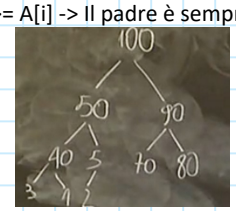
Heap sort intro

martedì 7 giugno 2022 14:04

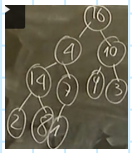
- O(nlogn)
- Non è stabile
- In loco
- Utile per:
- Heap = array visto come un albero binario quasi completo
  -> Si riempie da sinistra verso destra

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
heapSize(A) = quante caselle dell'array fanno parte dell'heap <= len(A)
A[Parent(i)] >= A[i] -> Il padre è sempre più grande dei figli

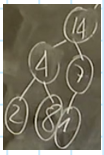


La posizione di un padre dato un figlio è $\frac{i}{2}$
Il figlio sinistro è $2 * i$
Il figlio destro è $2 * i + 1$
Altezza = log(n)
Max = Radice
Max2 = Uno dei due figli della radice
Max3 = Potrebbe essere ovunque
Minimo = Su una foglia
Numero foglie = $\frac{n}{2}$
Utile per: heapSort, coda di priorità
Heapify(A, h)

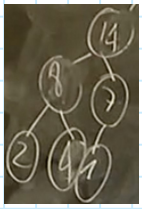
- Ordina un sottoalbero e nota, ordina ogni nodo modificato



Heapify(A, 2)
4 > 14 ? No, quindi scambia



Heapify(A, 4)
4 > 2? Si, non fare niente
4 > 8? No, scambia



Heapify(A, 9)
4 è una foglia
Non fare niente

```
Heapify(A, h):
        Largest = h
        L = left(h) # 2*h
        R = right(h) # 2*h+1

        # Cerchiamo il più grande fra il parent ed i figli
        If a[L] > A[largest] and L <= heapsize(A):
                Largest = L
        If a[r] > A[largest] and r <= heapsize(A):
                Largest = r

        If largest != h:
```

App = A[largest]
A[largest] = a[h]
A[h] = app
Heapify(A, largest)

T(n) = $t\left(2\frac{n}{3}\right) + O(1)$
F(n) = O(1)
R(N) = $n^{\log_b a} = n^{\log_3 1} = n^0 = 1$
$f(n) = r(n)$
$t(n) = O(1 * \ln n)$

E se non avessimo uno heap, come facciamo a chiamare heapify?
Chiamiamo heapify facendolo andare dalle foglie verso la radice
Questo è il buildheap

Buildheap(A):
    Heapsize(A) = len(A)
    For h = $\frac{n}{2}$ down to 1
        Heapify(A, h)