPIFY: - timp $\mathcal{O}(\lg n)$ - pastreaza proprietatea de arbore max-ordonat;
- BUILD-MAX-HEAP: - timp $\mathcal{O}(n)$ - creaza un arbore max-ordonat pornind de la un vector nesortat;
- HEAPSORT: - timp $\mathcal{O}(n \lg n)$ - sorteaza un vector;

.

1.1 Pastrarea proprietatii de arbore partial max-ordonat

Procedura MAX-HEAPIFY primeste ca parametri un vector si un indice al unui element din vector. Se presupune ca arborii binari cu radacinile $LEFT(i)$, respectiv $RIGHT(i)$ sunt arbori partial max-ordonati, dar e posibil ca $A[i]$ sa fie mai mic decat descendentii sai, astfel incalcand conditia de arbore max-ordonat.

PROCEDURE MAX-HEAPIFY (A, i)
1. $l = LEFT(i)$
2. $r = RIGHT(i)$
3. **if** $l \leq A.heap - size$ and $A[l] > A[i]$
4. $\quad largest = l$
5. **else** $largest = i$
6. **if** $r \leq A.heap - size$ and $A[r] > A[largest]$
7. $\quad largest = r$
8. **if** $largest \neq i$
9. $\quad$ **interschimba(A**[i]**,A**[$largest$]**)**
10. **MAX-HEAPIFY (A, largest)**

La fiecare pas se determina maximul dintre $A[i]$, $A[LEFT(i)]$ si $A[RIGHT(i)]$. Variabila *largest* stocheaza indicele elementului maxim. Daca maximul este $A[i]$ atunci subarborele cu radacina in nodul i este deja max-ordonat si procedura se termina. Altfel, daca unul din cei doi descendenti contine maximul, $A[i]$ se interschimba cu $A[largest]$, nodul i si descendentii sai satisfacand conditia de arbore partial max-ordonat. Totusi subarborele avand drept radacina nodul indexat de variabila *largest*, poate viola conditia de max-ordonare, in consecinta, procedura MAX-HEAPIFY apelandu-se recursiv pe acest sub-arbore.

Figura 2 exemplifica apelul procedurii MAX-HEAPIFY.

1.2 Analiza costurilor

Timpul de executie a procedurii MAX-HEAPIFY pe un sub-arbore de inaltime **n** cu radacina intr-un nod **i** este dat de:

- $\Theta(1)$: costul stabilirii unei relatii intre $A[i]$, $A[LEFT(i)]$ si $A[RIGHT(i)]$;
- Timpul de rulare a procedurii pe un sub-arbore cu radacina intr-un descendent al nodului i (presupunand ca apelul recursiv are loc).

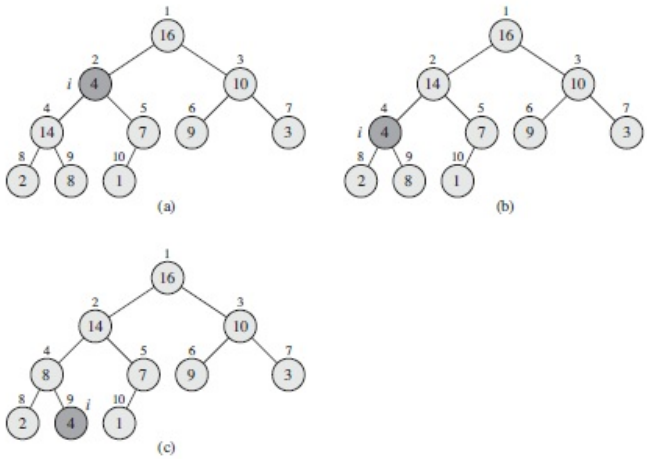

Figure 2: Apelul *MAX-HEAPIFY(A,2)*, unde $A.heap - size10$. (a) Configuratia initiala $A[2]$ violeaza conditia de arbore max-ordonat; (b) Interschimbare($A[2]$,$A[4]$): A[2] respecta conditia, dar A[4] nu, deci se apeleaza recursiv *MAX-HEAPIFY(A,4)*; (c) Interschimbare($A[4]$,$A[9]$) si apel recursiv *MAX-HEAPIFY (A,9)* - nu se mai fac alte schimbari.

Sub-arborii descendenti au inaltimea cel mult $2n/3$ (cazul cel mai defavorabil avand ultimul nivel pe jumatate plin), deci putem da urmatoarea relatie de recurenta: $T(n) \leq T(2n/3) + \Theta(1)$.

Solutia acestei recurente, obtinuta prin cazul 2 al Teoremei Master, este $T(n) = \mathcal{O}(\lg n)$. Alternativ, se poate spune ca timpul de executie a procedurii MAX-HEAPIFY pe un nod de inaltime h este de ordinul $\mathcal{O}(h)$. Figura 2 prezinta un exemplu de apel al acestei proceduri.

1.3 Construirea unui ansamblu

Se foloseste procedura MAX-HEAPIFY de jos in sus pentru a transforma un vector de lungime n intr-un ansamblu. Elementele A[($\lfloor n/2 \rfloor$+1) $\cdots$ n] sunt frunze, deci se incepe cu un Ansamblu de lungime 1. BUILD-MAX-HEAP parcurge nodurile ramase in ordine inversa, de la n/2 la 1, si apeleaza MAX-HEAPIFY

BUILD-MAX-HEAP(A)

1 A.heap-size = A.length
2 **for** i = $\lfloor A.length/2 \rfloor$ **downto** 1
3 MAX-HEAPIFY(A,i)

Figura 3 prezinta un exemplu de apel al acestei proceduri.

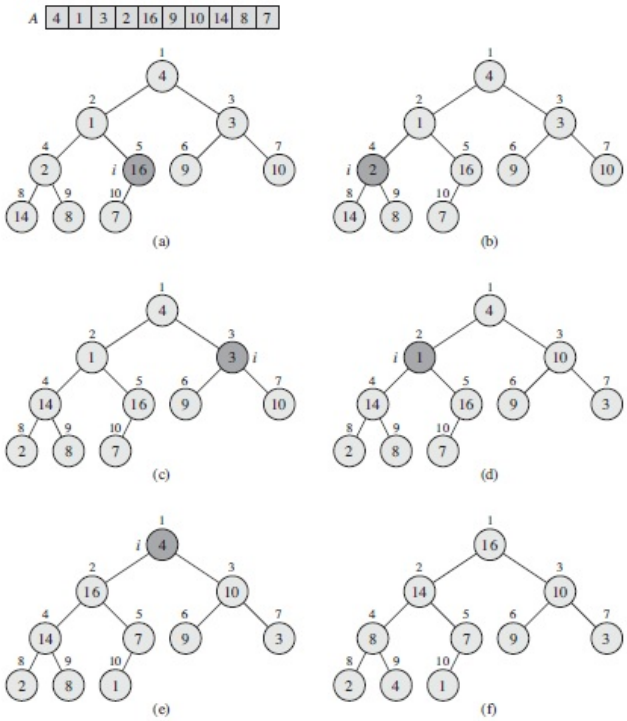

Figure 3: Exemplu de apel $BUILD - MAX - HEAP(A)$, unde $A.heap - size10$.

Costul fiecarui apel MAX-HEAPIFY este $\mathcal{O}(\lg n)$, iar BUILD-MAX-HEAP face $\mathcal{O}(n)$ asemenea apeluri. Concluzia, complexitatea este $\mathcal{O}(n \lg n)$.

1.4 Algoritmul HeapSort

Algoritmul HeapSort incepe prin apelarea procedurii BUILD-MAX-HEAP pentru a construi un max-ansamblu. Elementul maxim este stocat in radacina $A[1]$. La fiecare operatie conditia indeplinita este ca mutarile sa nu afecteze proprietatea de arbore partial max-ordonat.

```
    HEAPSORT(A)
1       BUILD-MAX-HEAP(A)
2       for i = A.length downto 2
3           exchange A[1] with A[i]
4           A.heap-size = A.heap-size - 1
5           MAX-HEAPIFY(A,1)
```

Figura 4 prezinta un exemplu de sortare, dupa ce initial a fost construit max-ansamblul.

Procedura HEAPSORT are complexitatea de $\mathcal{O}(n \lg n)$ deoarece apelul procedurii BUILD-MAXHEAP are $\mathcal{O}(n)$ si fiecare din cele $n - 1$ apeluri ale procedurii MAX-HEAPIFY se face in timp $\mathcal{O}(\lg n)$.

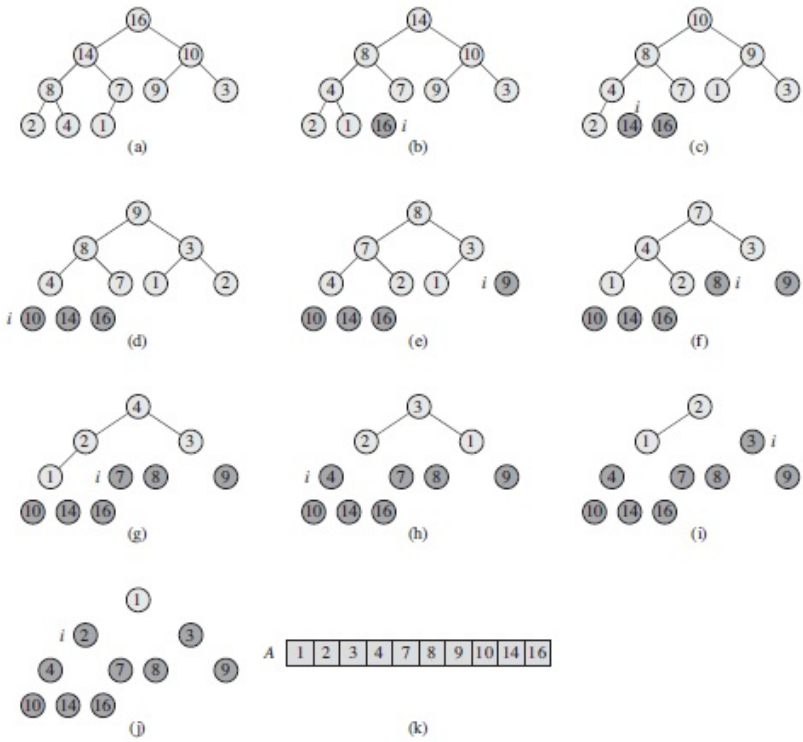

Figure 4: Sortarea cu Ansamble. (a) Ansamblul max-ordonat obtinut prin BUILD-MAX-HEAP; (b)-(j) Ansamblul dupa fiecare apel al MAX-HEAPIFY: doar nodurile cu gri deschis raman; (k) vectorul sortat rezultat