Appunti principali sul Heap

- L'Heap è un albero binario completo tranne per l'ultimo livello, tutti gli elementi dell'ultimo livello vengono inseriti da sinistra a destra

Heapify (x) funzione che ristabilisce le proprietà del Heap

```
l ← LEFT()
r ← RIGHT()
max ← x
IF ( l ≠ NULL AND l.KEY ≥ x.KEY)
  MAX ← l
IF ( r ≠ NULL AND r.KEY ≥ MAX.KEY)
  MAX ← r
IF (MAX =! x)
  SWAP (MAX, x)
HEAPIFY(MAX)
```

RICORDIAMO CHE:

```
left(x)
return 2·x

right(x)
return 2·x+1

parent(x)
return ⌊x/2⌋
```

Questa procedura ha un costo $O(\log n)$

- Bisogna ricordare che l'Heap presenta una proprietà d'ordinamento parziale, in base a se è MAX-HEAP o MIN-HEAP il padre risulta essere sempre l'elemento maggiore o minore.

- L'Heap solitamente viene implementato con l'uso di un array

- Per costruire un heap partendo da un array usiamo la buil-heap (A)

```
BUILD-HEAP (A)
```

```
Heap_size ← Arr.lenght
i ← ⌊Arr.lenght/2⌋
FOR i DOWN TO 1
  Heapify(Arr[i])
  i ← i-1
```

Questa procedura richiede $O(n \log n)$

- Oltre queste operazioni possiamo implementarne altre:

```
INSERT (A, x)
HEAPSIZE ← HEAPSIZE + 1
A[HEAPSIZE] ← x
i ← HEAPSIZE
WHILE (i > 1 AND A[i] < A[PARENT(i)])
  SWAP(A[i], A[PARENT(i)])
  i ← PARENT(i)
```

Costo $O(\log n)$

Serve a inserire nuovi valori nell'array

- EXTRACT: Serve per estrarre il minimo o il massimo
  Scambia il primo e l'ultimo elemento;
  decrementa HEAPSIZE e chiama HEAPIFY sulla radice

- DELETE: Procedura per eliminare un elemento
  Uguale a EXTRACT, ma si chiama su un generico modo

- DECREASE-KEY o INCREASE-KEY diminuisce o incrementa il valore di una chiave esistente

```
DECREASE-KEY (A, i, K)
A[i] ← K
WHILE (i > 1 AND A[i] < A[parent(i)])
```

```
    SWAP( A[i], A[PARENT(i)])
    i ← PARENT(i)
```

Con queste strutture dati si utilizza un algoritmo d'ordinamento in loco con costo $O(n \log n)$ chiamato HEAPSORT

```
HEAPSORT (A)
 HEAPSIZE ← A.lenght
 BUILD HEAP(A)
 FOR i ← 1 TO HEAPSIZE - 1
  EXTRACT()
```

Costo $(n \log n)$