Università di Pisa

Pietro Ducange

Algoritmi e strutture dati

Heap e HeapSort

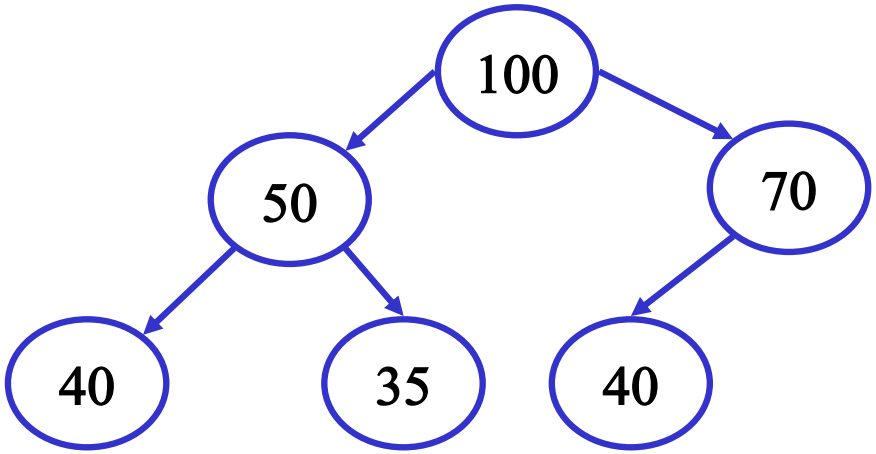
a.a. 2020/2021

Si ringrazia la prof. Nicoletta De Francesco per aver messo a disposizione la maggior parte delle slide utilizzate nella presente lezione

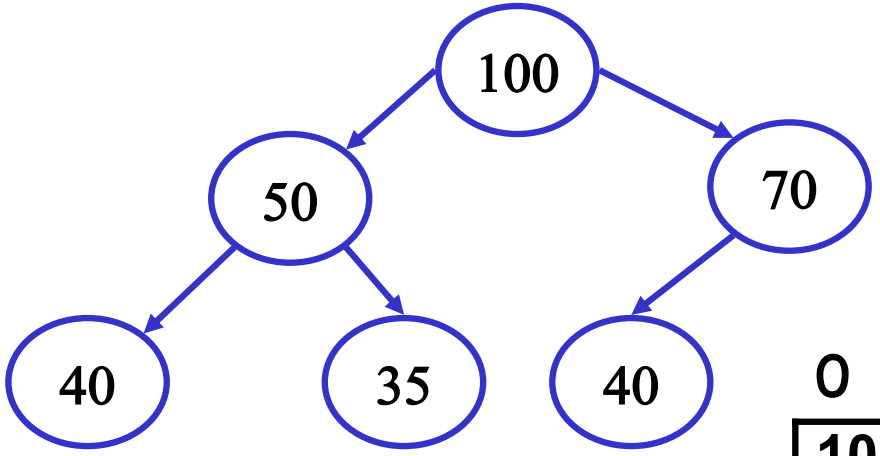
Heap: definizione

Heap: albero binario quasi bilanciato con le proprietà:

- **i nodi dell'ultimo livello sono addossati a sinistra**
- **in ogni sottoalbero l'etichetta della radice é maggiore o uguale a quella di tutti i discendenti.**



Heap: memorizzazione in array



0	1	2	3	4	5	6	7
100	50	70	40	35	40		

figlio sinistro di i : 2i+1

figlio destro di i : 2i+2

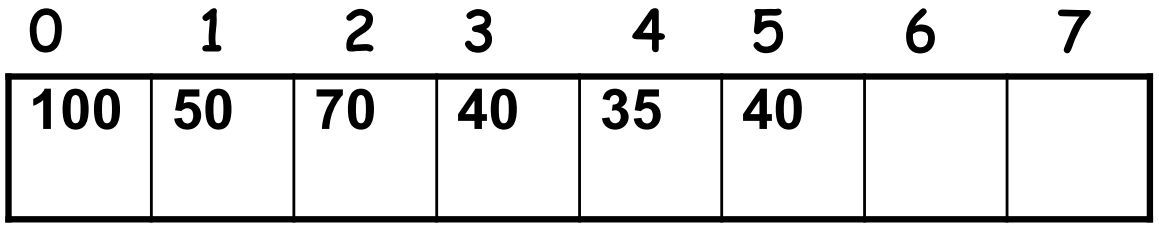
padre di i : (i-1)/2

OPERAZIONI

- **inserimento** di un nodo
- **estrazione** dell'elemento maggiore (radice)

Classe Heap

```
class Heap {
int * h;
int last;    //indice dell'ultimo elemento
void up(int);
void down(int);
void exchange(int i, int j){
        int k=h[i]; h[i]=h[j];h[j]=k;
}
public:
Heap(int);
~Heap();
void insert(int);
int extract();
};
```



last=5

Heap: costruttore e distruttore

```
Heap::Heap(int n){
        h=new int[n];
        last=-1;
}

Heap::~Heap() {
        delete [] h;
        }
```

Heap: inserimento

- memorizza l'elemento nella prima posizione libera dell'array
- fai risalire l'elemento tramite scambi figlio-padre per mantenere la proprietà dello heap

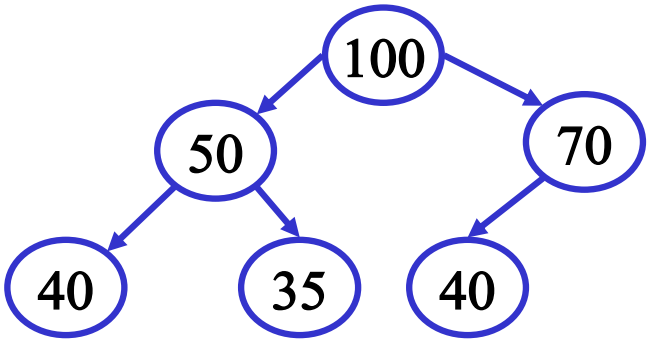
```
void Heap::insert (int x) {
   h[++last]=x;
   up(last);
}
```

Heap: inserimento funzione up

```
void Heap::up(int i) {   // i è l'indice dell'elemento da far risalire
    if (i > 0)           // se non sono sulla radice
   if (h[i] > h[(i-1)/ 2]) {   // se l'elemento è maggiore del padre
     exchange(i,(i-1)/2);      // scambia il figlio col padre
     up((i-1)/2);              // e chiama up sulla nuova posizione
   }                           // altrimenti termina
}
```

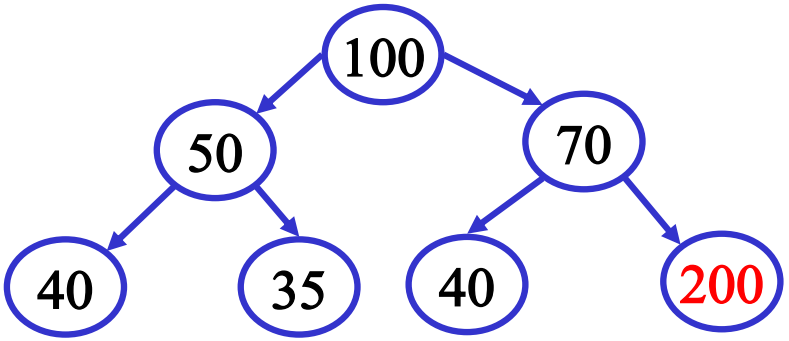
- la funzione termina o quando viene chiamata con l'indice 0 (radice) o quando l'elemento è inferiore al padre
- La complessità è O(logn) perchè ogni chiamata risale di un livello

Heap: esempio di inserimento



0	1	2	3	4	5	6	7
100	50	70	40	35	40		

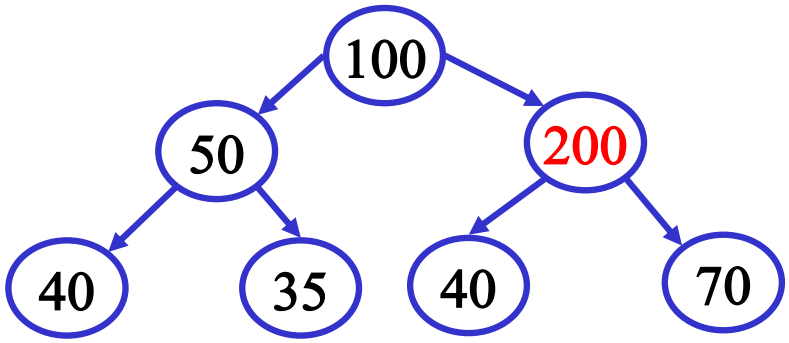
insert(200)



0	1	2	3	4	5	6	7
100	50	70	40	35	40	200	

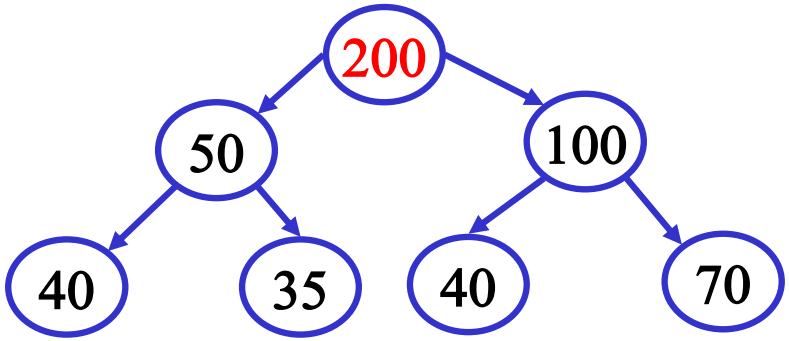
Heap: esempio di inserimento

up(6)



0	1	2	3	4	5	6	7
100	50	200	40	35	40	70	

up(2)



0	1	2	3	4	5	6	7
200	50	100	40	35	40	70	

up(0)

Heap: estrazione

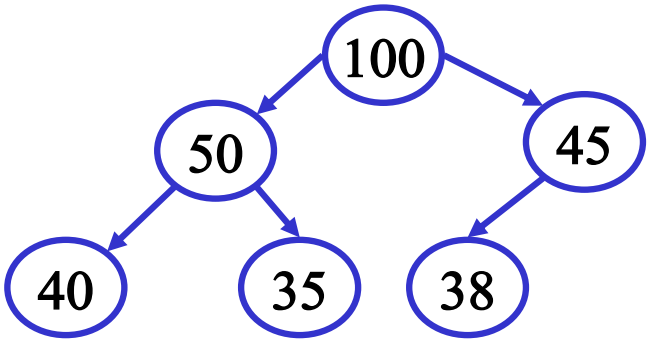
- restituisci il primo elemento dell'array
- metti l'ultimo elemento al posto della radice e decrementa last
- fai scendere l'elemento tramite scambi padre-figlio per mantenere la proprietà dello heap

```
int Heap::extract() {
    int r=h[0];
    h[0]=h[last--];
    down(0);
    return r;
 }
```

Heap: estrazione funzione down

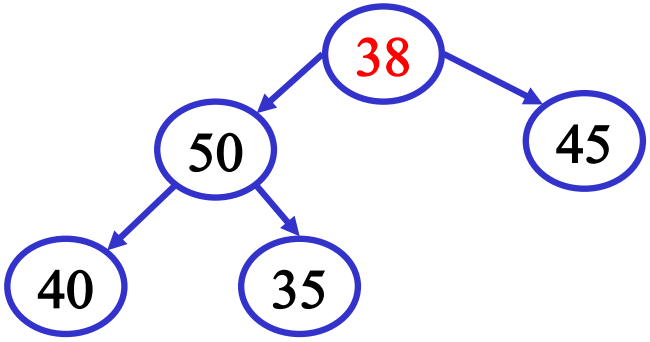
```
void Heap::down(int i) {    // i è l'indice dell'elemento da far scendere
   int son=2*i+1;           // son = indice del figlio sinistro (se esiste)
   if (son == last) {       // se i ha un solo figlio (è l'ultimo dell'array)
     if (h[son] > h[i])     // se il figlio è maggiore del padre
       exchange(i,last);    // fai lo scambio, altrimenti termina
   }
   else if (son < last) {                  // se i ha entrambi I figli
     if (h[son] < h[son+1]) son++; // son= indice del maggiore fra i due
     if (h[son] > h[i]) {          // se il figlio è maggiore del padre
       exchange(i,son);            // fai lo scambio
       down(son);                  // e chiama down sulla nuova posizione
     }       // altrimenti termina (termina anche se i non ha figli)
   }
}                    complessità : O(log n)
```

Heap: esempio di estrazione



0	1	2	3	4	5	6	7
100	50	45	40	35	38		

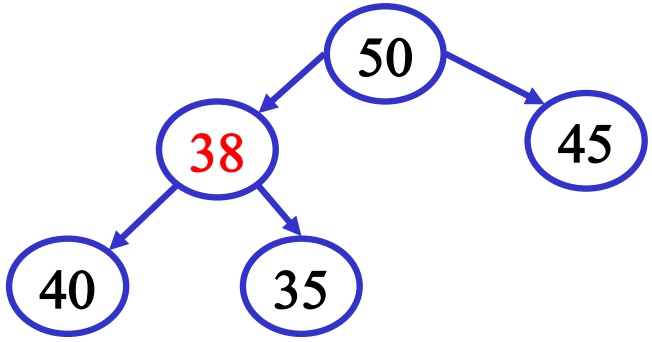
extract() -> 100



0	1	2	3	4	5	6	7
38	50	45	40	35	38		

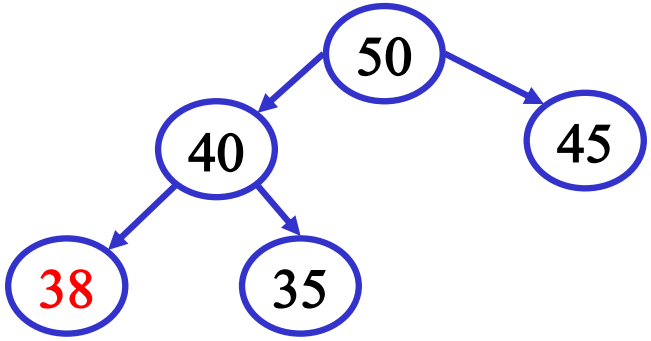
Heap: esempio di estrazione

down(0)



0	1	2	3	4	5	6	7
50	38	45	40	35	38		

down(1)



0	1	2	3	4	5	6	7
50	40	45	38	35	38		

down(3)

Quando si usa lo heap?

Lo heap è particolarmente indicato per l'implementazione del tipo di dato astratto ***coda con priorità***.

Si tratta di una coda in cui gli elementi contengono, oltre all'informazione, un intero che ne definisce la priorità.

In caso di estrazione, l'elemento da estrarre deve essere quello con maggiore priorità.

Algoritmo di ordinamento Heapsort

- trasforma l'array in uno heap (buildheap)
- esegui n volte l'estrazione scambiando ogni volta il primo elemento dell'array con quello puntato da last

```
void heapSort(int* A, int n) {// n dimensione
dell'array

   buildHeap(A,n-1);                    // O(n)

   int i=n-1;

   while (i > 0) {                      // O(nlogn)

                extract(A,i);

        }
                          O(nlogn)
}
```

down modificata

```
void down(int * h, int i, int last) {
   int son=2*i+1;
   if (son == last) {
     if (h[son] > h[i]) exchange(h, i,last);
   }
   else if (son < last) {
     if (h[son] < h[son+1]) son++;
     if (h[son] > h[i]) {
       exchange(h, i,son);
       down(h, son, last);
     }
   }
}
```

O(logn)

I parametri sono l'array, l'indice dell'elemento da far scendere, l'ultimo elemento dello heap

Estract modificata

```
void extract(int* h, int & last) {
        exchange(h, 0,last--);
        down(h, 0, last);
}
```

- I parametri sono l'array e l'ultimo elemento dello heap
- L'ultimo elemento viene scambiato con il primo
- Non si restituisce nulla

O(logn)

Trasforma l'array in uno heap (buildheap)

- Esegui la funzione down sulla prima metà degli elementi dell'array (gli elementi della seconda metà sono foglie)
- Esegui down partendo dall'elemento centrale e tornando indietro fino al primo

```
void buildHeap(int* A, int n) { //n+1 è la dimensione dell'array
 for (int i=n/2; i>=0; i--) down(A,i,n);
}
```

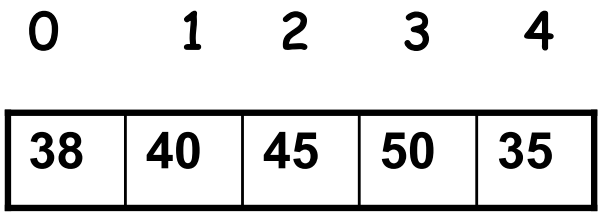
O(n)

Heapsort

```
void down(int * h, int i, int last) {
     int son=2*i+1;
     if (son == last) {
         if (h[son] > h[i]) exchange(h, i,last);
     }
     else if (son < last) {
         if (h[son] < h[son+1]) son++;
         if (h[son] > h[i]) {
             exchange(h, i,son);
             down(h, son, last);
         }
     }
}

void extract(int* h, int & last) {
     exchange(h, 0,last--);
     down(h, 0, last);
}

void buildHeap(int* A, int n) {
   for (int i=n/2; i>=0; i--) down(A,i,n);
}

void heapSort(int* A, int n) {
   buildHeap(A,n-1);
   int i=n-1;
   while (i > 0) extract(A,i);
}
```

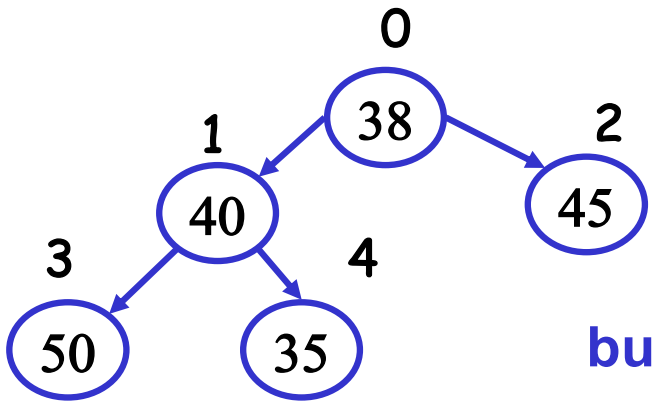
Esempio di heapsort

heapSort(A, int 5)



A

Esempio di heapsort: buildHeap



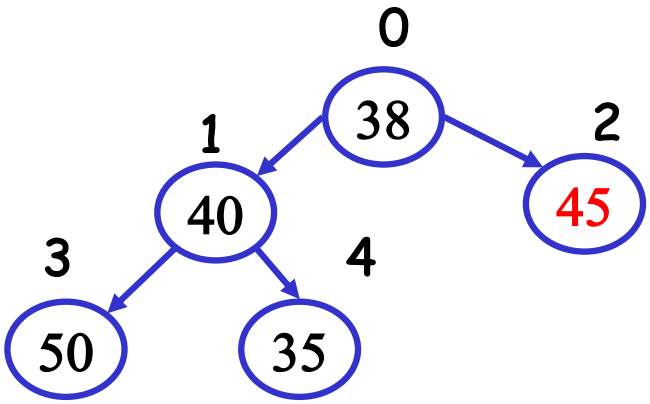
0	1	2	3	4
38	40	45	50	35

A

buildHeap(A, 4)

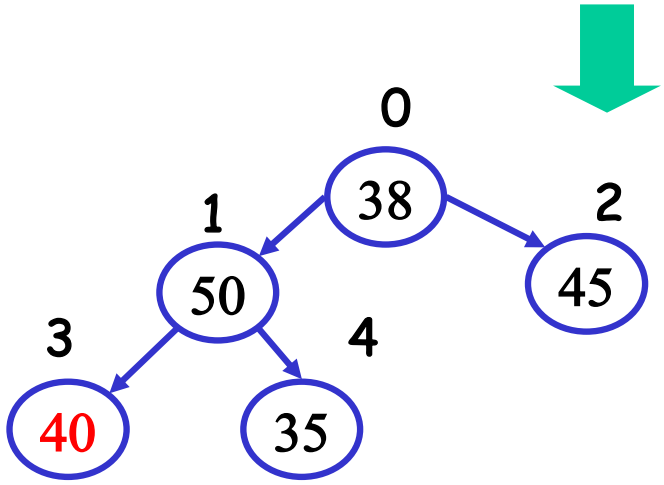
```
void buildHeap(int* A, int n) {

for (int i=n/2; i>=0; i--) down(A,i,n);}
```

down(A,2,4);



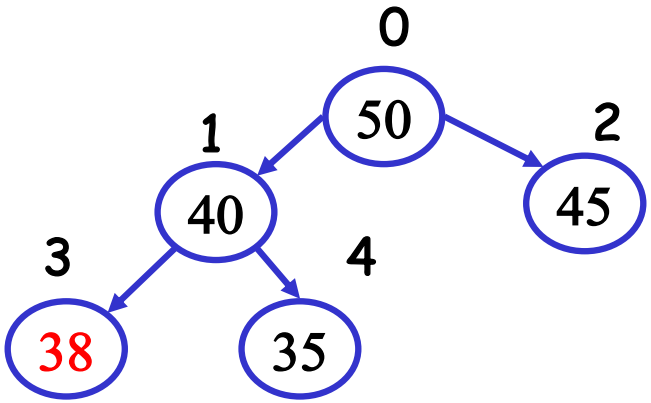
0	1	2	3	4
38	40	45	50	35

Esempio di heapsort: buildHeap



down(A,1,4);

0	1	2	3	4
38	50	45	40	35

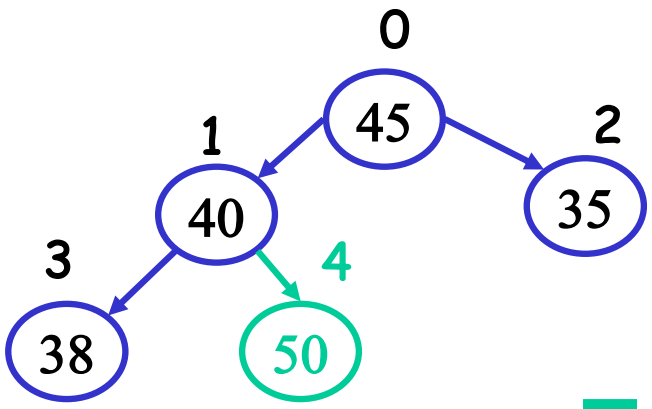


down(A,0,4);

0	1	2	3	4
50	40	45	38	35

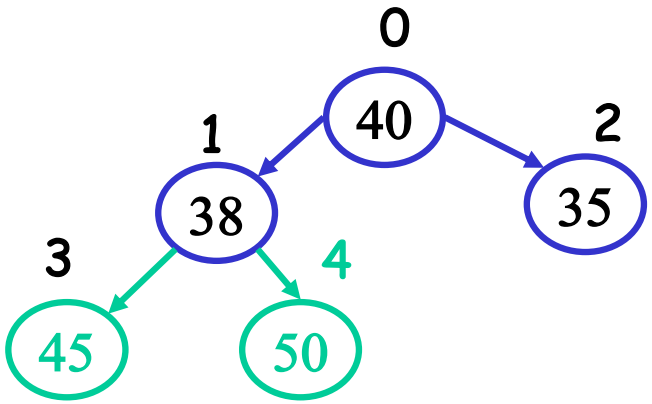
Esempio di heapsort: estrazioni

extract(A,4);



0	1	2	3	4
45	40	35	38	50

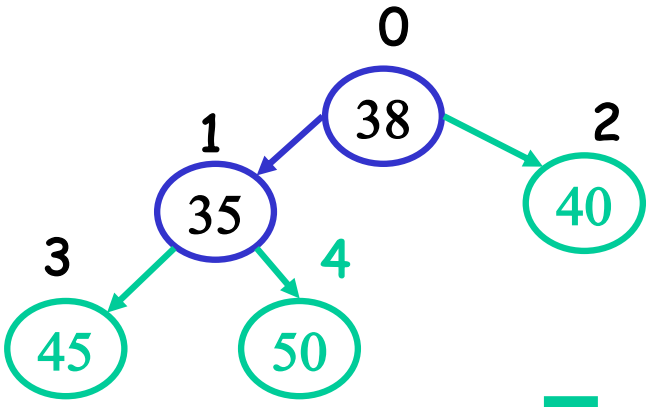
extract(A,3);



0	1	2	3	4
40	38	35	45	50

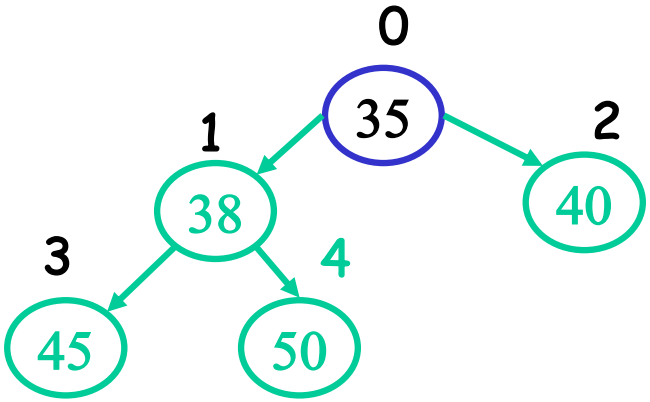
Esempio di heapsort: estrazioni

extract(A,2);



0	1	2	3	4
38	35	40	45	50

extract(A,1);



0	1	2	3	4
35	38	40	45	50

Un Video Interessante

https://www.youtube.com/watch?v=2DmK_H7IdTo&ab_channel=MichaelSambol

Riferimenti Bibliografici

Demetrescu:

Paragrafo 4.3

Cormen:

Capitolo 6

Esercizio I

Dato lo heap: [100, 90, 80, 70, 80, 50, 20, 10]

Indicare lo heap dopo una estrazione e il successivo inserimento del nodo 85. Indicare le chiamate a up e down

Esercizio II

Scrivere una funzione C++ che stampi in ordine simmetrico gli elementi di uno heap

```
void inorder(int * A, int i, int last)
```

Prima chiamata: `inorder (A, 0, last)`