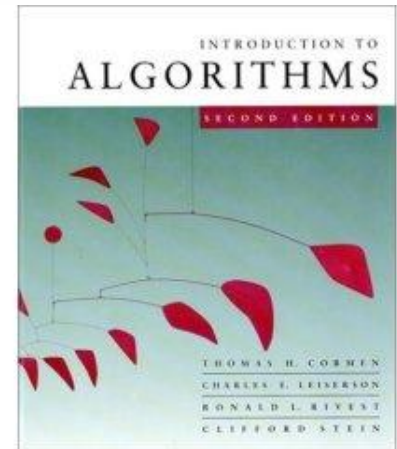




Heapsort

Algoritmi, Strutture Dati e Calcolo Parallelo

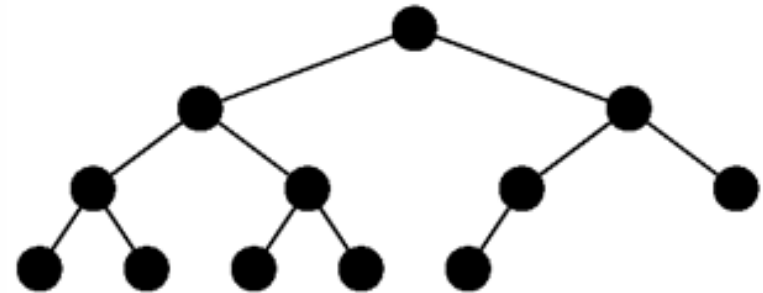
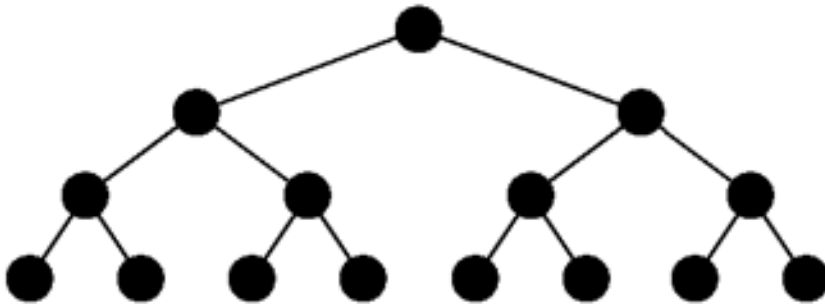
- ❑ Questo materiale è tratto dalle trasparenze del corso Introduction to Algorithms (2005-fall-6046) tenuto dal Prof. Leiserson all'MIT (<http://people.csail.mit.edu/cel/>)
- ❑ trasparenze del corso "Algoritmi e Strutture Dati" del prof. Alberto Montresor dell'Università di Trento
- ❑ T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein Introduction to Algorithms, Second Edition, The MIT Press, Cambridge, Massachusetts London, England McGraw-Hill Book Company
- ❑ Queste trasparenze sono disponibili sui siti <http://webpace.elet.polimi.it/lanzi>
<http://www.slideshare.net/pierluca.lanzi>



Ripasso sugli alberi binari...

- Albero binario perfetto
 - ▶ Tutte le foglie hanno la stessa altezza h
 - ▶ I nodi interni hanno due figli (sono di grado 2)
- Un albero perfetto
 - ▶ Ha altezza $\log N$
 - ▶ Un albero di altezza h , contiene $2^{h+1}-1$

- Albero binario completo
 - ▶ Tutte le foglie hanno profondità h o $h-1$
 - ▶ Tutti i nodi a livello h sono "accatastati" a sinistra
 - ▶ Tutti i nodi interni hanno grado 2, eccetto al più uno



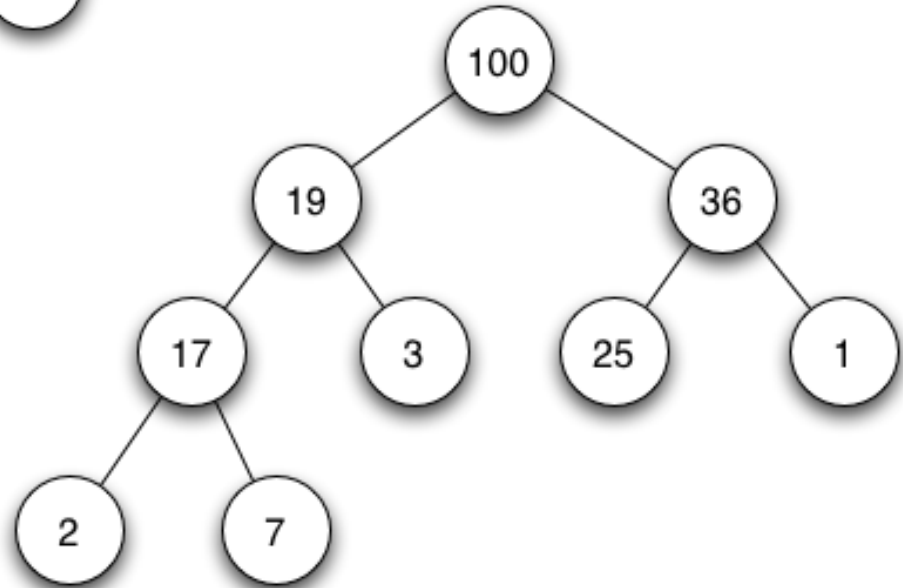
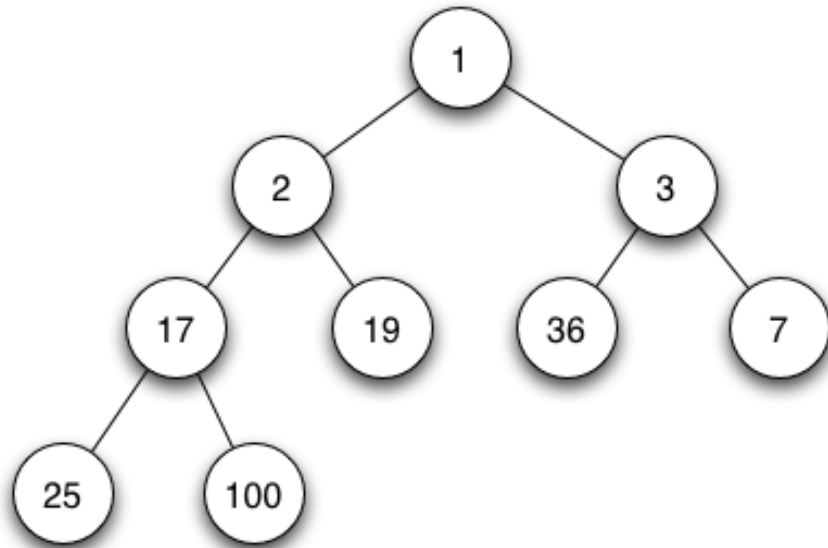
Heap...

Heap = Albero binario completo
che soddisfa la "heap property"

Min-heap: i valori dei nodi sono minori dei valori
sui nodi figlio ovvero, $A[\text{Parent}(i)] \leq A[i]$

Max-heap: i valori dei nodi sono maggiori dei
valori sui nodi figlio ovvero, $A[\text{Parent}(i)] \geq A[i]$

C'è un ordinamento parziale sui valori nei nodi

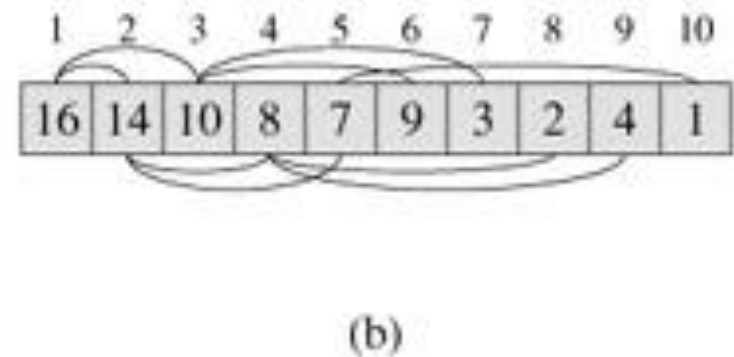
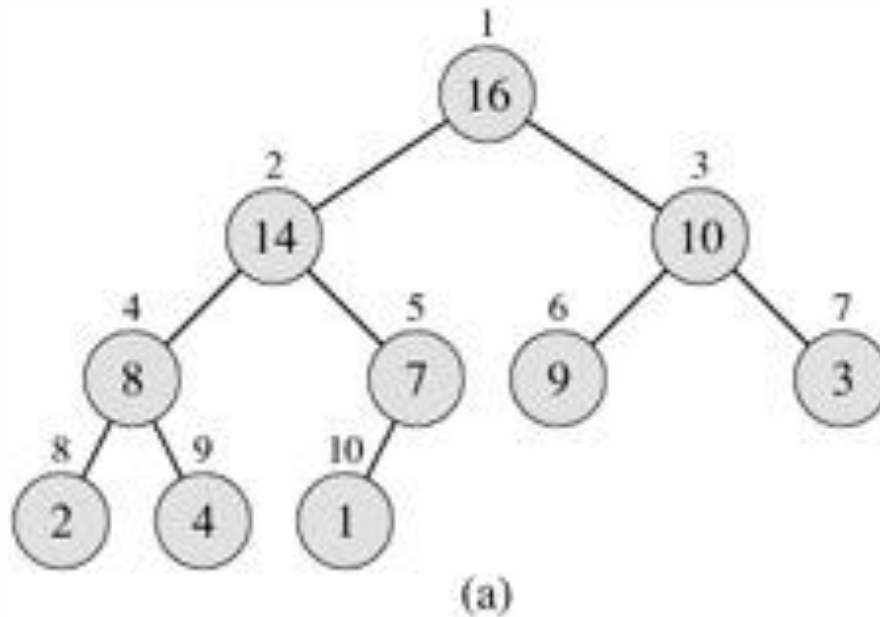


Immagini da Wikipedia

<http://upload.wikimedia.org/wikipedia/commons/b/bf/Max-heap.png>

<http://upload.wikimedia.org/wikipedia/commons/6/69/Min-heap.png>

Due rappresentazioni: alberi binari oppure array-heap per alberi binari completi



- $A[1]$ contiene la radice
- $\text{Parent}(i) = \lfloor i/2 \rfloor$
- $\text{Left}(i) = 2i$ mentre $\text{Right}(i) = 2i+1$

Rappresentare un max-heap con un array?

elementi heap \leq # elementi array

$A[1]$ contiene la radice

$\text{Parent}(i) = \lfloor i/2 \rfloor$

$\text{Left}(i) = 2i$ mentre $\text{Right}(i) = 2i+1$

Tipo di Dato Astratto per la Heap

```
template<class Elem, class Comp> class maxheap{
private:
    Elem* Heap;           // Pointer to the heap array
    int size;            // Maximum size of the heap
    int n;               // Number of elems now in heap
    void siftdown(int); // Put element in place
public:
    maxheap(Elem* h, int num, int max);
    int heapsize() const;
    bool isLeaf(int pos) const;
    int leftchild(int pos) const;
    int rightchild(int pos) const;
    int parent(int pos) const;
    bool insert(const Elem&);
    bool removemax(Elem&);
    bool remove(int, Elem&);
    void buildHeap();
};
```

Gestione delle max-heap

Max-heapify

mantiene la proprietà di max-heap

Build-Max-Heap

costruisce un max-heap da zero

Heapsort

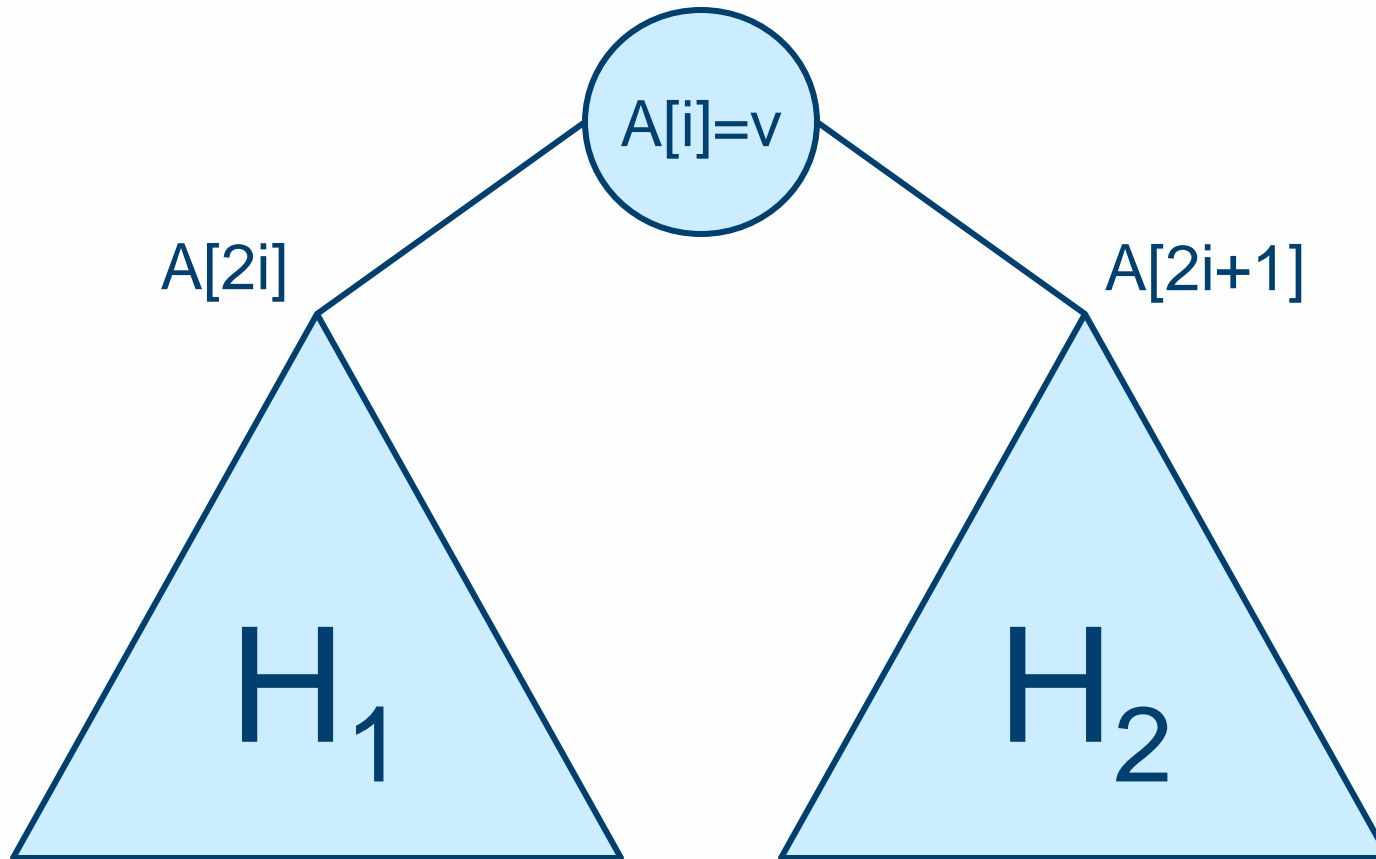
ordina un array sul posto

- ❑ Heapify(A, i): ripristina la proprietà di Heap nel sottoalbero radicato in i assumendo che i suoi sottoalberi destro e sinistro siano già degli Heap

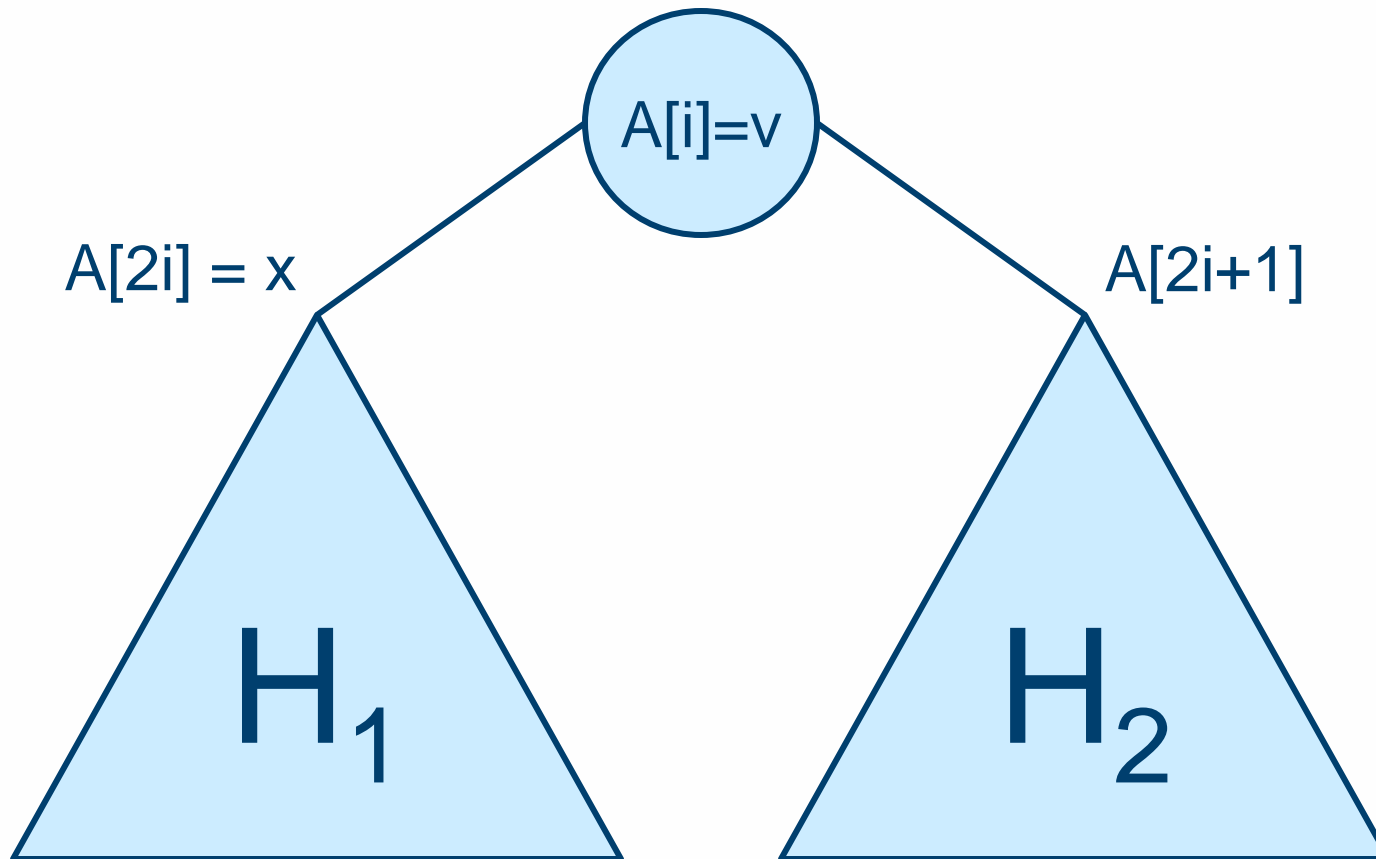
- ❑ **Input**
 - ▶ Un array A e un indice i
 - ▶ Gli alberi binari con radici $\text{Left}(i)$ e $\text{Right}(i)$ sono max-heap
 - ▶ Ma l'albero binario con radice i non è un max-heap

- ❑ **Obiettivo**
 - ▶ Ripristinare la proprietà di max-heap sul sottoalbero con radice i
 - ▶ Far "scendere" l'elemento $A[i]$ nell'array

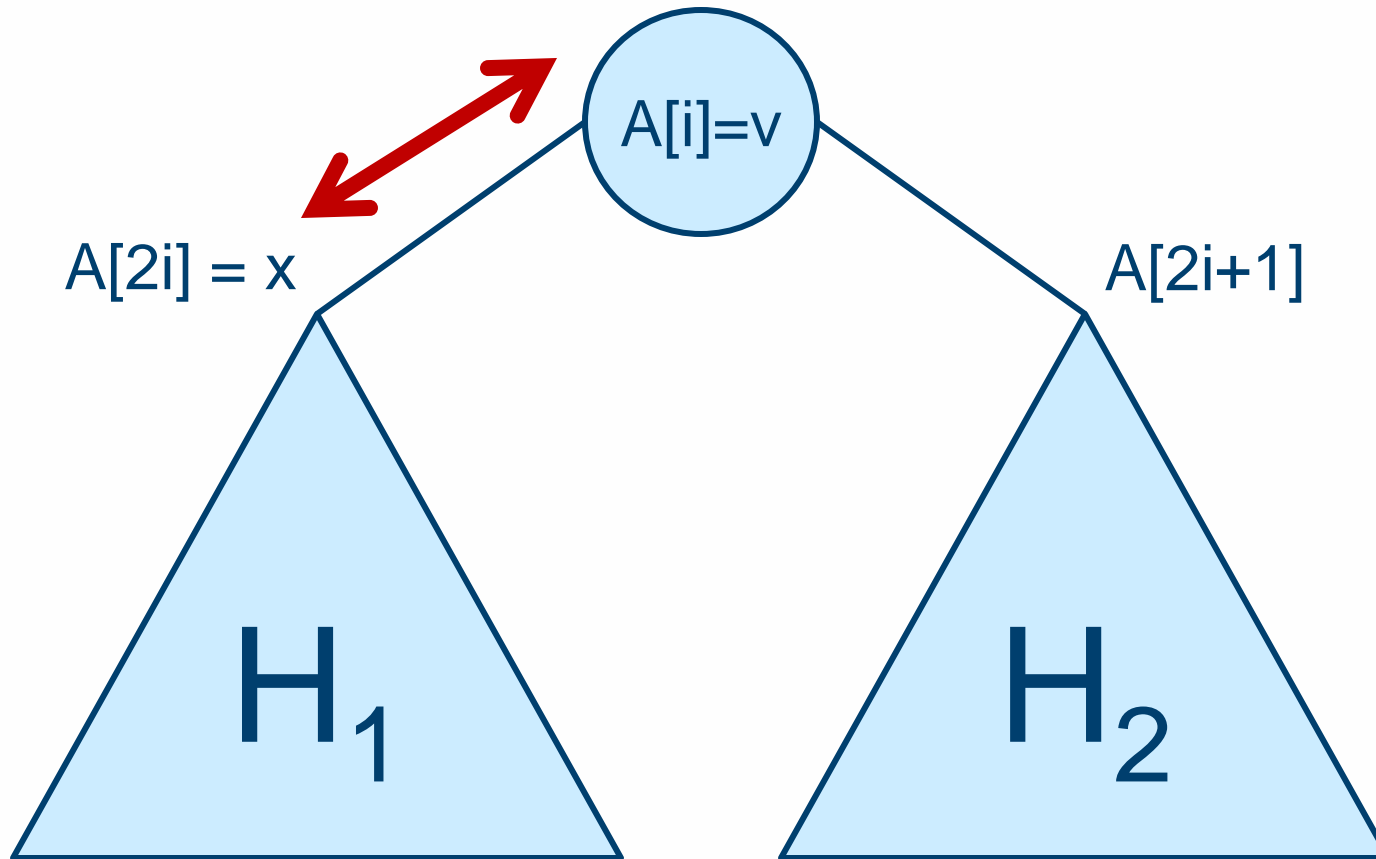
- Assumiamo di avere due heap H_1 e H_2 con radici $A[2i]$ e $A[2i+1]$ (nel caso di implementazione in un vettore) e un nuovo elemento v in posizione i



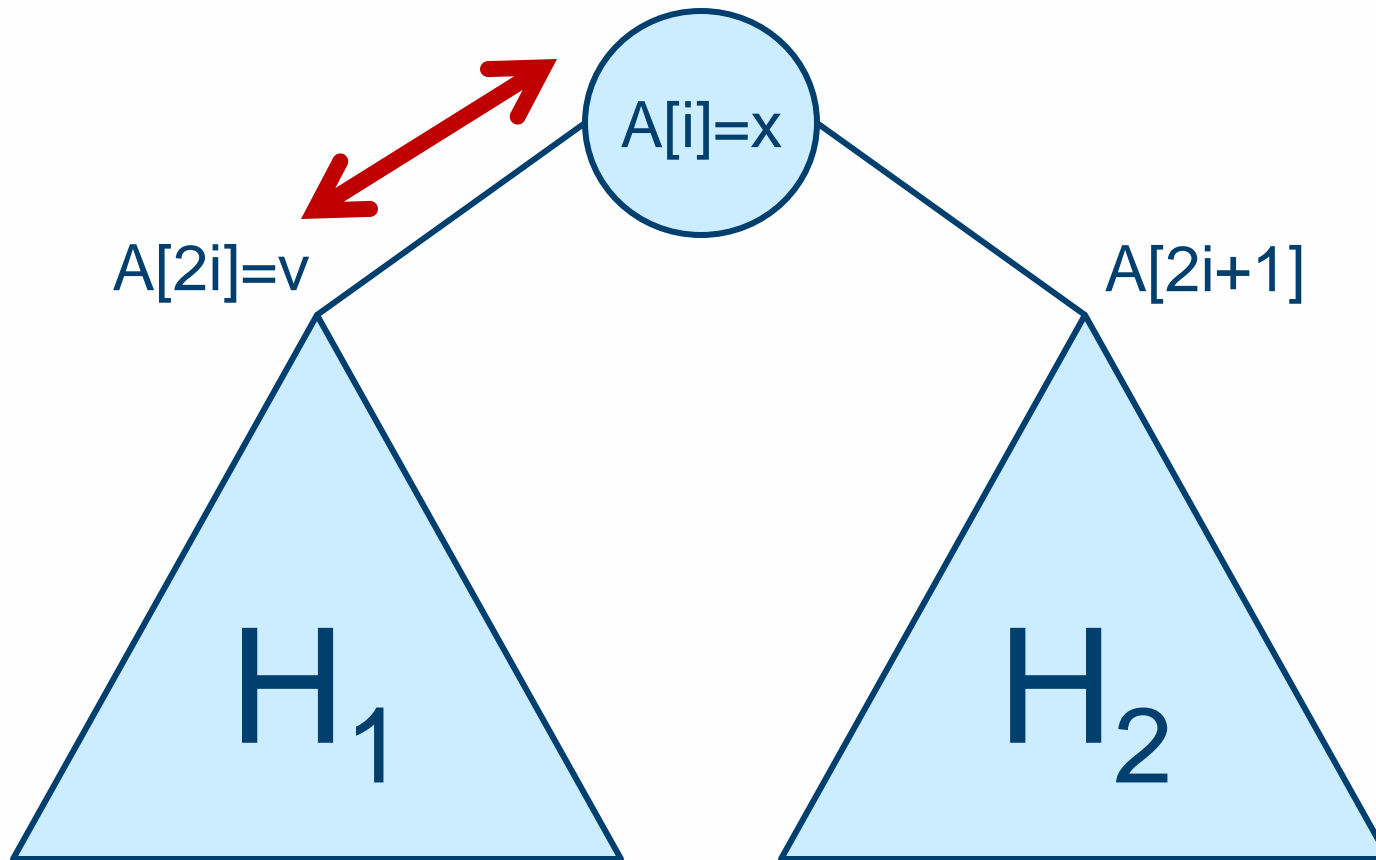
- Assumiamo che il valore v in posizione i violi la proprietà di heap ovvero, $v < A[2i]$ oppure $v < A[2i+1]$



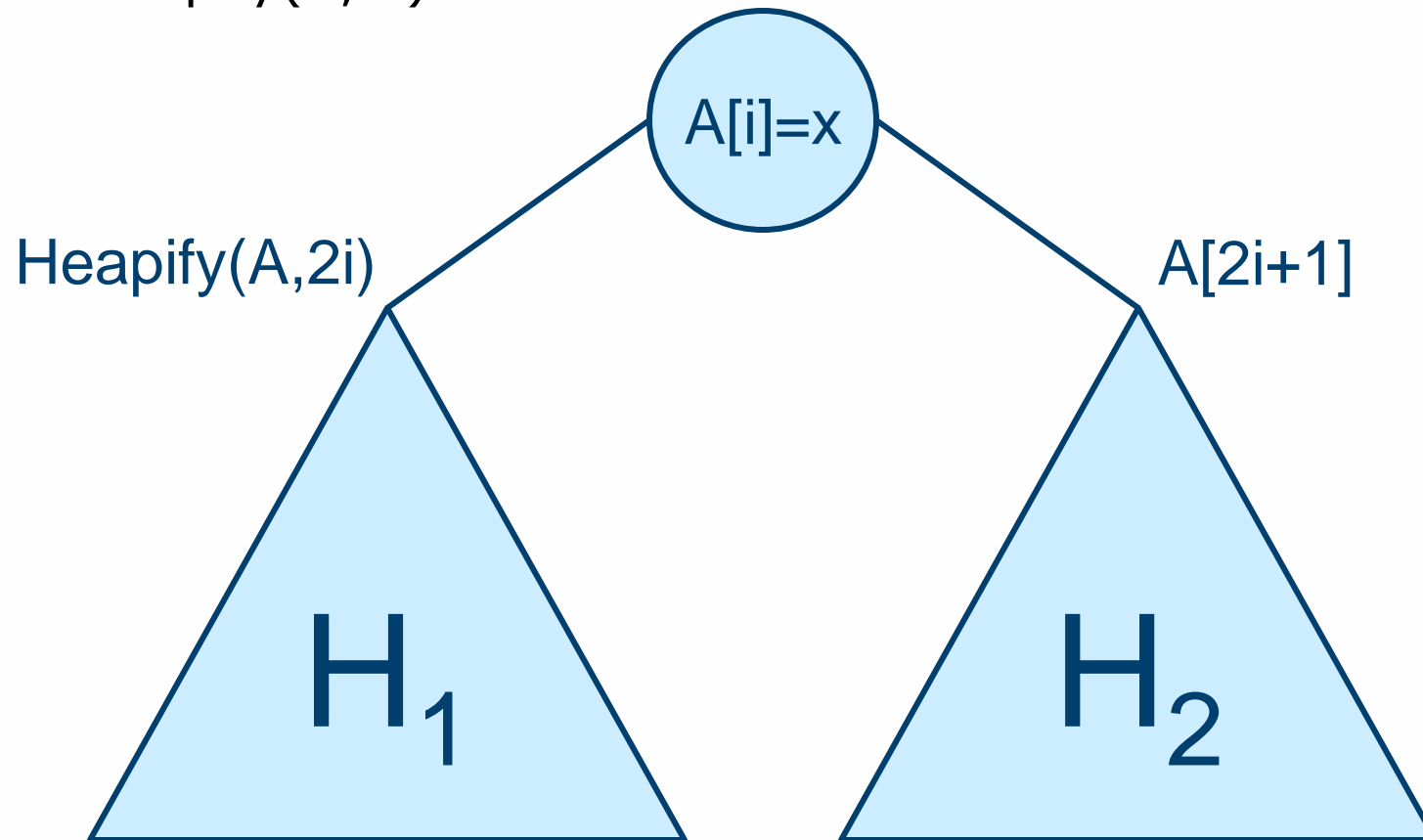
- ❑ Scambia v in posizione i con la più grande delle radici
- ❑ Supponiamo ad esempio che $A[2i] \geq A[2i+1]$



- Scambia $A[i]$ con $A[2i]$



- Applica ricorsivamente la procedura di Heapify sul sottoalbero la cui radice è stata scambiata
- In questo caso si riapplica al sottoalbero di radice $2i$, ovvero Heapify(A,2i)



```
max-heapify(A, i)
  l := left(i)
  r := right(i)
  if (l ≤ A.heapsize and A[l] > A[i])
    then max := l
  else max := i
  if (r ≤ A.heapsize and A[r] > A[max])
    then max := r
  if max ≠ i then
    swap(A, i, max)
    max-heapify(A, max)
```

□ Principio generale

- ▶ Sia $A[1..N]$ un array da ordinare
- ▶ I nodi $A[\lfloor N/2 \rfloor + 1..N]$ sono foglie dell'albero e quindi heap di un elemento da cui iniziare

□ La procedura build-max-heap() attraversa i restanti nodi dell'albero ed esegue max-heapify()

```
build-max-heap(A)
```

```
    A.heapsize := A.length
```

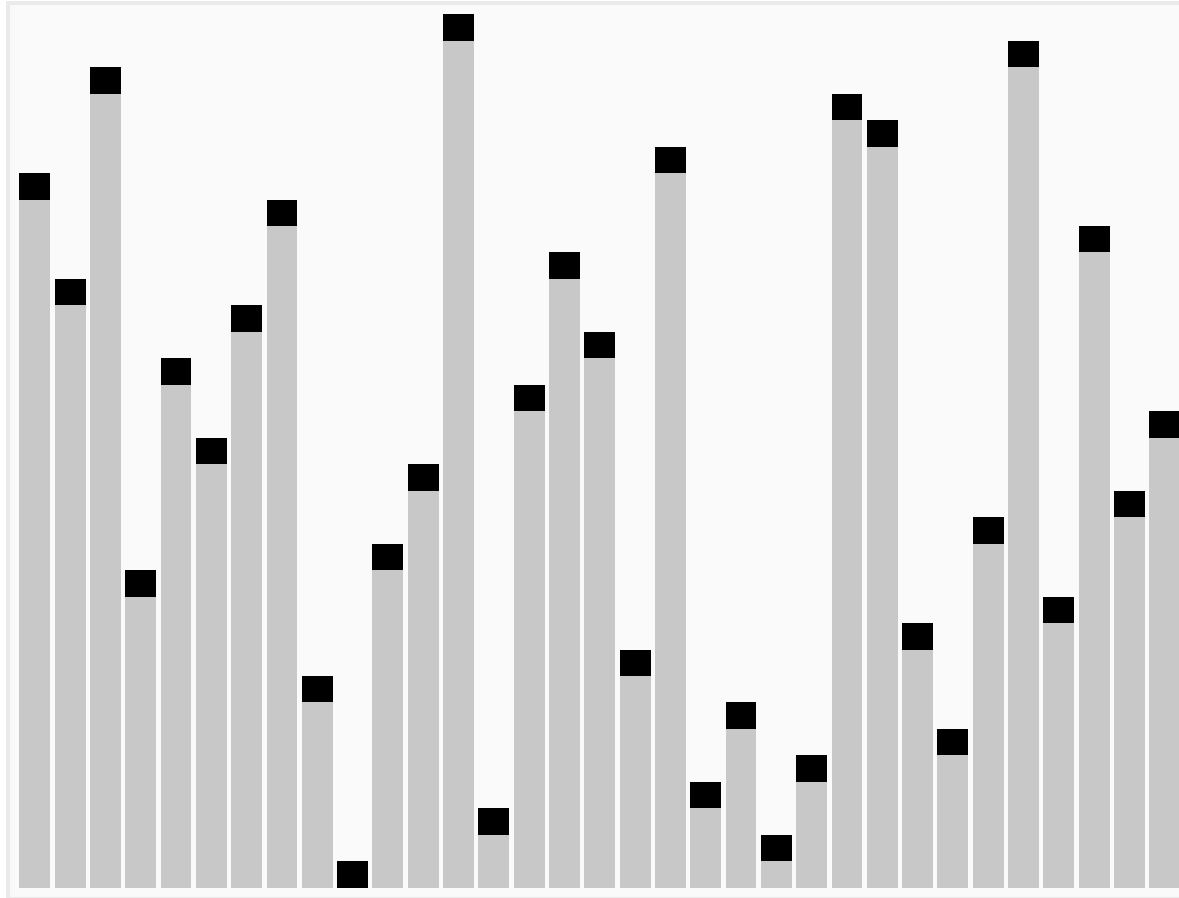
```
    for (i=A.length/2; i>=1; i--)
```

```
        max-heapify(A, i);
```

□ Intuizione

- ▶ Il primo elemento dello heap è sempre il massimo
- ▶ Andrebbe collocato nell'ultima posizione
- ▶ L'elemento in ultima posizione? Viene messo in testa
- ▶ Chiama `max-heapify()` su $n-1$ elementi, per ripristinare la situazione

```
heapsort(A) // array di n elementi
  build-max-heap(A)
  for (i=n; n>=2; i--)
    swap(A, 1, i)
    A.heapsize := A.heapsize-1
  max-heapify(A, 1)
```



Animazione del heapsort da Wikipedia

<http://en.wikipedia.org/wiki/Heapsort>

http://de.wikipedia.org/wiki/Image:Sorting_heapsort_anim.gif

Code di priorità

Code di priorità

memorizzano insiemi di oggetti
a cui è associata una priorità

restituisce l'elemento di priorità maggiore

Esempio: sistema operativo multitasking,
i job in esecuzione hanno diverse priorità

Implementazione: max-heap

- ❑ `insert(S, x)`
inserisce l'elemento x nella coda
- ❑ `maximum(S)`
restituisce l'elemento in S con priorità più alta
- ❑ `extract-max(S)`:
rimuove e restituisce l'elemento in S con priorità più alta
- ❑ `increase-priority(S, x, k)`:
aumenta la priorità di x al nuovo valore k , dove $k > x.p$

Complessità...

Complessità?

Max-heapify, nel caso medio, è $\Theta(\log n)$

Build-Max-Heap è $\Theta(\log n)$

Heapsort è $\Theta(n \log n)$

max-heap-insert, heap-extract-max,
heap-increase-key, ecc. sono $O(\log n)$

Problemi

- ❑ Problema 1: Considerare i valori memorizzati in un vettore,

1 3 20 32 80 15 22 40 60 93 81 42

Rappresenta una heap? (motivare opportunamente la risposta)

- ❑ Problema 2: Definiamo “invecchiamento” della heap l’operazione con cui è possibile decrementare/incrementare tutte le chiavi della stessa quantità. Esiste un algoritmo banale di complessità $\Theta(n)$. Delineare una struttura dati che estendendo il concetto di heap che abbiamo visto permetta di effettuare l’invecchiamento in $\Theta(1)$.
- ❑ Problema 3: Scrivere una classe C++ per implementare una coda FIFO utilizzando una heap.
- ❑ Problema 4: Qual è il numero minimo e massimo di una heap di altezza h ? (motivare opportunamente la risposta).

Sommario

- ❑ Heap: albero binario completo che soddisfa la heap property

- ❑ Due implementazioni
 - ▶ Alberi binari
 - ▶ Array

- ❑ Tre procedure principali (max o min)
 - ▶ Max-heapify: mantiene la proprietà di max-heap
 - ▶ Build-Max-Heap: costruisce un max-heap da zero
 - ▶ Heapsort: ordina un array sul posto

- ❑ Code di priorità
 - ▶ Memorizzano insiemi di oggetti con una priorità
 - ▶ Implementate tipicamente con una max-heap