 Ordina un array A.

A[5, 2, 4, 6, 1, 3]

2 5 4 6 1 3

2 4 5 6 1 3

2 4 5 6 1 3

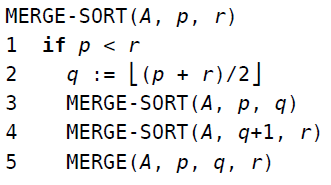
1 2 4 5 6 3

1 2 3 4 5 6

Costo: Θ(n) -> se l’array A è già ordinato (caso ottimo)

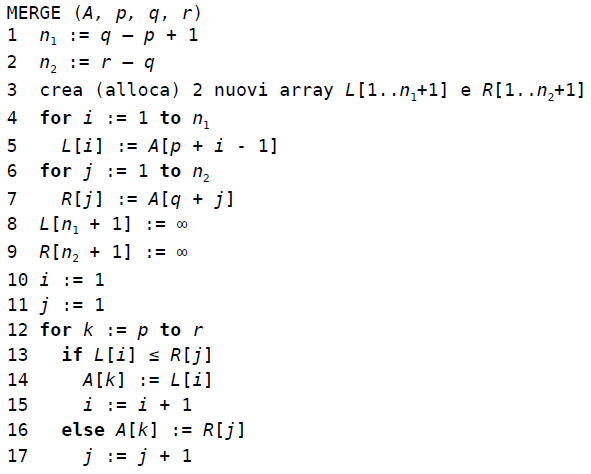
Θ(n2) -> se l’array A è ordinato in modo decrescente (caso pessimo)

-----------------------------------------------------------------------------------------------------------------------------------------------



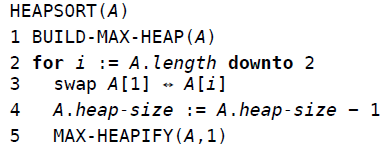
Ordina un array A in questo modo:

* Se l’array da ordinare ha meno di due elementi è ordinato.
* Si divide l’array in 2 sottoarray, ognuno con la metà degli elementi di quello originario.
* Si ordinano i 2 sotto array riapplicando l’algoritmo
* Si fondono i 2 sottoarray con il merge.

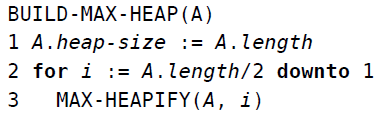
Costo: Θ(n log(n))

* Si va all’inizio dei 2 sottoarray
* Si prende il minimo dei due elementi correnti
* Si inserisce tale minimo alla fine dell’array da restituire
* Si avanza di uno nell’array di cui si è preso il minimo
* Si ripete dal secondo punto

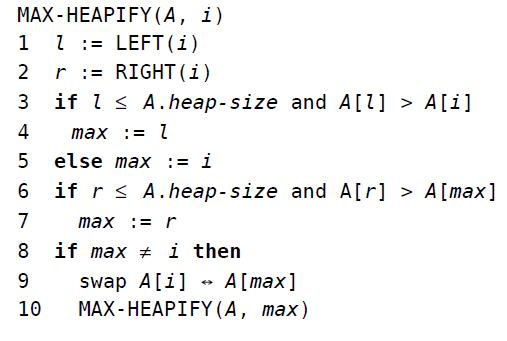
Costo: Θ(n)

* a ogni ciclo piazziamo l’elemento più grande (che è il primo dell’array, essendo un max-heap) in fondo alla parte di array ancora da ordinare (che è quella corrispondente allo heap)
* Lo heap si decrementa di 1 e si ricostruisce il max-heap mettendo come radice l’ultima foglia a destra dell’ultimo livello, e invocando max-heapify.

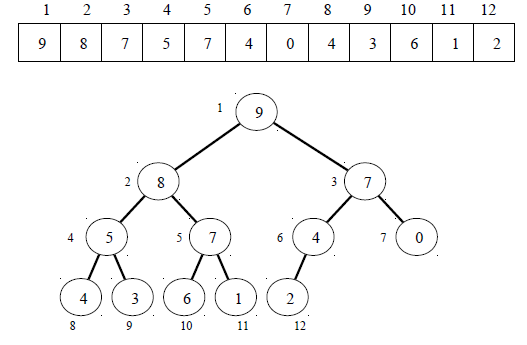
Costo: O(n log(n))

* costruiamo il max-heap bottom-up, dalle foglie fino ad arrivare alla radice.
* Tutti gli elementi da A.lenght/2 in poi sono foglie, quelli prima sono nodi interni

Costo: O(n log(n))

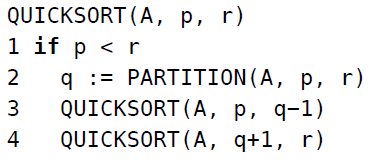
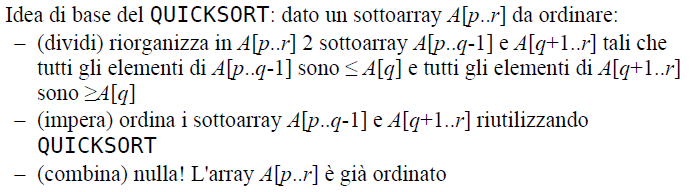
* modifica l’array in modo tale che tutto l’albero di radice sia un max-heap

Ecco un esempio di MaxHeap:

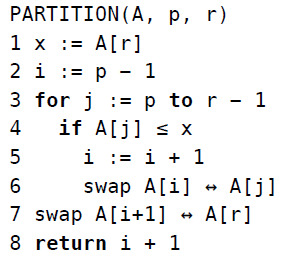


Costo: O(log(n))

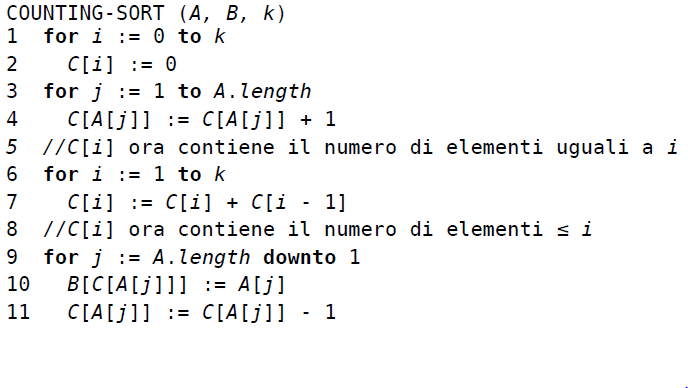
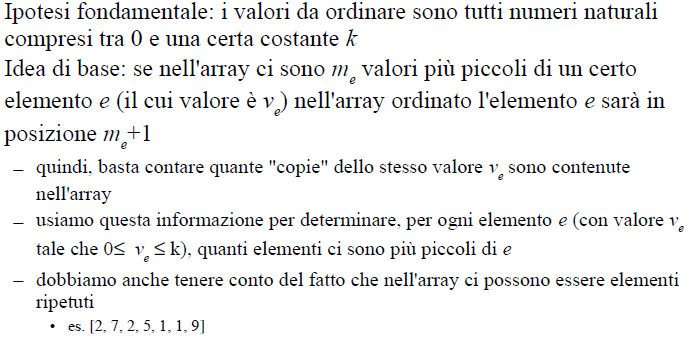
----------------------------------------------------------------------------------------------------------------------------------------------



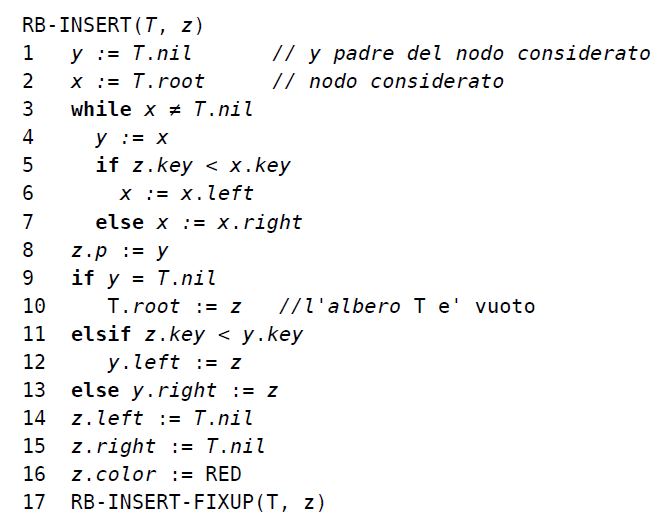
Costo: Θ(n log(n)) -> caso medio Θ(n2) -> caso pessimo

* Partiziona l’array in 2 parti
* X = A[r] è il pivot
* Da p a i (incluso) partizione con elementi ≤ x
* Da i +1 a j-1 partizione con elementi > x

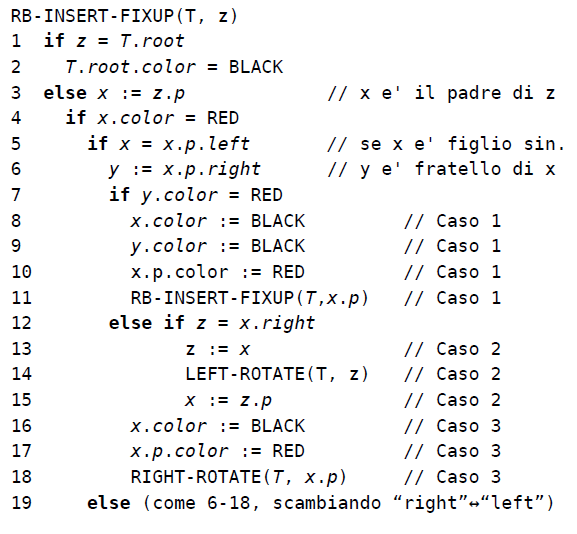
Costo: Θ(n)

 Costo: Θ(n + k)

-----------------------------------------------------------------------------------------------------------------------------------------------

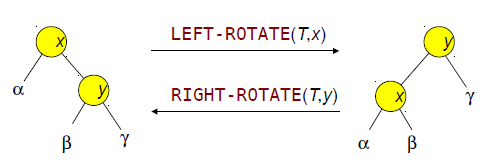


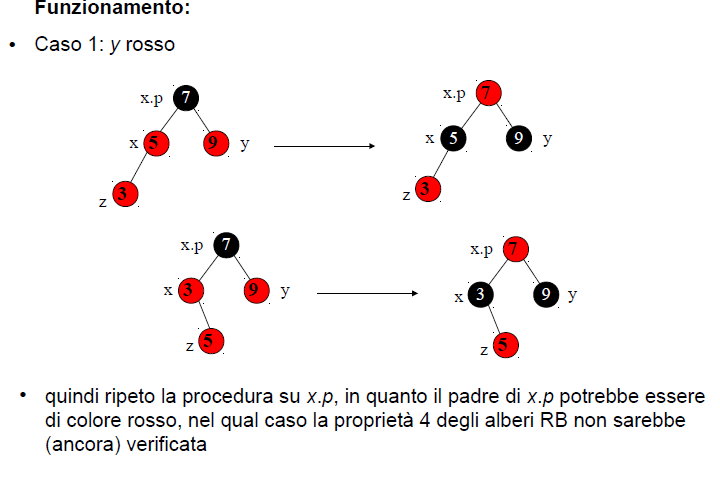
* Funziona come un tree-insert
* Dopo invoca il fixup per ristabilire le proprietà di un albero RB

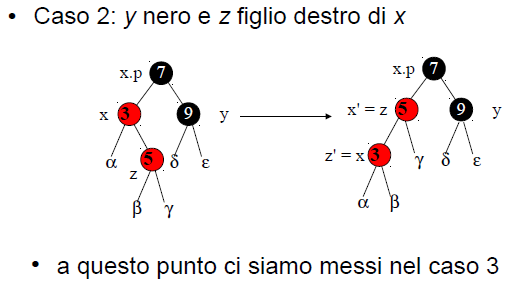


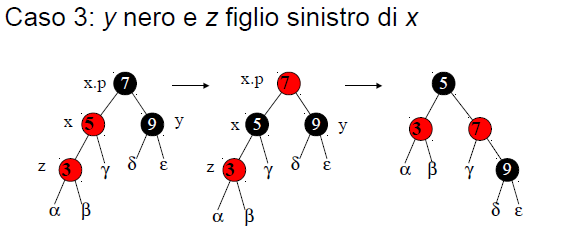
Invocato sempre su un nodo z tale per cui z.color=red

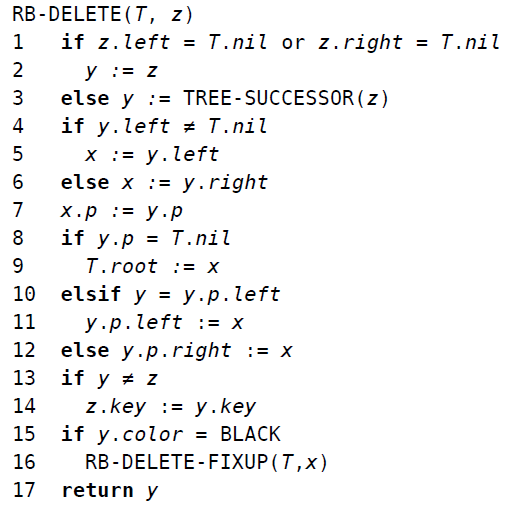
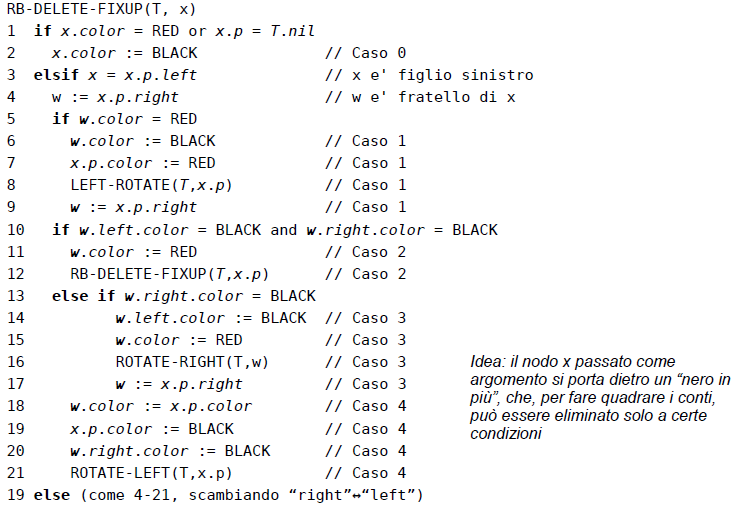
Costo: O(log(n))



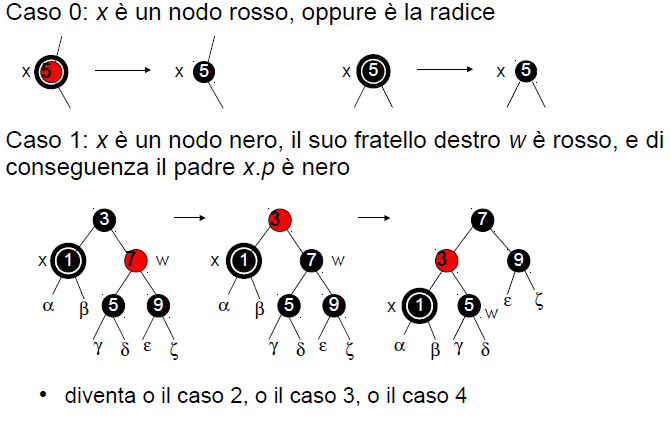
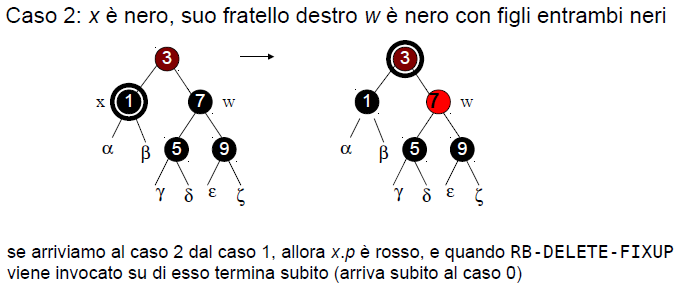


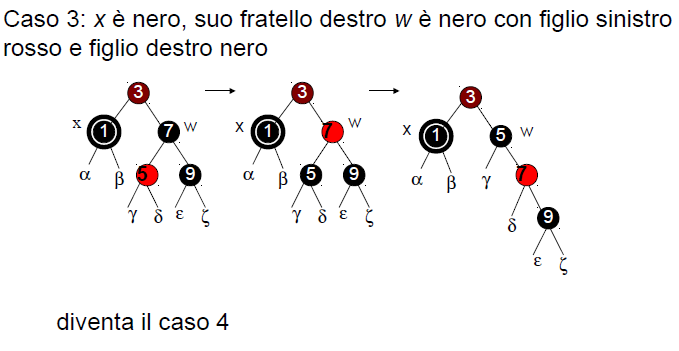
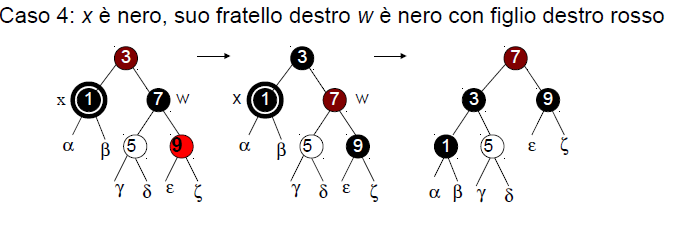


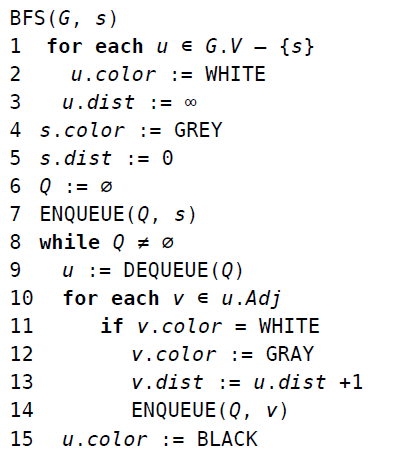
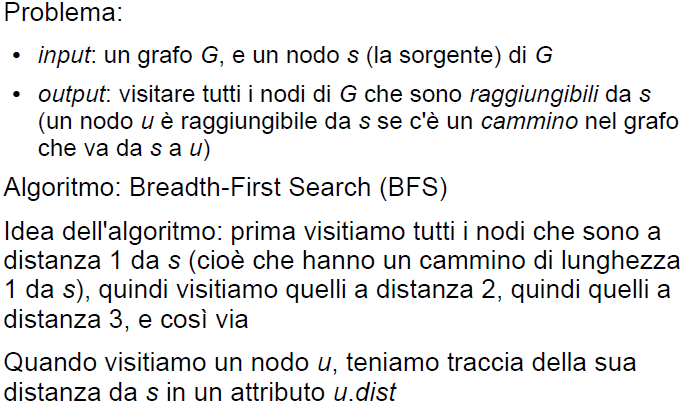




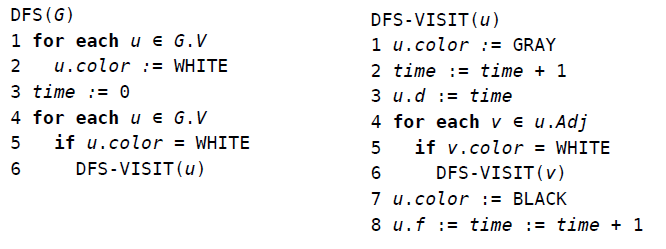
Costo: O(log(n))





-----------------------------------------------------------------------------------------------------------------------------------------------

Costo: O(V + E)



Costo: Θ(V + E)

