Modulo: Strutture dati dinamiche gerarchiche [P2_09]

Unità didattica: struttura dati heap [6-T]

Titolo: Definizioni ed algoritmi di gestione

Argomenti trattati:
- ✓ Definizione e proprietà di un heap
- ✓ Rappresentazione di un heap mediante array
- ✓ Algoritmo di ripristino della “proprietà heap” su un nodo

Prerequisiti richiesti: alberi binari

Struttura dati HEAP

Un heap è un albero binario quasi completo i cui nodi sono etichettati tramite chiavi (da un insieme ordinato).

Proprietà max-heap:

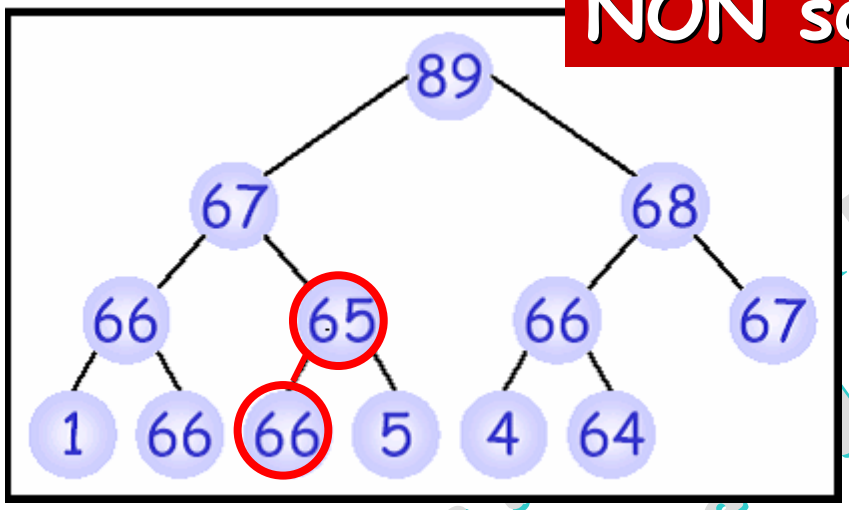
Se x è un qualsiasi nodo dell'heap (ad esclusione della radice) si ha $key(padre(x)) \geq key(x)$

Proprietà min-heap: $key(padre(x)) \leq key(x)$

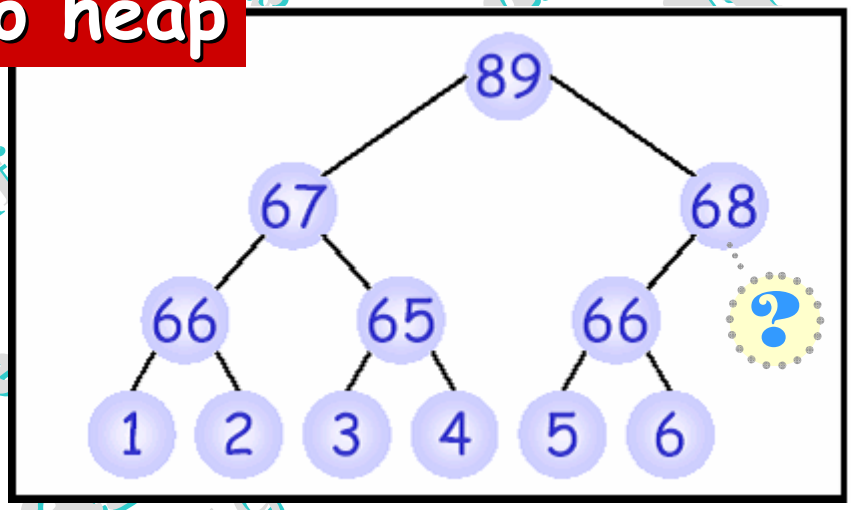
Ne consegue che in un **max-heap** (risp. **min-heap**) il **massimo** (risp. **minimo**) è memorizzato nella radice accessibile con tempo: $T(N)=\Theta(1)$.

Esempi

NON sono heap

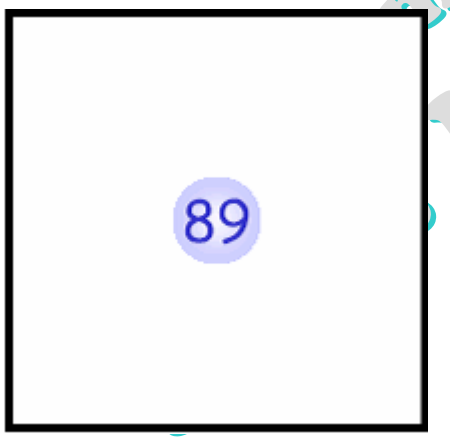


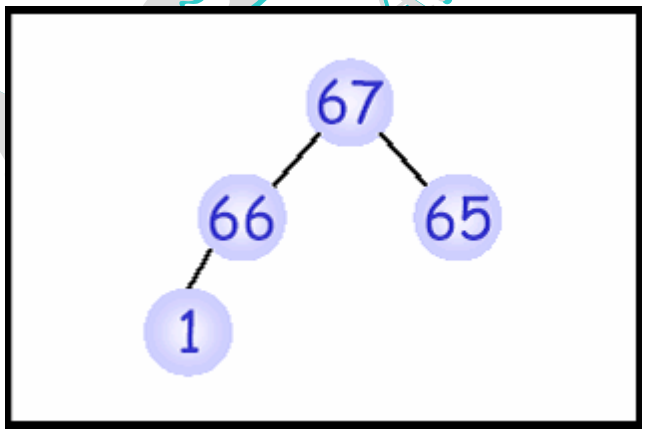
non verifica la proprietà heap

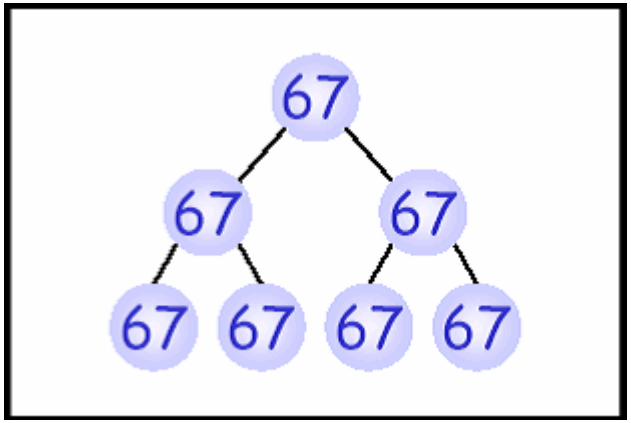


non è un albero quasi completo

sono heap







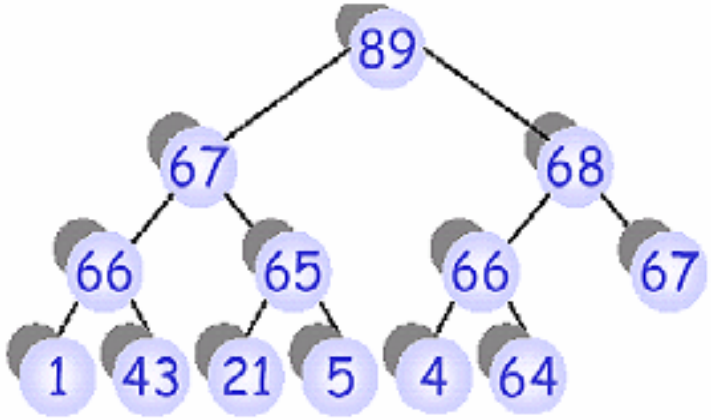
HEAP: rappresentazione tramite array

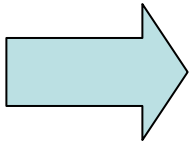
Essendo un albero binario, un heap può essere memorizzato in un array con solite le relazioni:

$\forall\ i \neq 1$ $\quad padre(a_i) = a_{\lfloor i/2 \rfloor}$

$\forall\ i \leq \lfloor n/2 \rfloor$ $\quad figlio\_sx(a_i) = a_{2i}$

$\forall\ i \leq \lfloor (n-1)/2 \rfloor$ $\quad figlio\_dx(a_i) = a_{2i+1}$





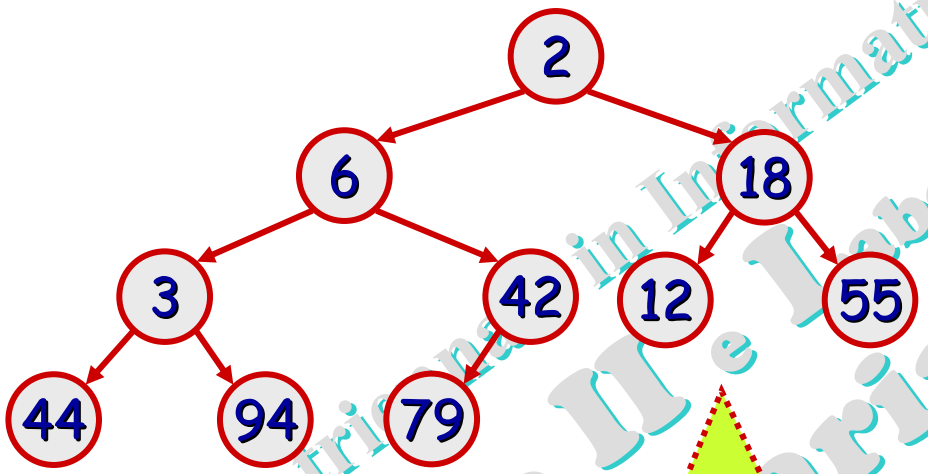
array a

1	89
2	67
3	68
4	66
5	65
6	66
7	67
8	1
9	43
10	21
11	5
12	4
13	64

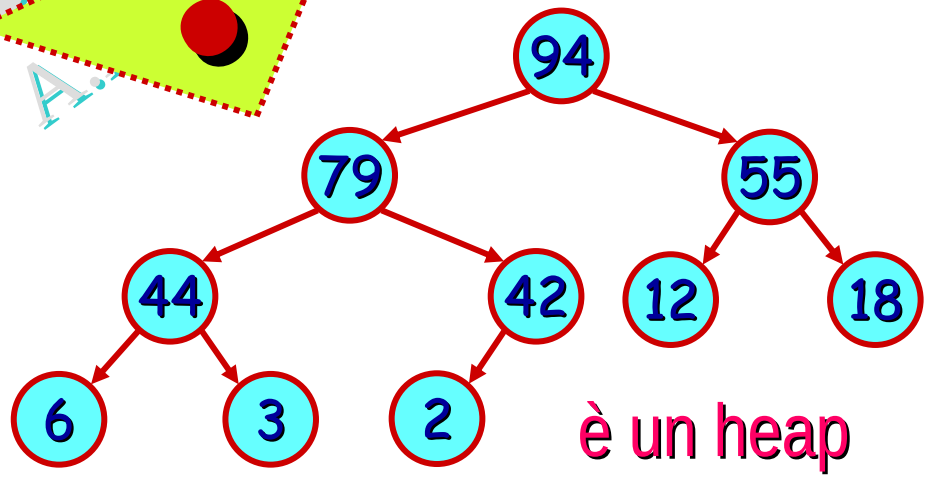
Come trasformare un array in un heap?

albero binario quasi completo rappresentato su array

2
6
18
3
42
12
55
44
94
79



non è un heap



è un heap

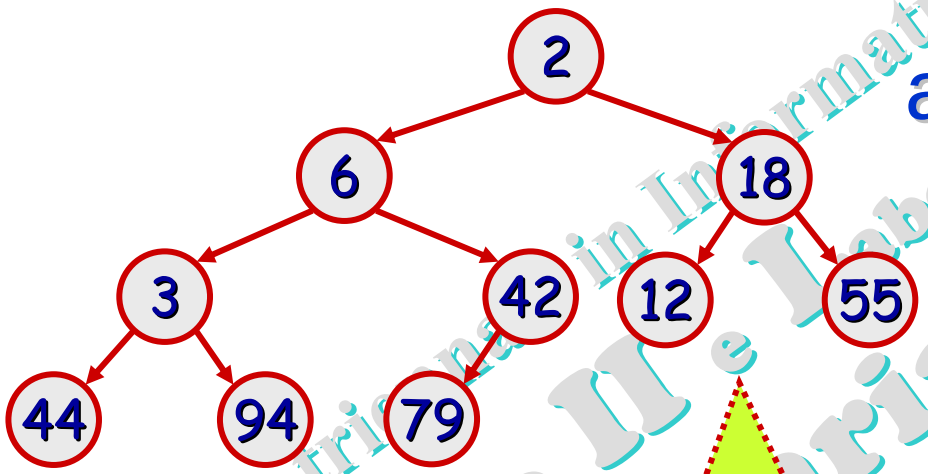
heap rappresentato su array

94
79
55
44
42
12
18
6
3
2

L'heap non è unico!

a albero binario rappresentato su array

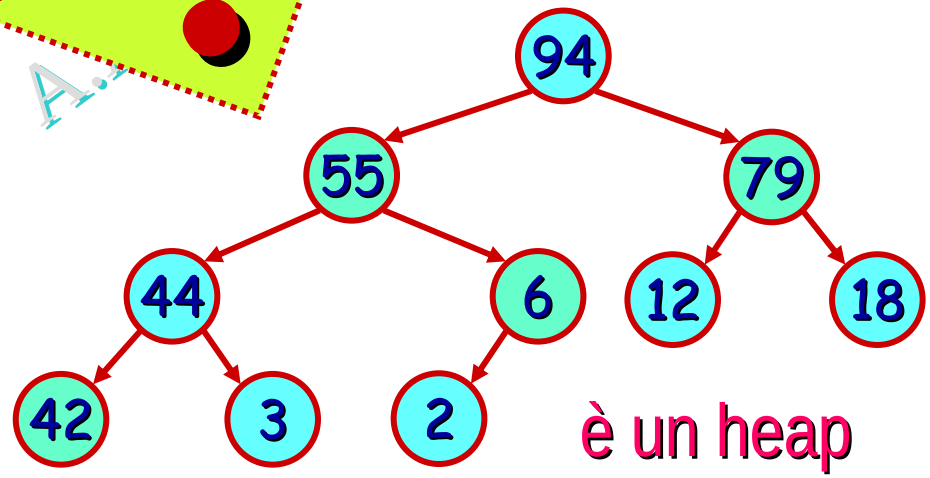
2
6
18
3
42
12
55
44
94
79



non è un heap

a heap rappresentato su array

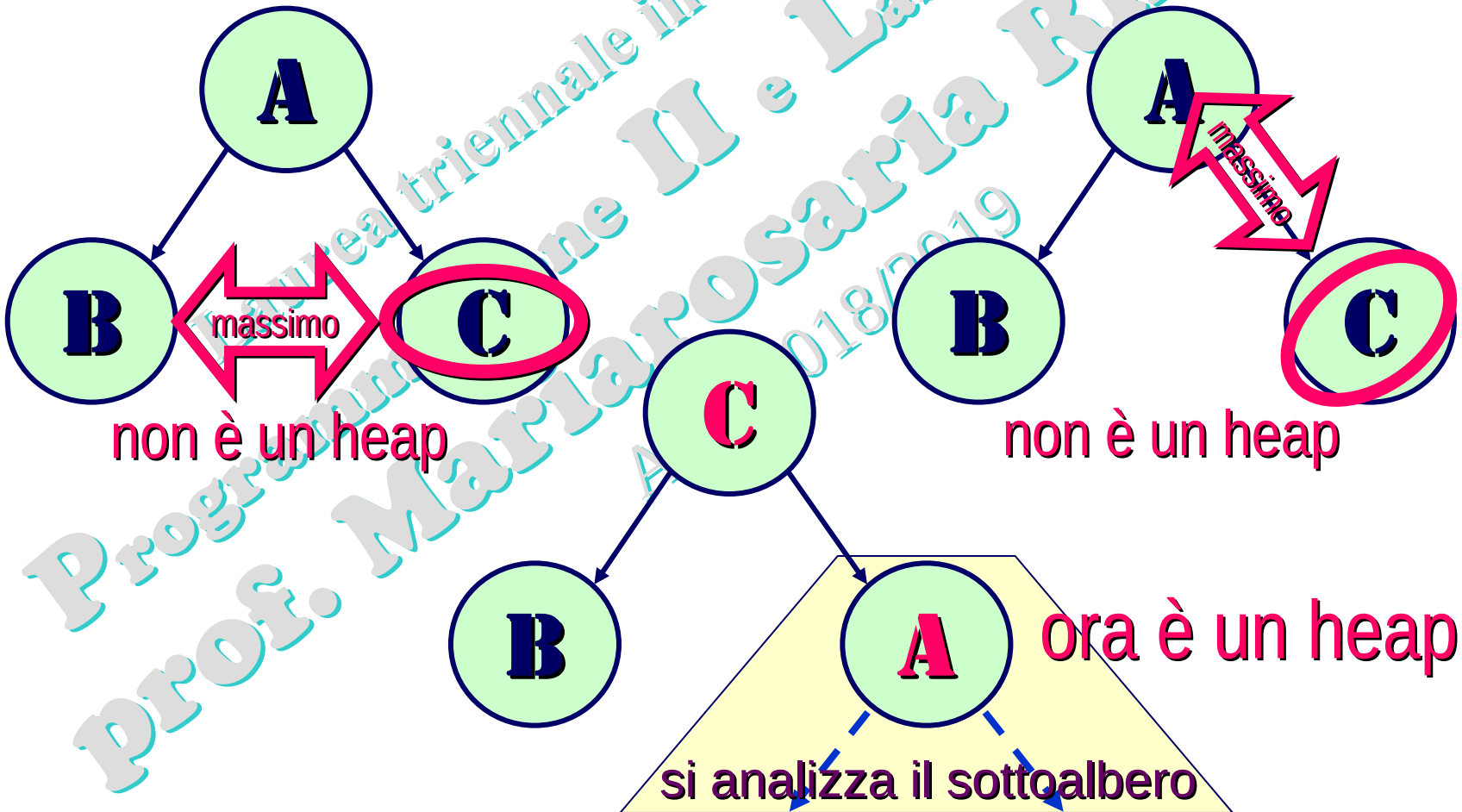
94
55
79
44
6
12
18
42
3
2



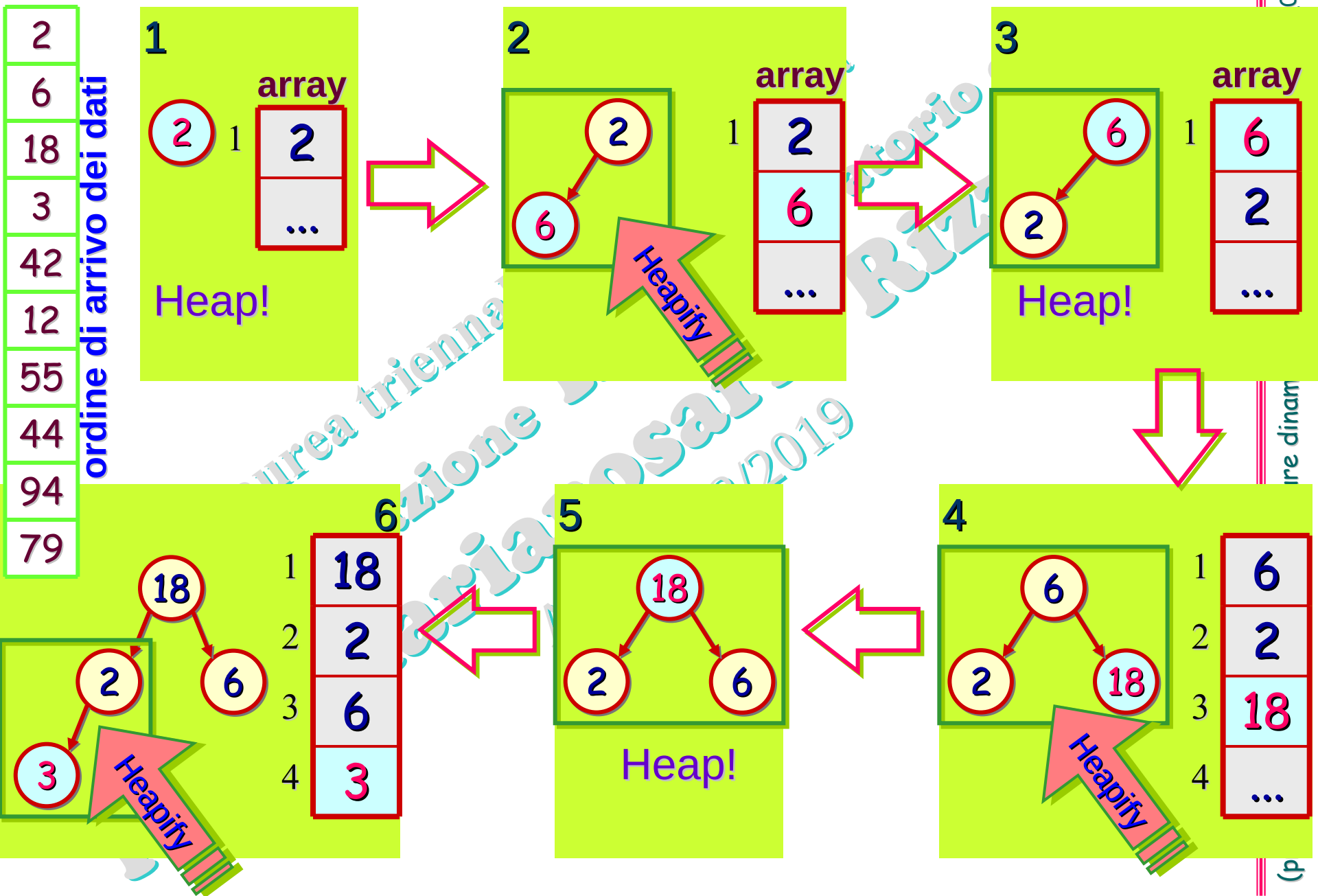
è un heap

Operazione base: procedura Heapify

Per ripristinare la proprietà heap si considera un nodo ed i suoi figli: si determina il figlio con chiave massima e si scambia il valore della chiave fra padre e tale figlio se non verificano la proprietà heap. Se è avvenuto lo scambio, si scende poi nel relativo sottoalbero per ripristinare la proprietà heap.



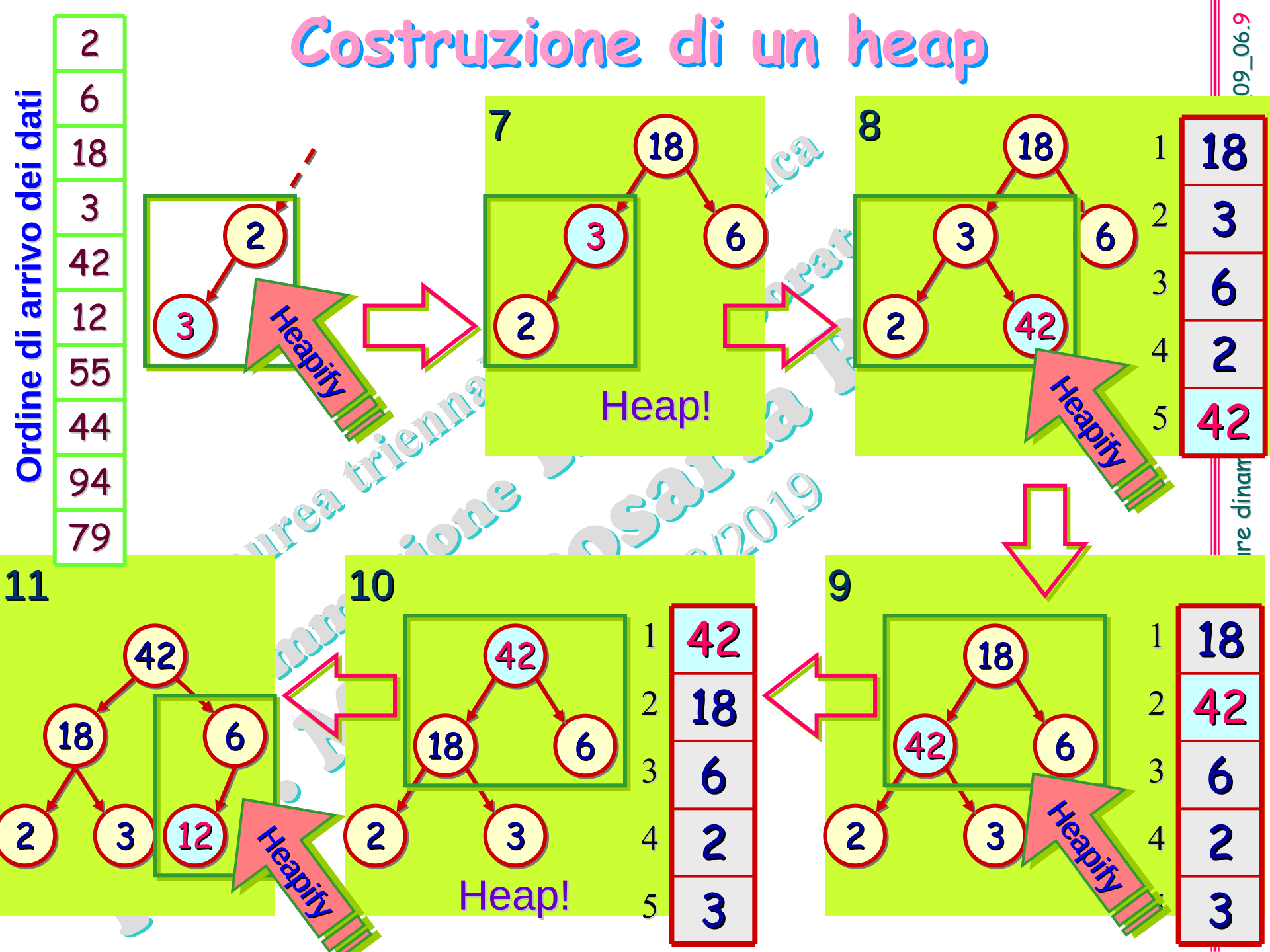
Costruzione di un heap: idea dell'algoritmo



Costruzione di un heap

Ordine di arrivo dei dati

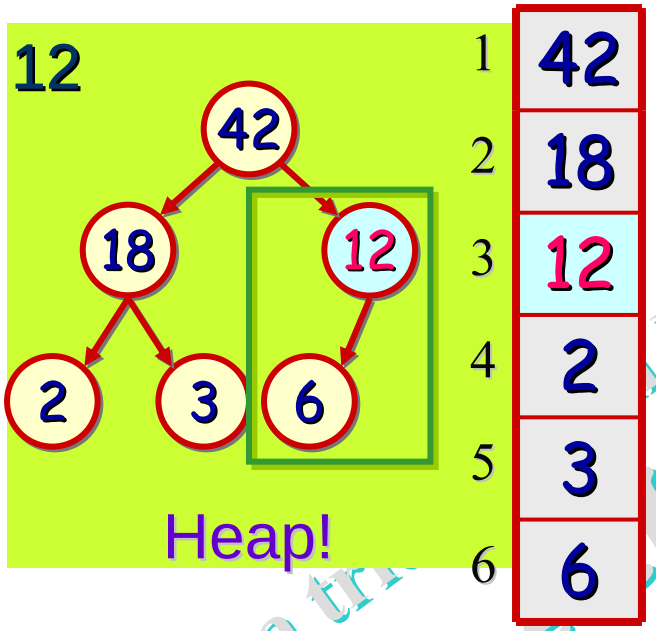
2
6
18
3
42
12
55
44
94
79



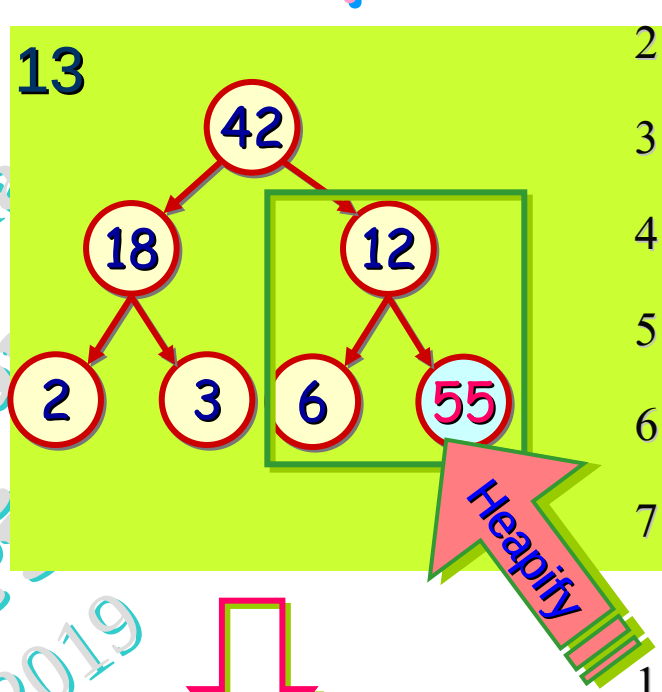
Costruzione di un heap

ordine di arrivo dei dati

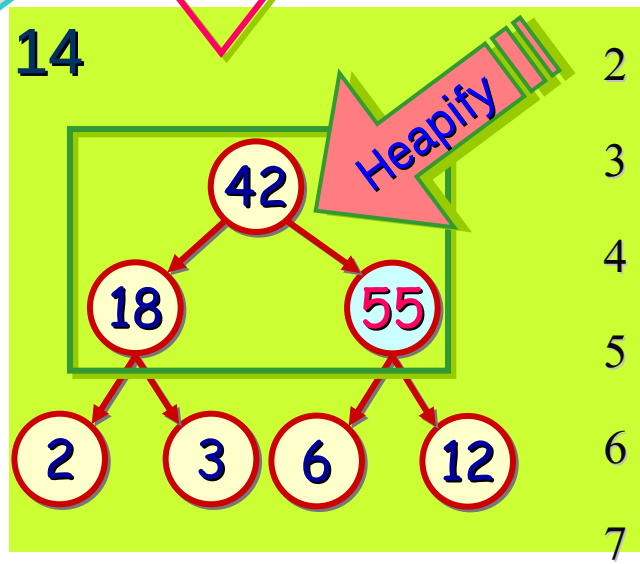
2
6
18
3
42
12
55
44
94
79



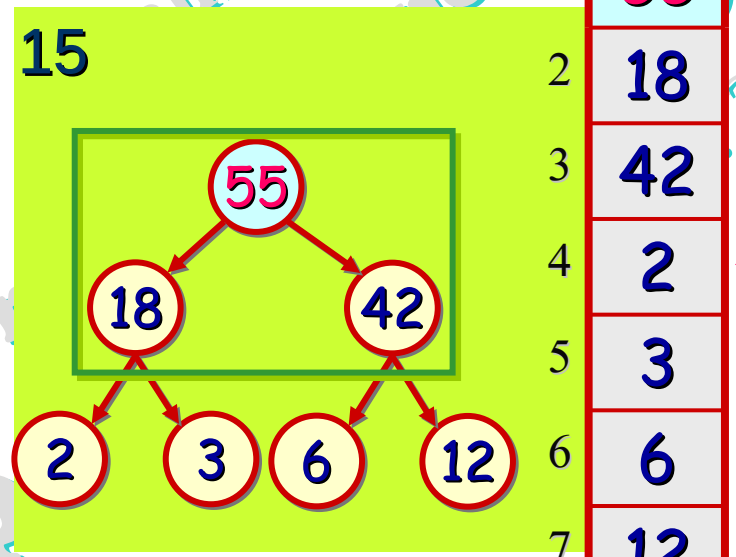
1	42
2	18
3	12
4	2
5	3
6	6



1	42
2	18
3	12
4	2
5	3
6	6
7	55



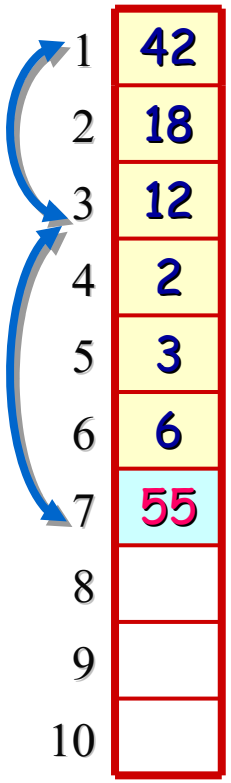
1	42
2	18
3	55
4	2
5	3
6	6
7	12



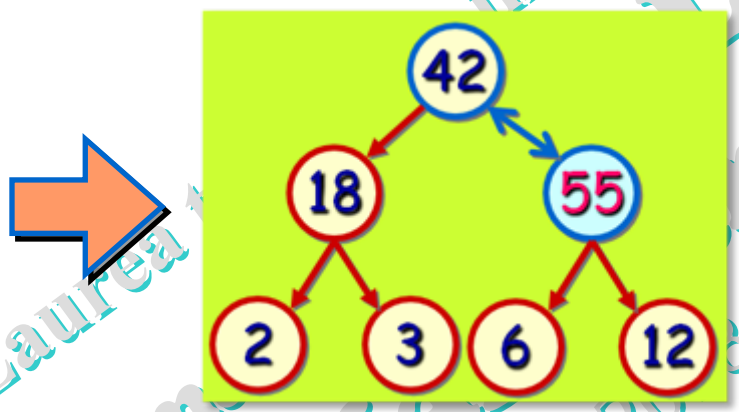
1	55
2	18
3	42
4	2
5	3
6	6
7	12

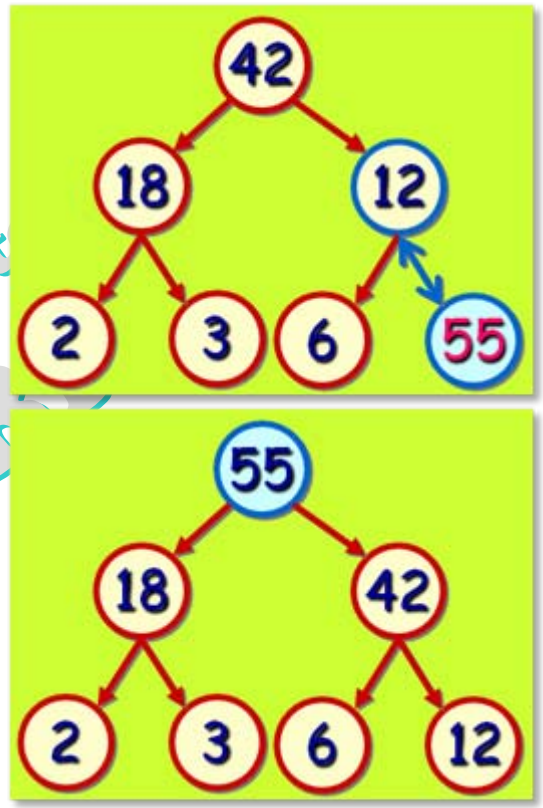
...

Inserimento di un nuovo nodo nell'heap



1. Il nuovo nodo viene inserito nella prima componente libera dell'array
2. Viene ripristinata la proprietà dell'heap "bottom-up" confrontando il nuovo nodo col padre, scambiandoli eventualmente.






```
Max_Heap_Insert(A,x)
{
    Heap_size(A) := Heap_size(A)+1;
    i := Heap_size(A);
    A[i] := x;
    while (i>1 or A[parent(i)] < A[i])
    {
            scambia(A[i],A[parent(i)]);
            i := parent(i);
    }
}
```

$$T(N)=O(\log_2 N)$$

Code con Priorità (Priority Queues)

Un coda con priorità è una struttura dati che serve per mantenere un insieme di elementi, ciascuno con un valore che rappresenta la sua priorità.
Analogamente agli heap, esistono due tipi di code con priorità: code con max-priorità e code con min-priorità.

Operazioni su una coda con max-priorità:

- Inserire un elemento con data priorità.
- Estrarre l'elemento di max-priorità.
- Aumentare la priorità di un elemento.

Una coda con max-priorità viene usata, ad es., dallo scheduler del S.O.: quando un job è ultimato o interrotto, viene selezionato il job con priorità più alta tra quelli in attesa nella coda; un nuovo job può essere aggiunto nella coda in un momento qualsiasi.

Una coda con min-priorità viene usata, ad es., dall'event-driven simulator: la priorità degli eventi da simulare nella coda è il tempo in cui l'evento si verifica. Gli eventi vanno simulati secondo l'ordine dei loro tempi perché un evento può generare altri eventi. Es.: utenti collegati ad un server.

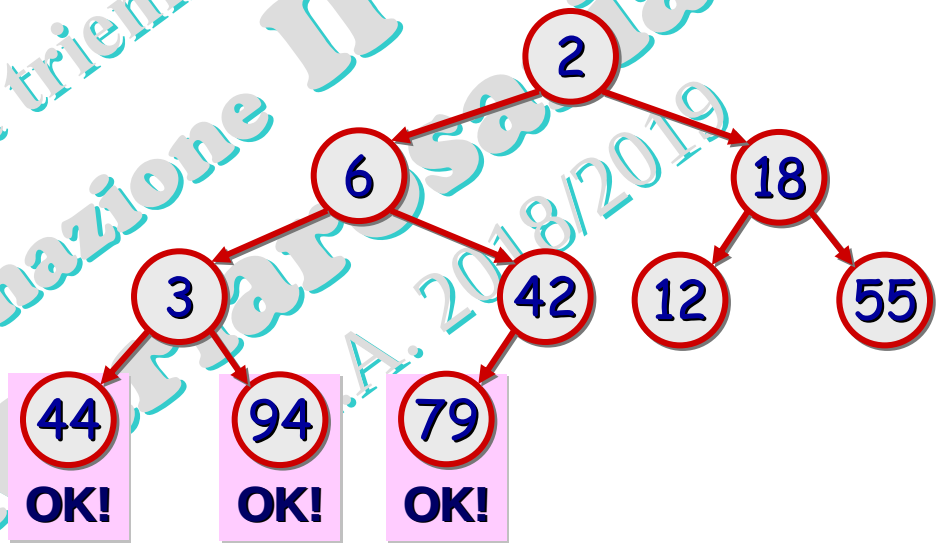
Una coda con max-priorità (min-priorità) può essere implementata mediante un max-heap (min-heap).

Come trasformare un albero binario in un heap?

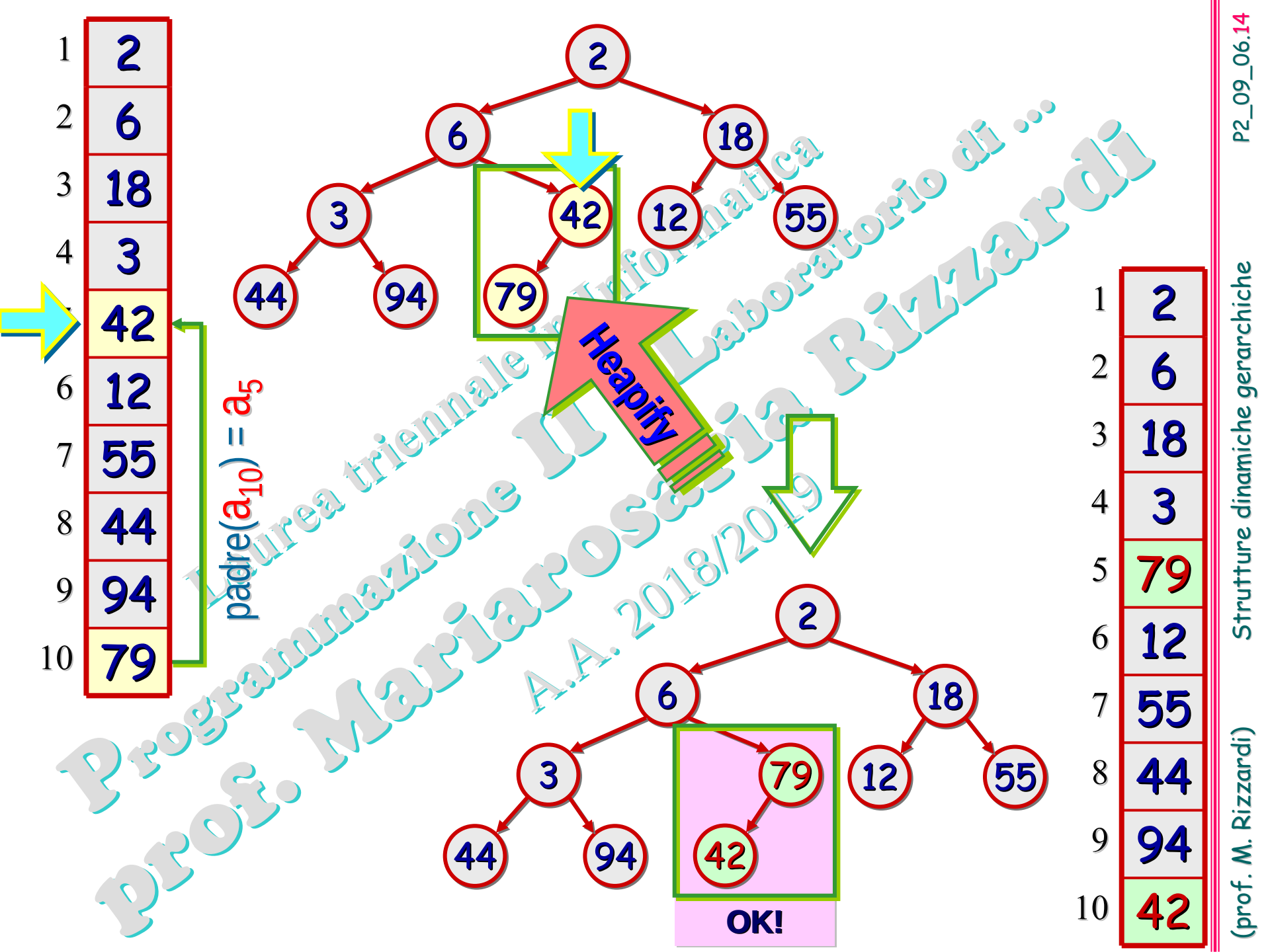
Si usa la procedura **Heapify*** in modo **bottom-up**, dai **livelli più bassi** dell'albero in su, per convertire l'array in un heap.

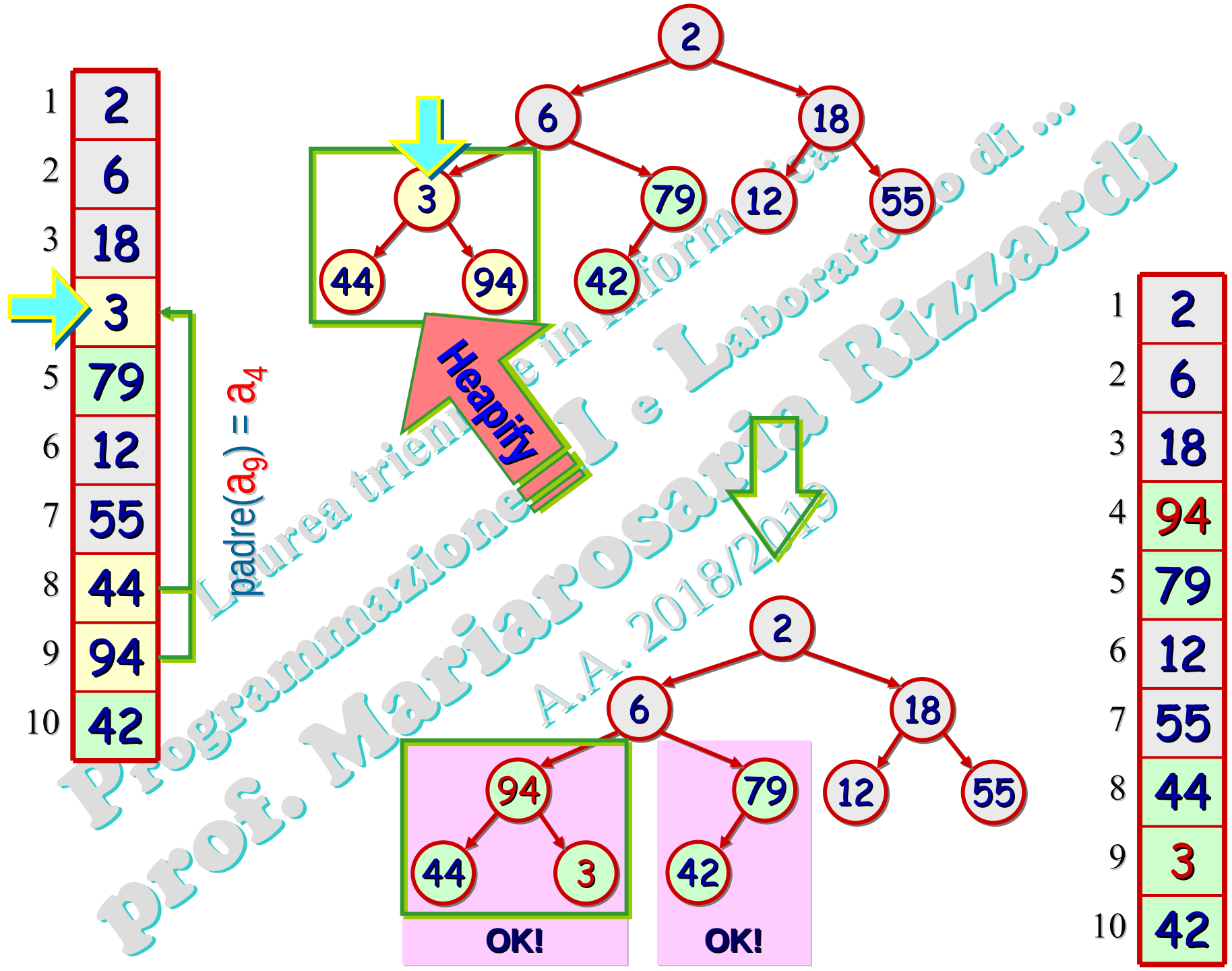
* Heapify ripristina la proprietà heap sui nodi

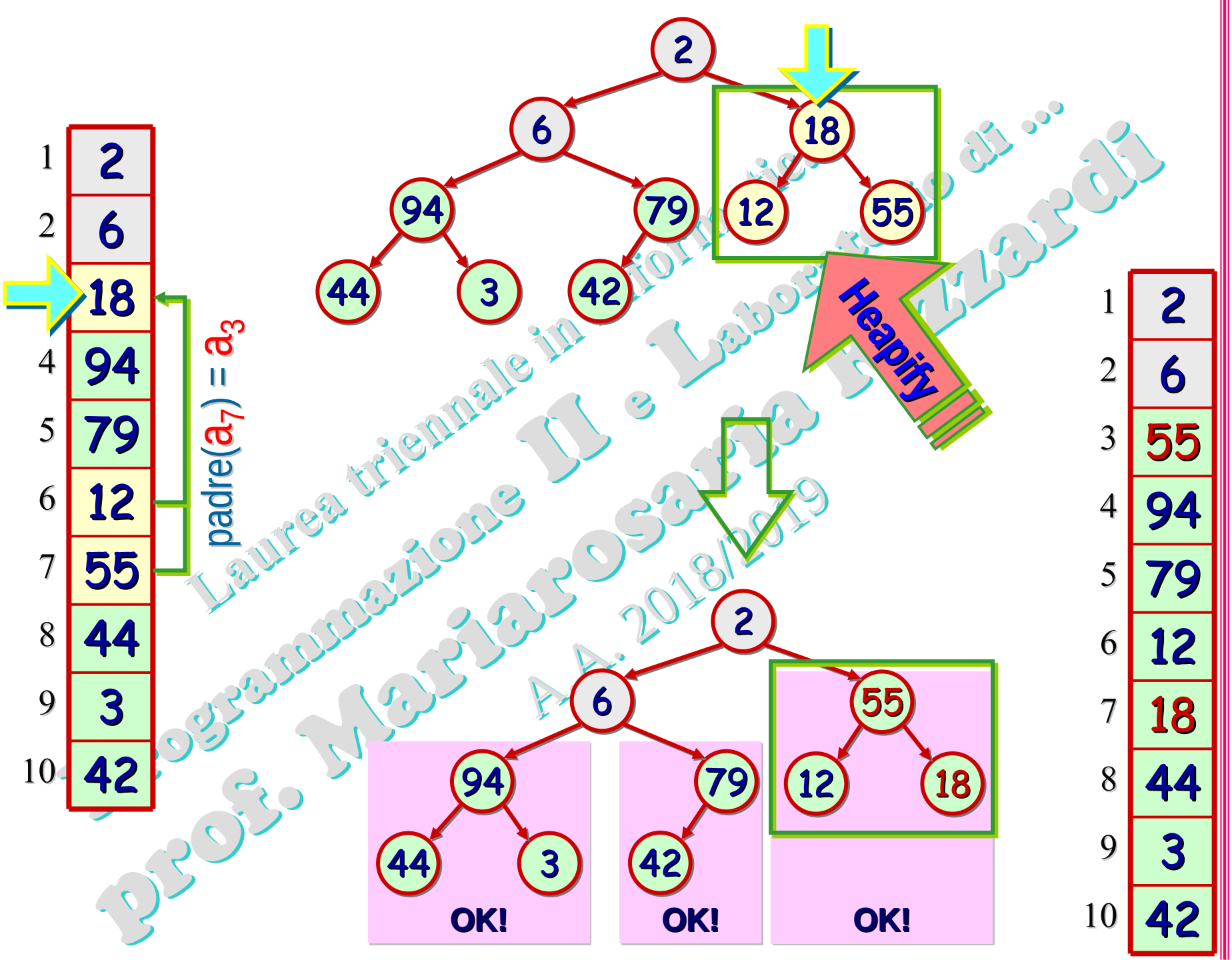
2
6
18
3
42
12
55
44
94
79



1) Poiché le singole foglie costituiscono ognuna un heap, si inizia ad applicare la procedura Heapify dal penultimo livello (dal padre dell'ultima foglia).

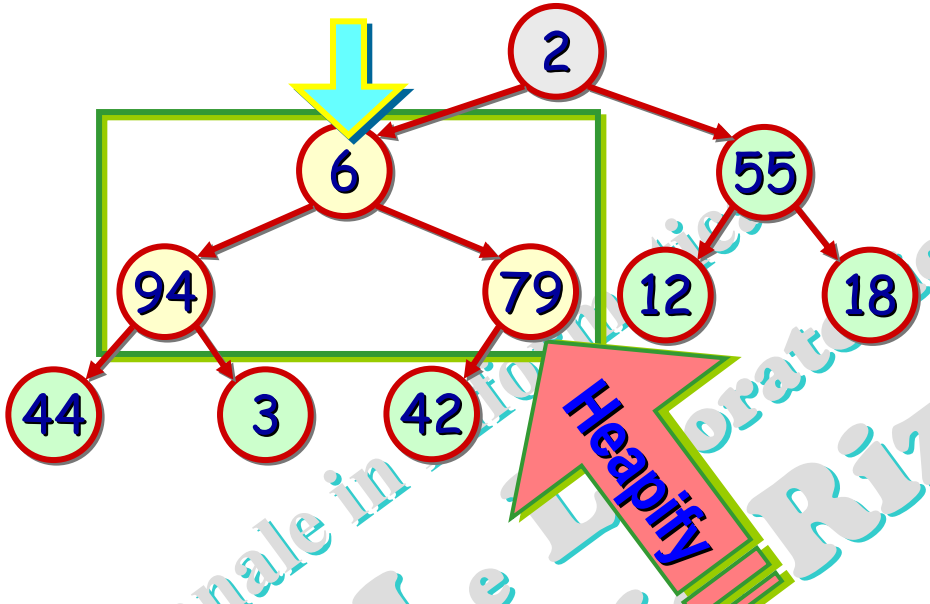


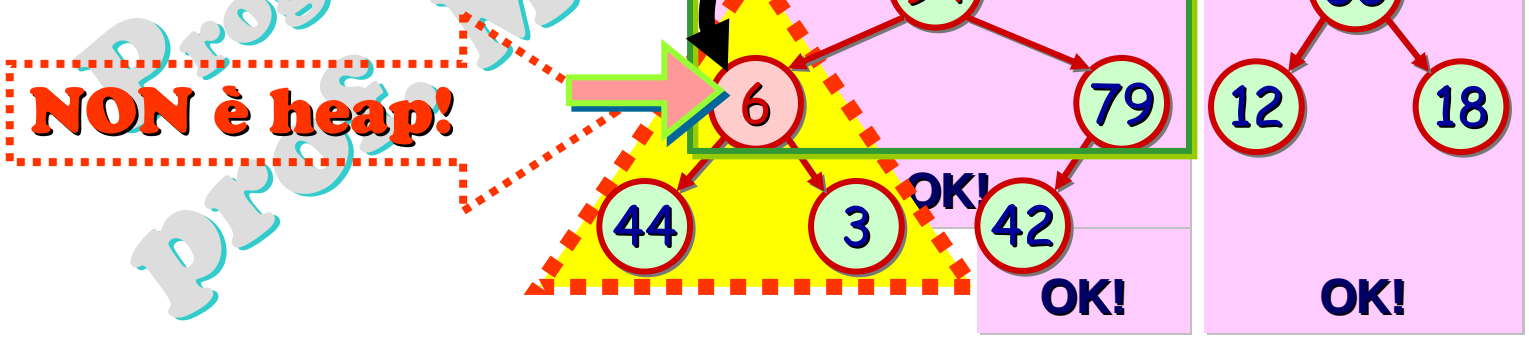

1 2
2 6
3 18
3
5 79
6 12
7 55
8 44
9 94
10 42
padre(a_9) = a_4
2
6
18
3
79
12
55
44
94
42
Heapify
2
6
18
94
79
12
55
44
3
42
OK!
OK!
1 2
2 6
3 18
4 94
5 79
6 12
7 55
8 44
9 3
10 42
Programmazione I e Laboratorio di ...
prof. Mariarosaria Rizzardi
Laurea triennale in Informatica
A.A. 2018/2019
P2_09_06.15
Strutture dinamiche gerarchiche
(prof. M. Rizzardi)


1 2
2 6
3 18
4 94
5 79
6 12
7 55
8 44
9 3
10 42
padre(a7) = a3
2
6
18
94
79
12
55
44
3
42
Heapify
2
6
55
94
79
12
18
44
3
42
OK!
OK!
OK!
1 2
2 6
3 55
4 94
5 79
6 12
7 18
8 44
9 3
10 42
Laurea triennale in Informatica
Programmazione II e Laboratorio di ...
prof. Mariarosaria Rizzardi
A.A. 2018/2019
P2_09_06.16
Strutture dinamiche gerarchiche
(prof. M. Rizzardi)

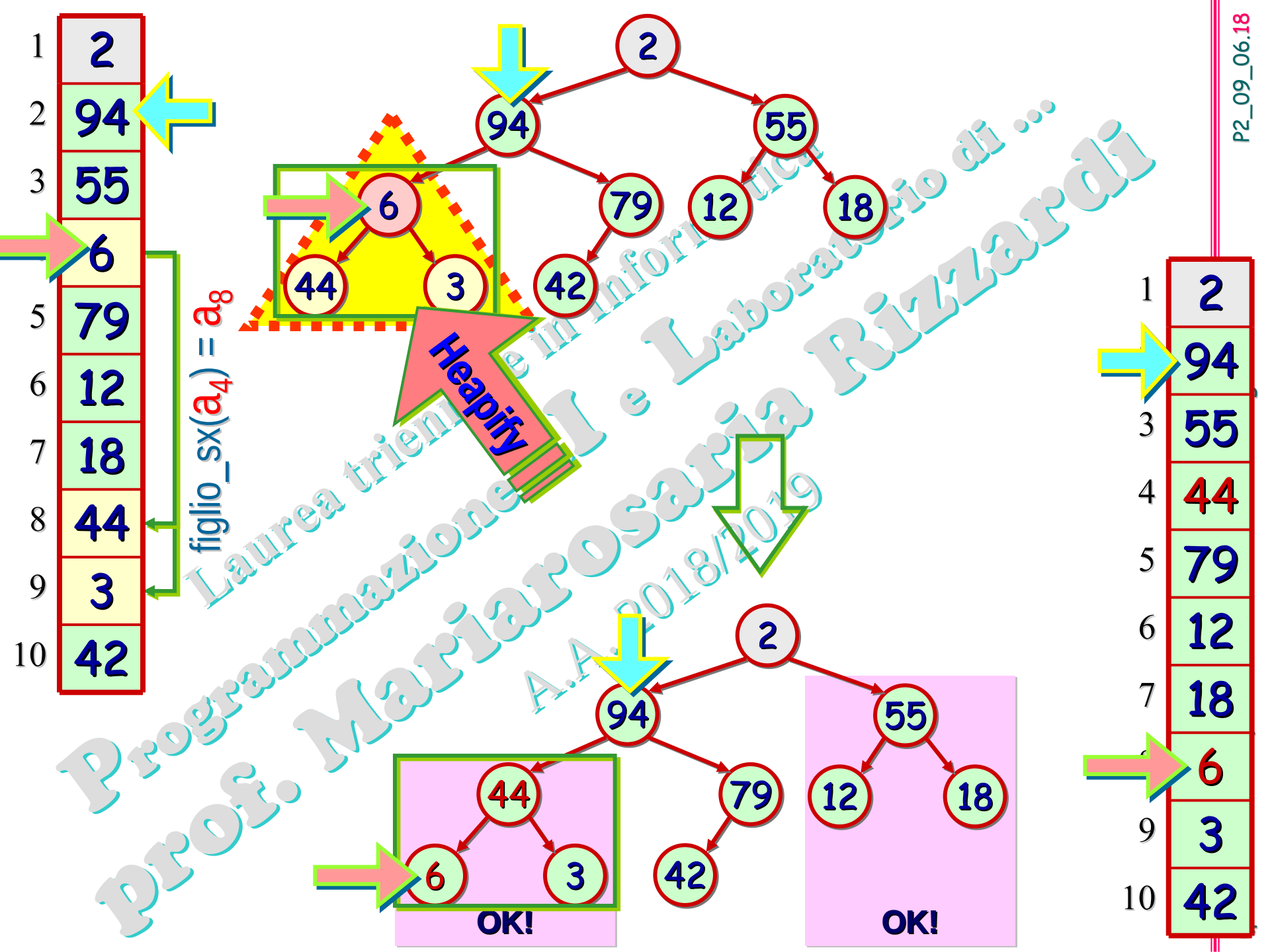
1	2
2	6
3	55
4	94
5	79
6	12
7	18
8	44
9	3
10	42

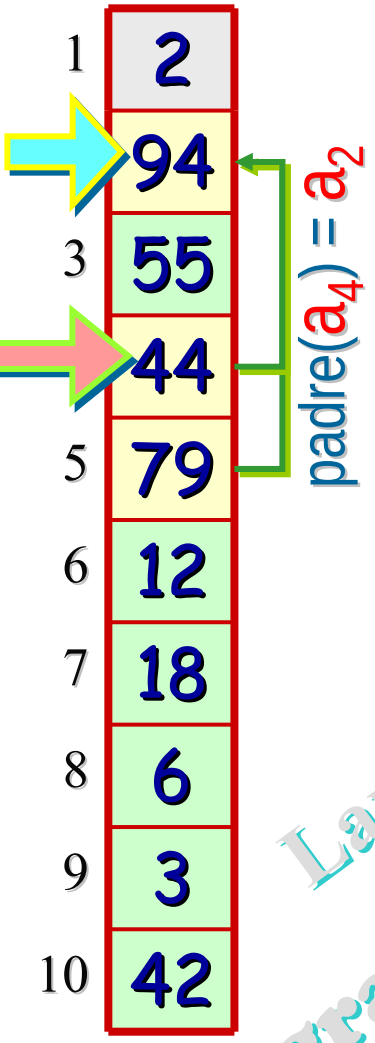
padre(a_5) = a_2

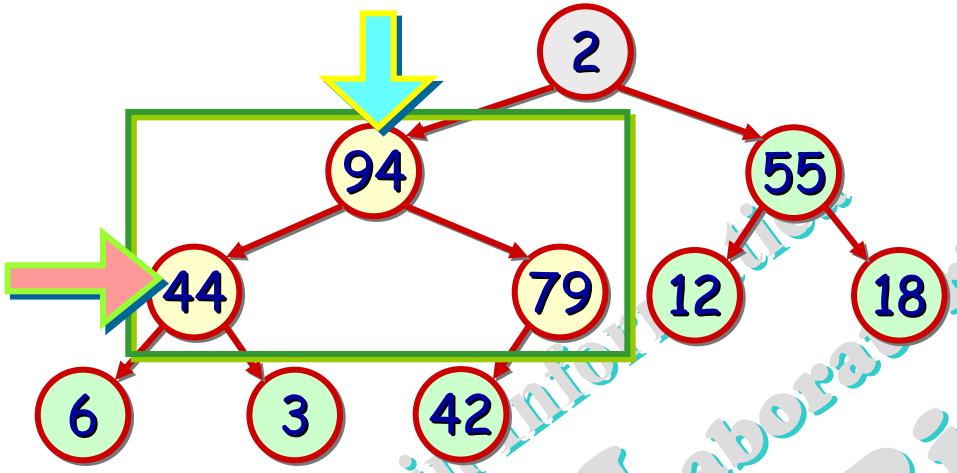




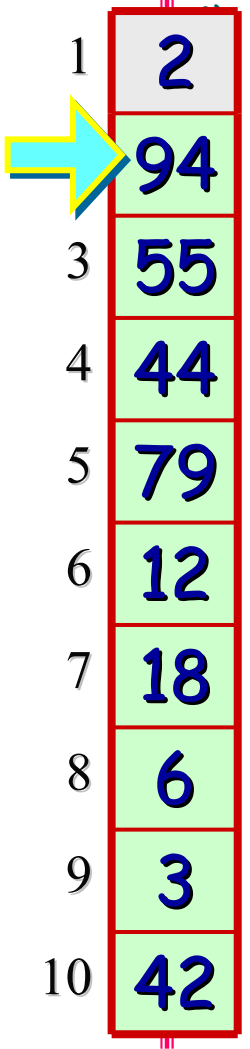
1	2
2	94
3	55
4	6
5	79
6	12
7	18
8	44
9	3
10	42

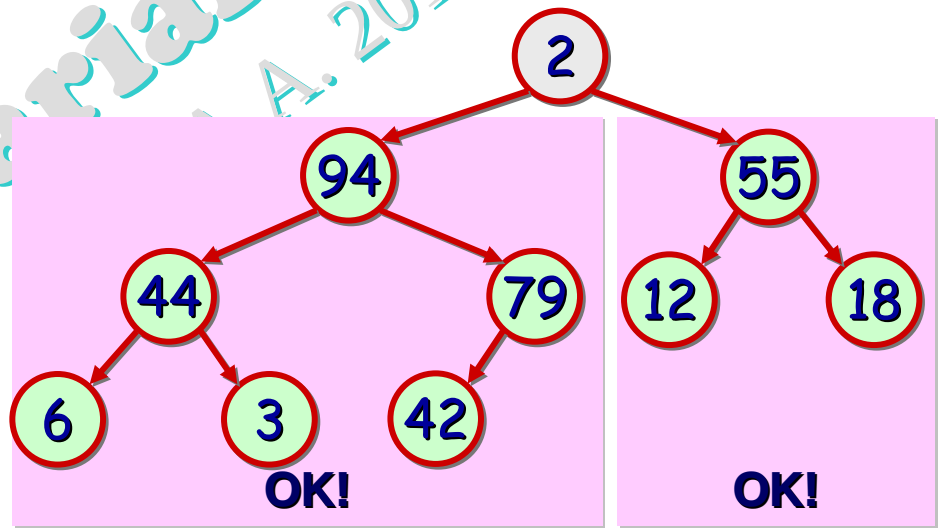

1 2
2 94
3 55
6
5 79
6 12
7 18
8 44
9 3
10 42
figlio_sx(a4) = a8
2
94
55
6
79
12
18
44
3
42
Heapify
2
94
55
44
79
12
18
6
3
42
OK!
OK!
1 2
94
3 55
4 44
5 79
6 12
7 18
6
9 3
10 42
Laurea triennale in Informatica
Programmazione I e Laboratorio di ...
prof. Mariarosaria Rizzardi
A.A. 2018/2019

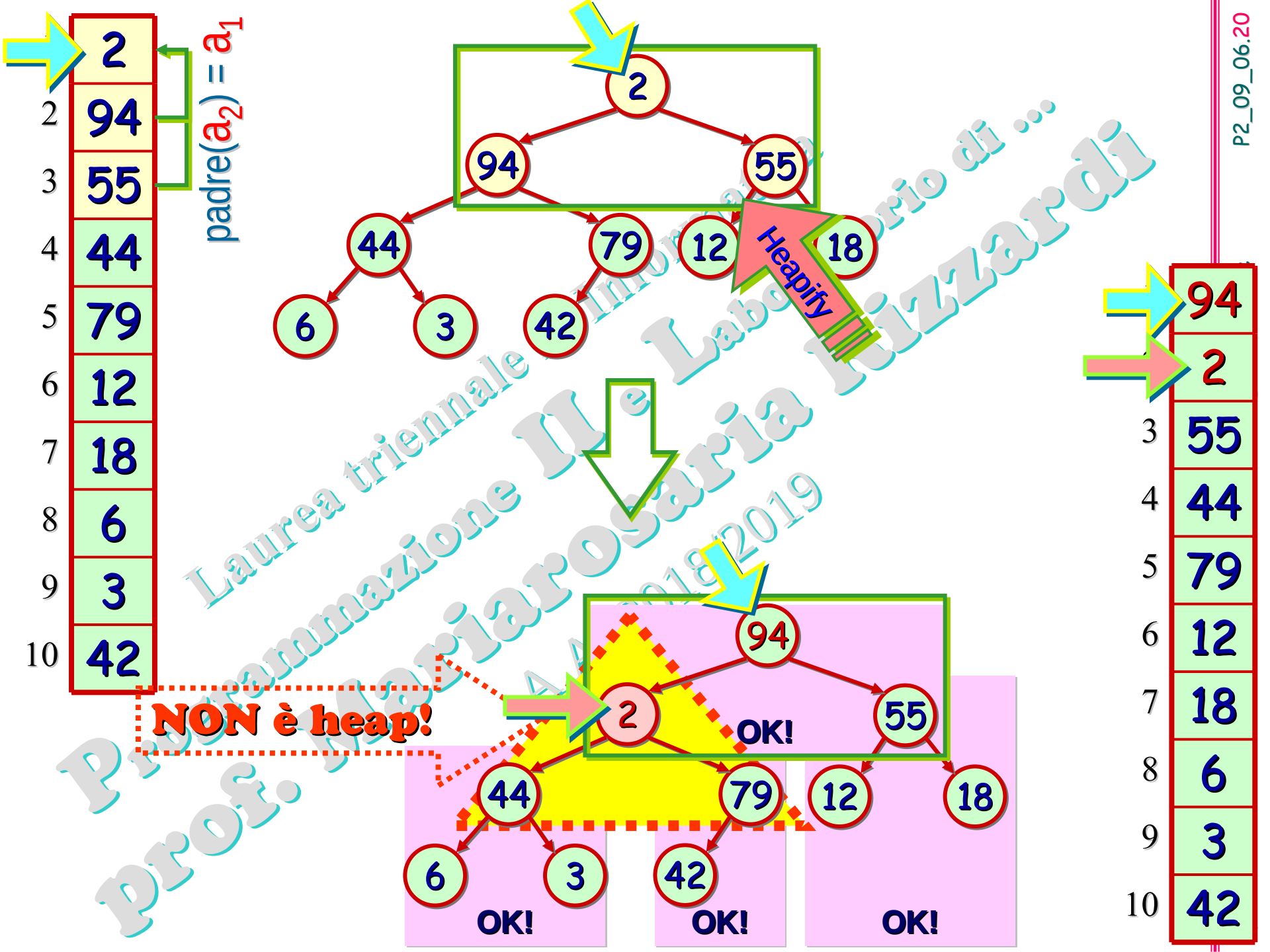


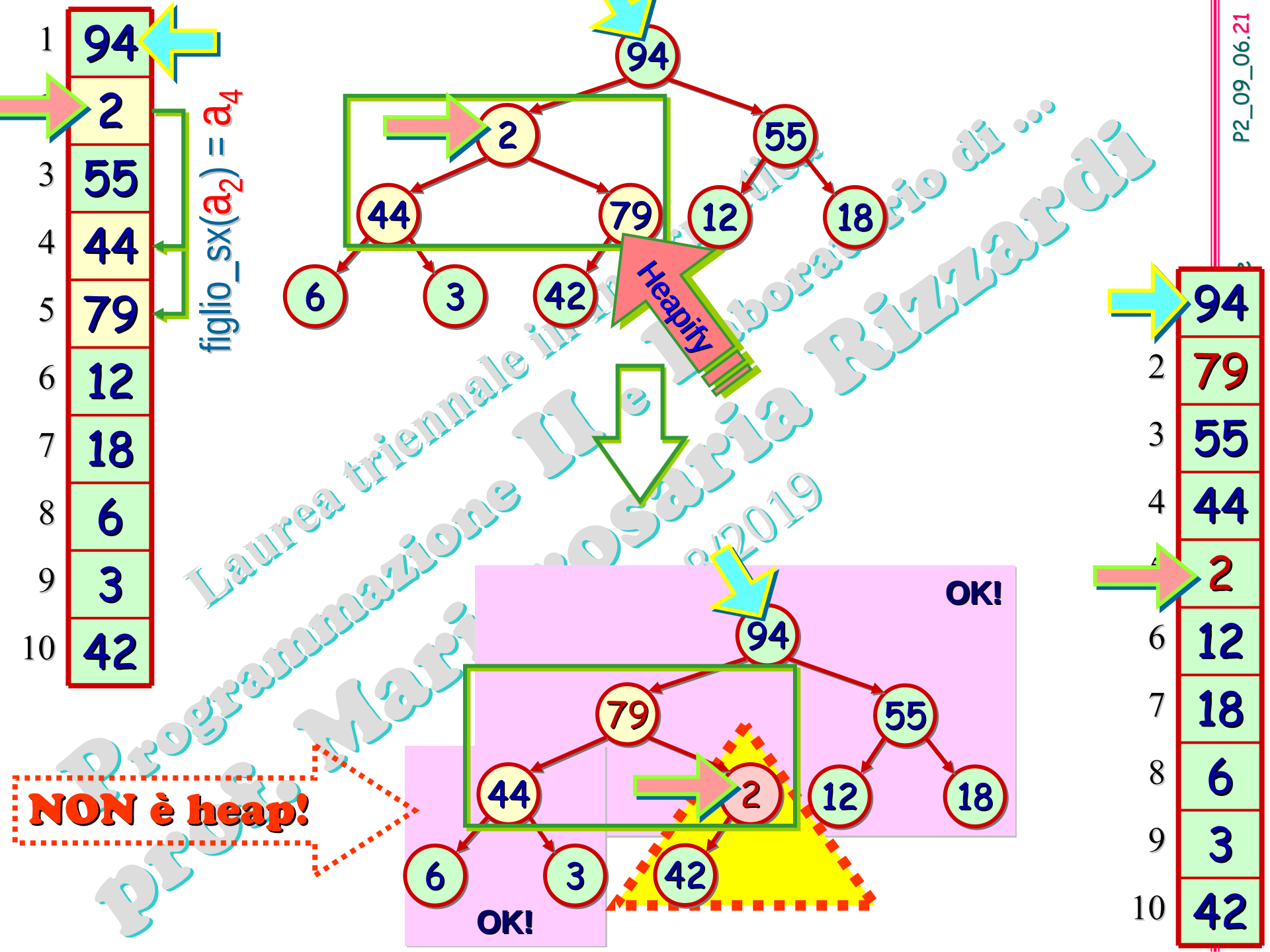


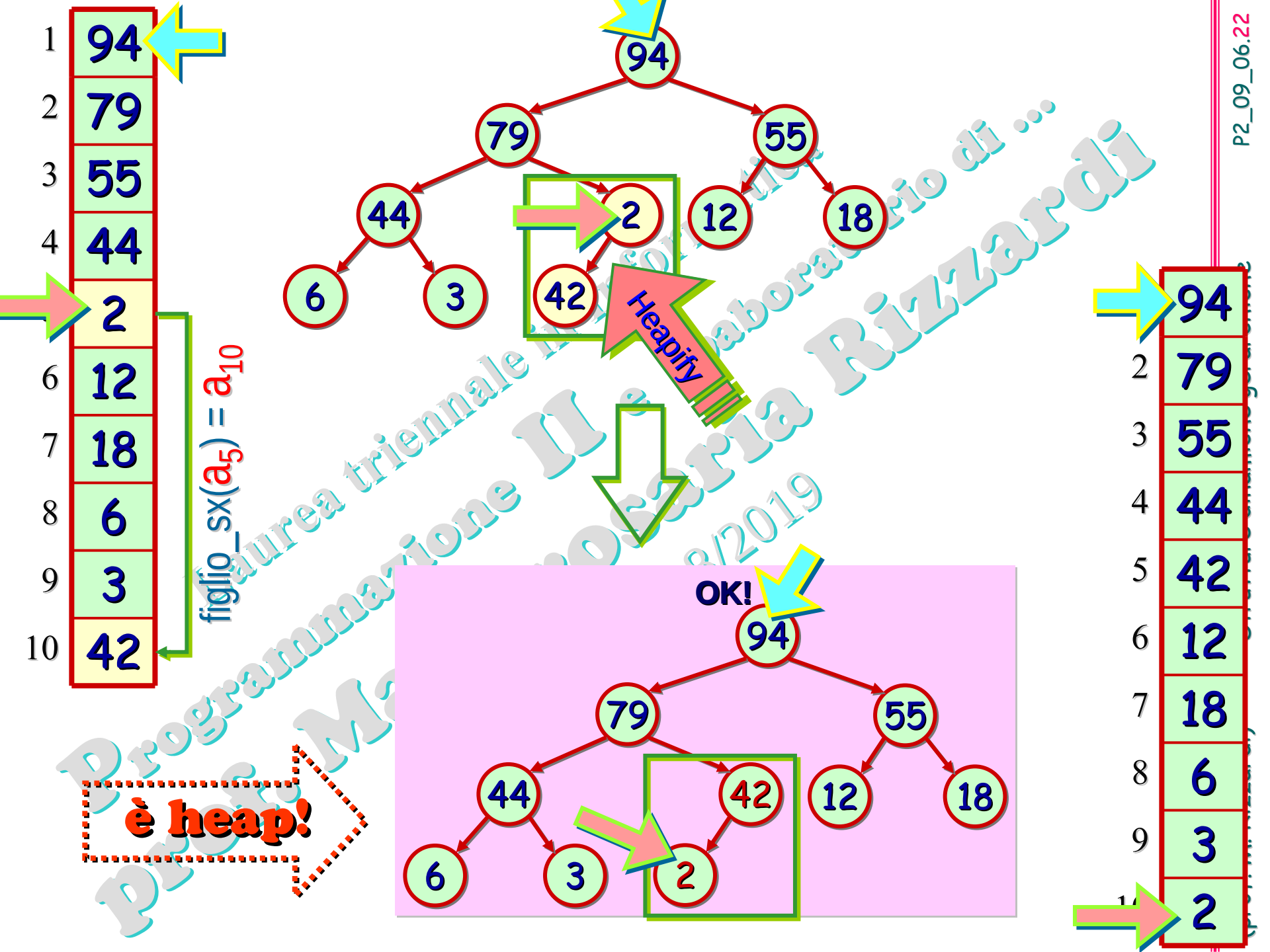
è heap!






2
94
55
44
79
12
18
6
3
42
padre(a_2) = a_1
Heapify
NON è heap!
OK!
OK!
OK!
OK!
94
2
55
44
79
12
18
6
3
42
Laurea triennale in Informatica
Programmazione II
prof. Mariarosaria Rizzardi
A.A. 2018/2019
Laboratorio di ...
P2_09_06.20


1 94
2 2
3 55
4 44
5 79
6 12
7 18
8 6
9 3
10 42
figlio_sx(a2) = a4
94
2
55
44
79
12
18
6
3
42
Heapify
NON è heap!
94
79
55
44
2
12
18
6
3
42
OK!
OK!
94
2 79
3 55
4 44
2
6 12
7 18
8 6
9 3
10 42
P2_09_06.21


1 94
2 79
3 55
4 44
2
6 12
7 18
8 6
9 3
10 42
figlio_sx(a5) = a10
94
79 55
44 2 12 18
6 3 42
Heapify
OK!
94
79 55
44 42 12 18
6 3 2
è heap!
94
2 79
3 55
4 44
5 42
6 12
7 18
8 6
9 3
2
P2_09_06.22

Esercizi

1 Scrivere *function* C iterativa per la costruzione di un *heap* rappresentato mediante array.

[liv. 2]

2 Scrivere *function C* iterativa per la trasformazione in un *heap* di un albero binario rappresentato mediante array.

[liv. 3]