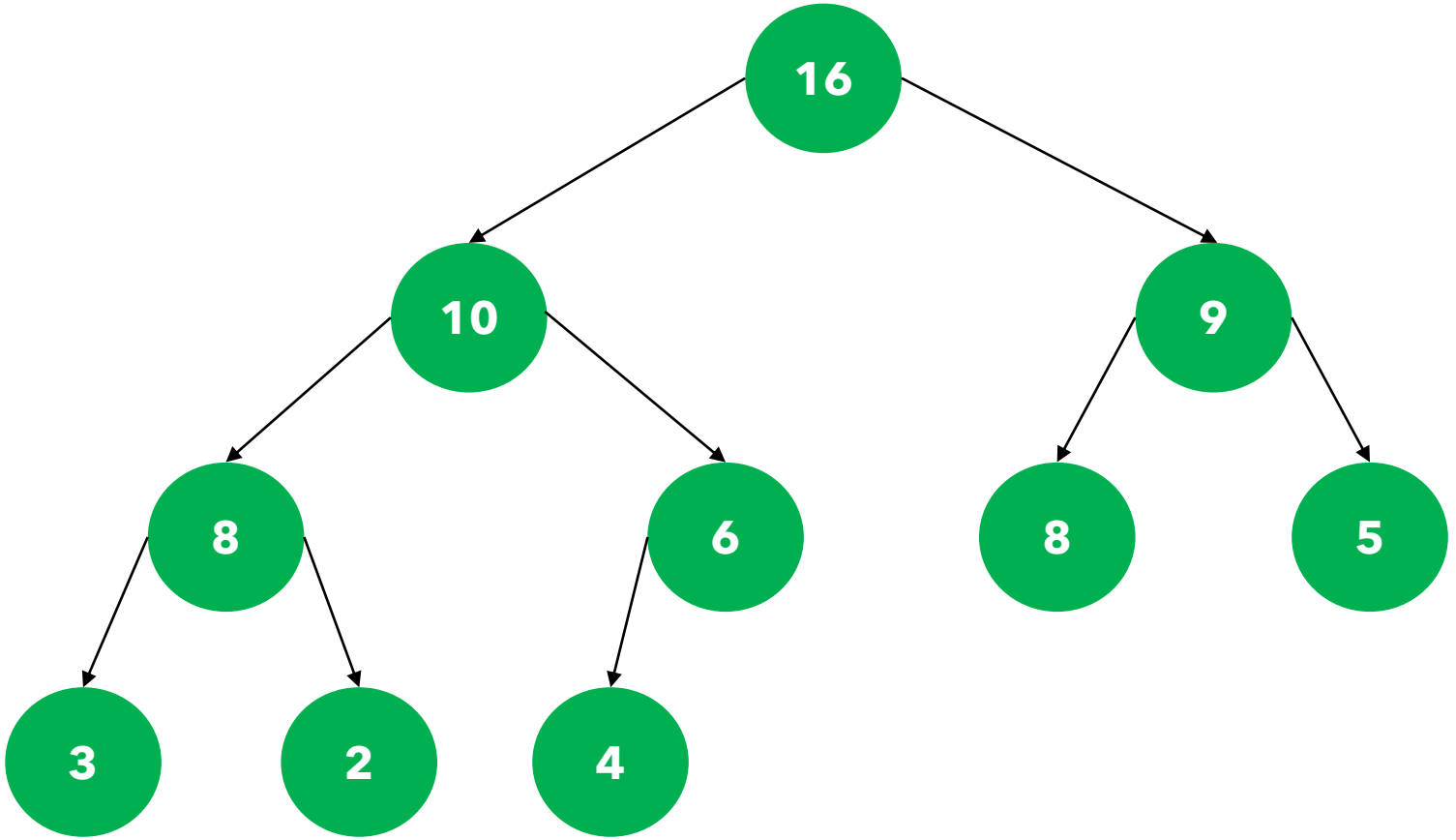
Heapsort

Liceo G.B. Brocchi – Bassano del Grappa (VI)
Liceo Scientifico – opzione scienze applicate
Giovanni Mazzocchin

Heap – albero binario



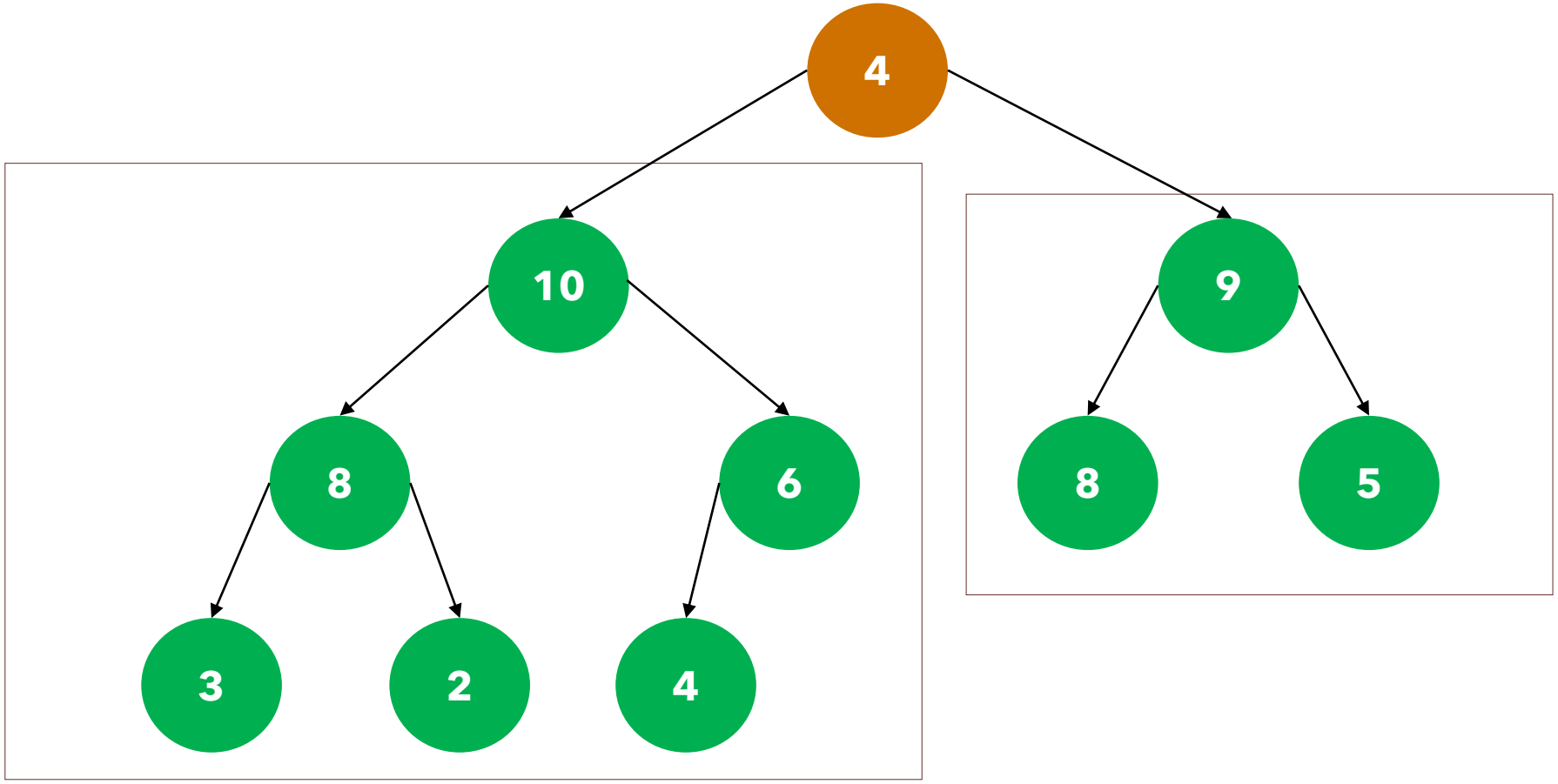
Max-heap property:
per ogni nodo n di un max heap, i suoi figli contengono nodi con chiavi minori o uguali alla chiave di n

Heap - array

values	16	10	9	8	6	8	5	3	2	4
indexes	1	2	3	4	5	6	7	8	9	10

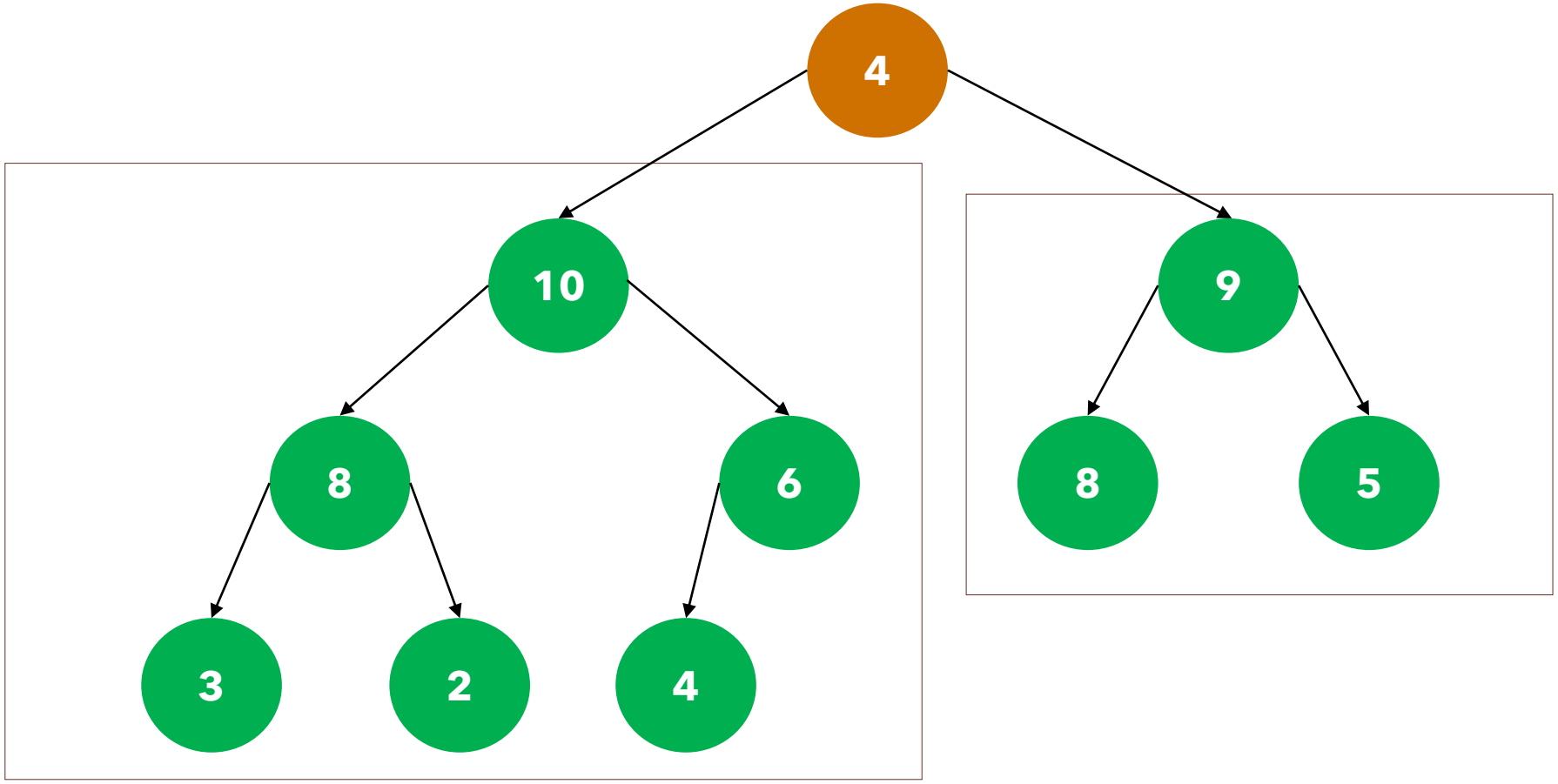
- `left(i): 2 * i`
- `right(i): 2 * i + 1`
- `A[left(i)]` è il figlio sinistro di `A[i]`
- `A[right(i)]` è il figlio destro di `A[i]`

max_heapify – esempio 1



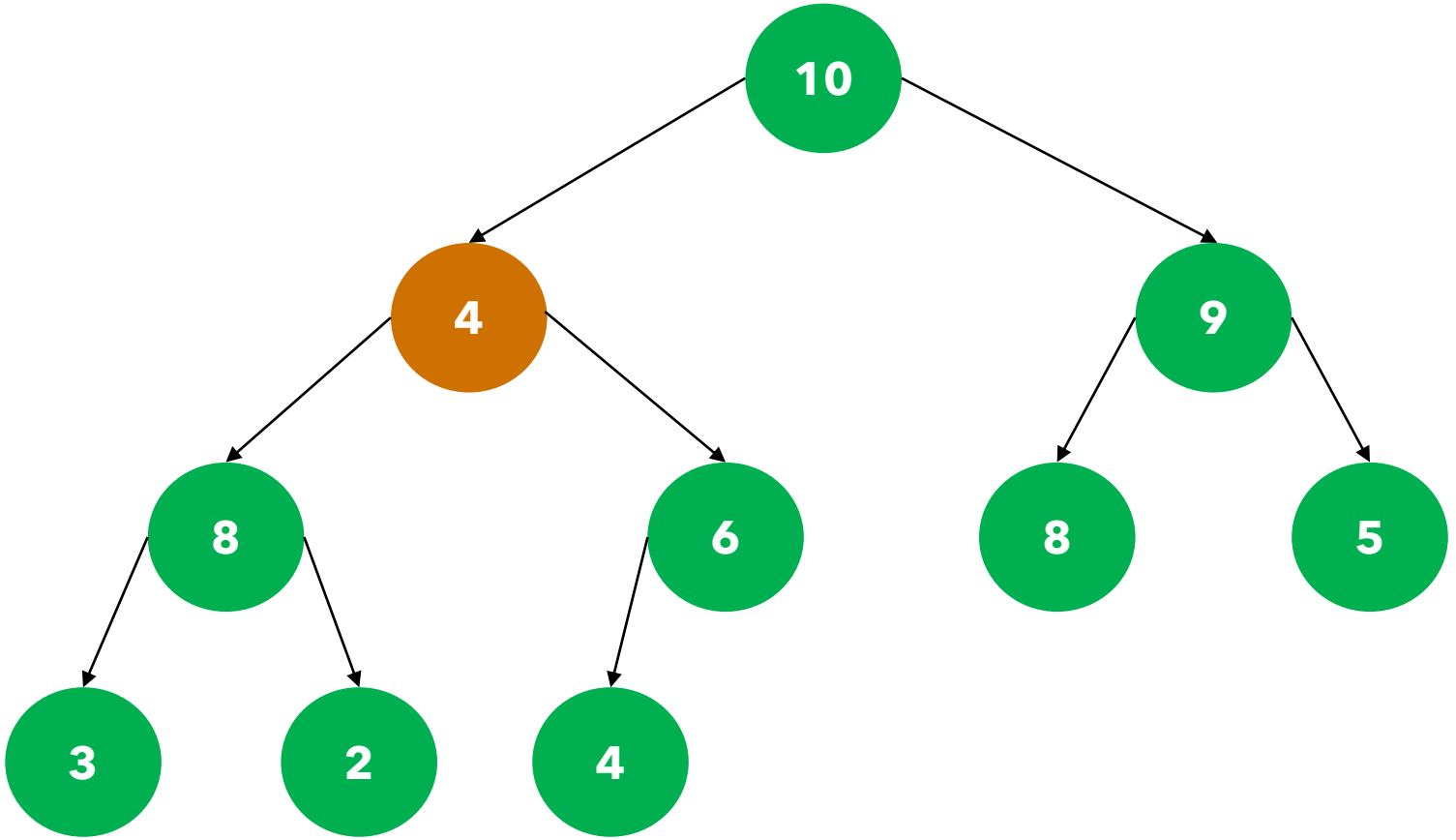
questo non è un max heap, ma i sottoalberi del nodo arancio sono 2 max heap

max_heapify – esempio 1



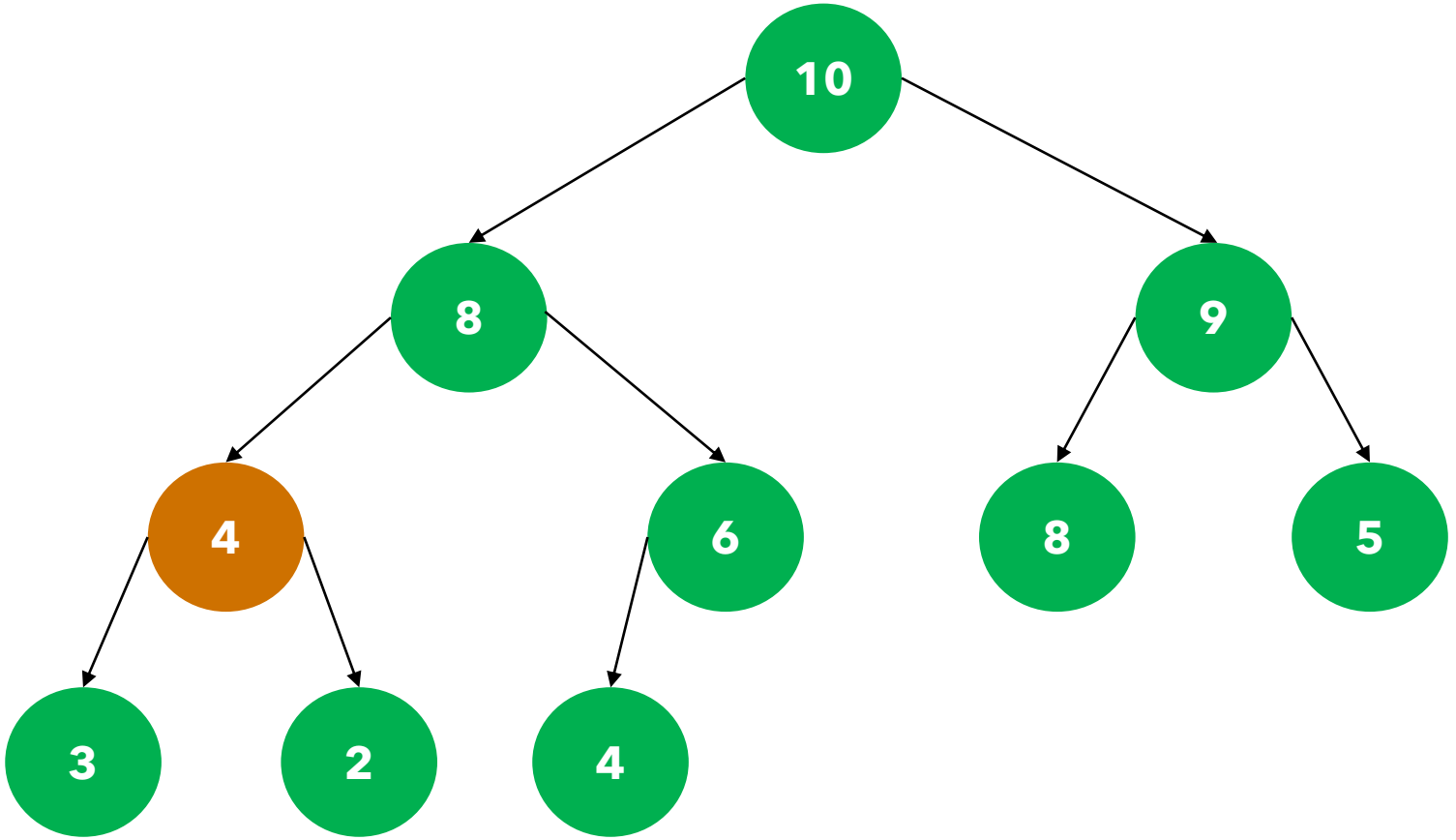
max_heapify è un algoritmo che trasforma un albero del genere in un max heap

`max_heapify` – esempio 1



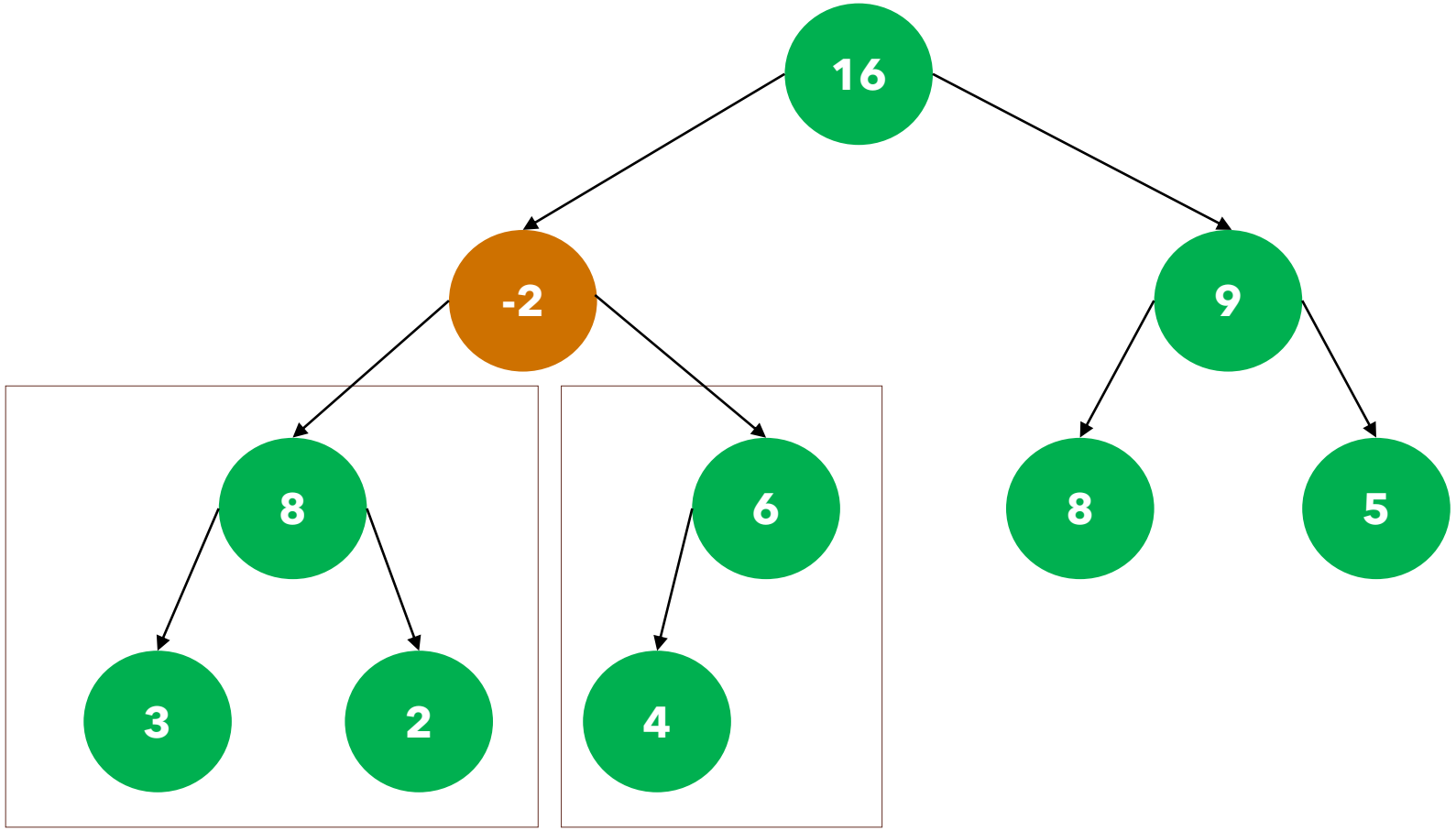
- il nodo su cui è invocato `max_heapify` viene swappato con il figlio maggiore
- l'operazione viene effettuata ricorsivamente sul sottoalbero la cui radice è stata aggiornata, fintantoché la *max-heap property* è violata

`max_heapify` – esempio 1



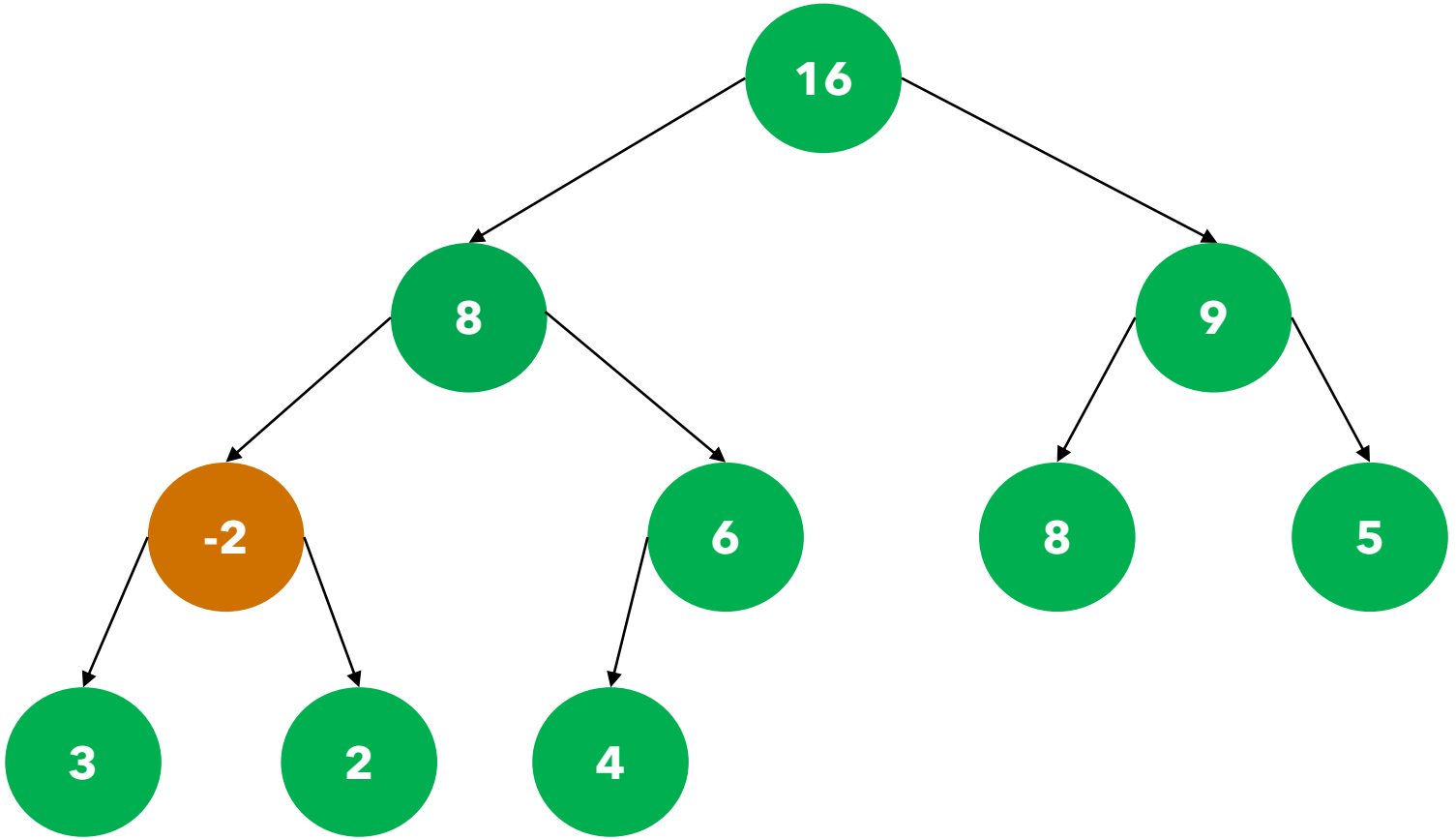
- il nodo su cui è invocato `max_heapify` viene swappato con il figlio maggiore
- l'operazione viene effettuata ricorsivamente sul sottoalbero la cui radice è stata aggiornata, fintantoché la *max-heap property* è violata

`max_heapify` – esempio 2



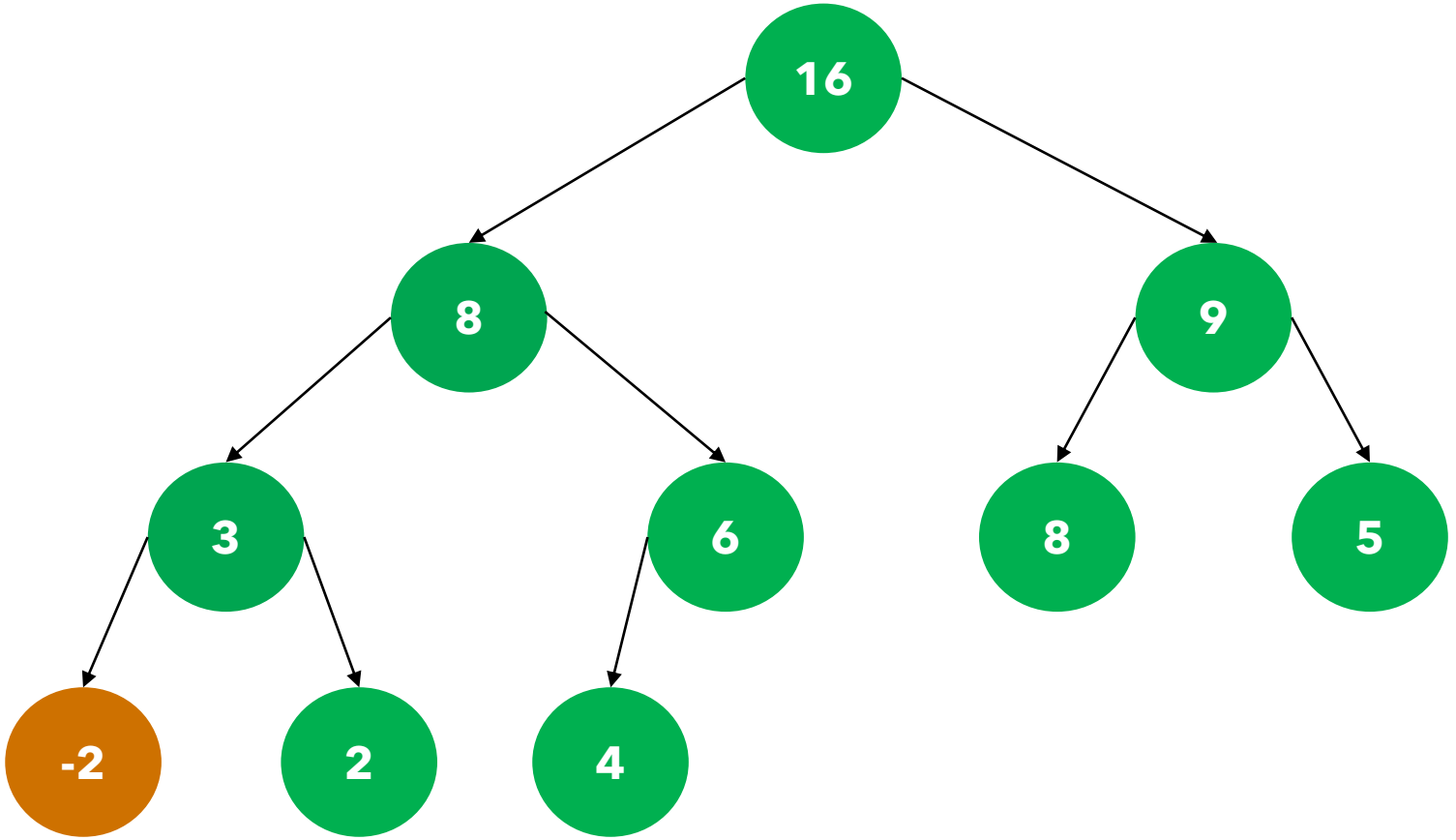
in questo caso, `max_heapify` va invocata sul nodo arancio

`max_heapify` – esempio 2



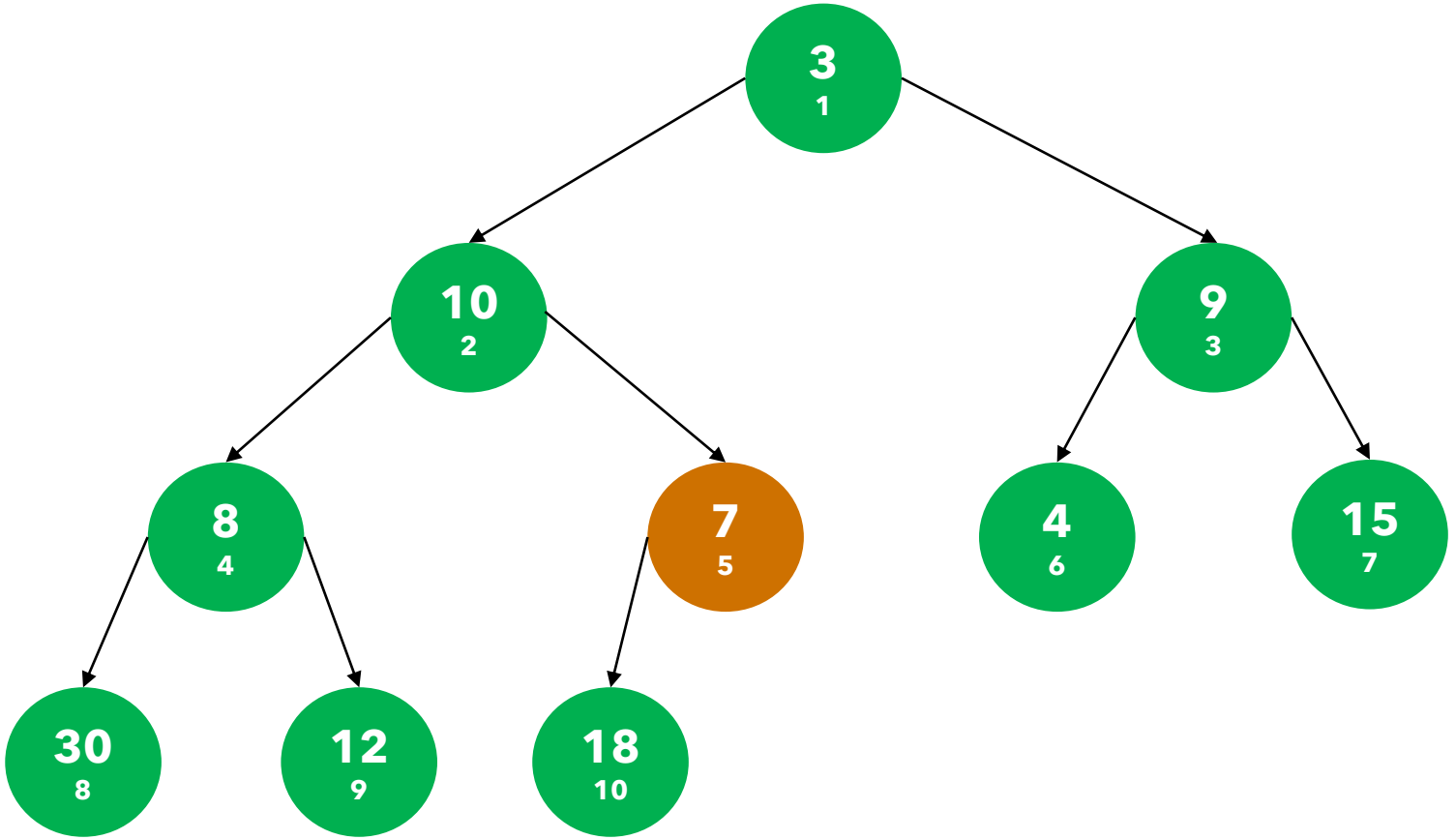
invocazione ricorsiva di `max_heapify` sul nodo arancio

max_heapify – esempio 2



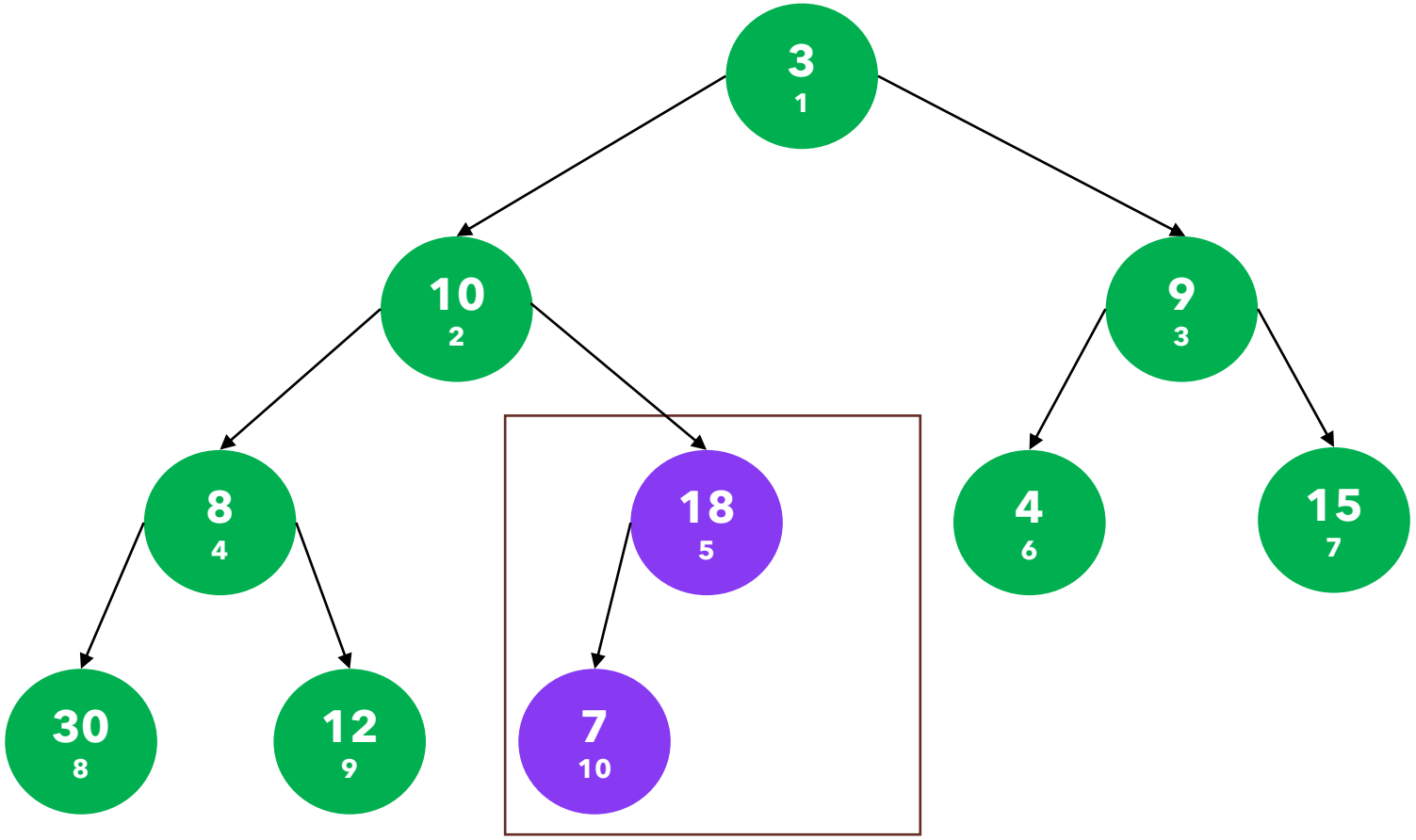
caso base raggiunto, il risultato è un max heap

build_max_heap

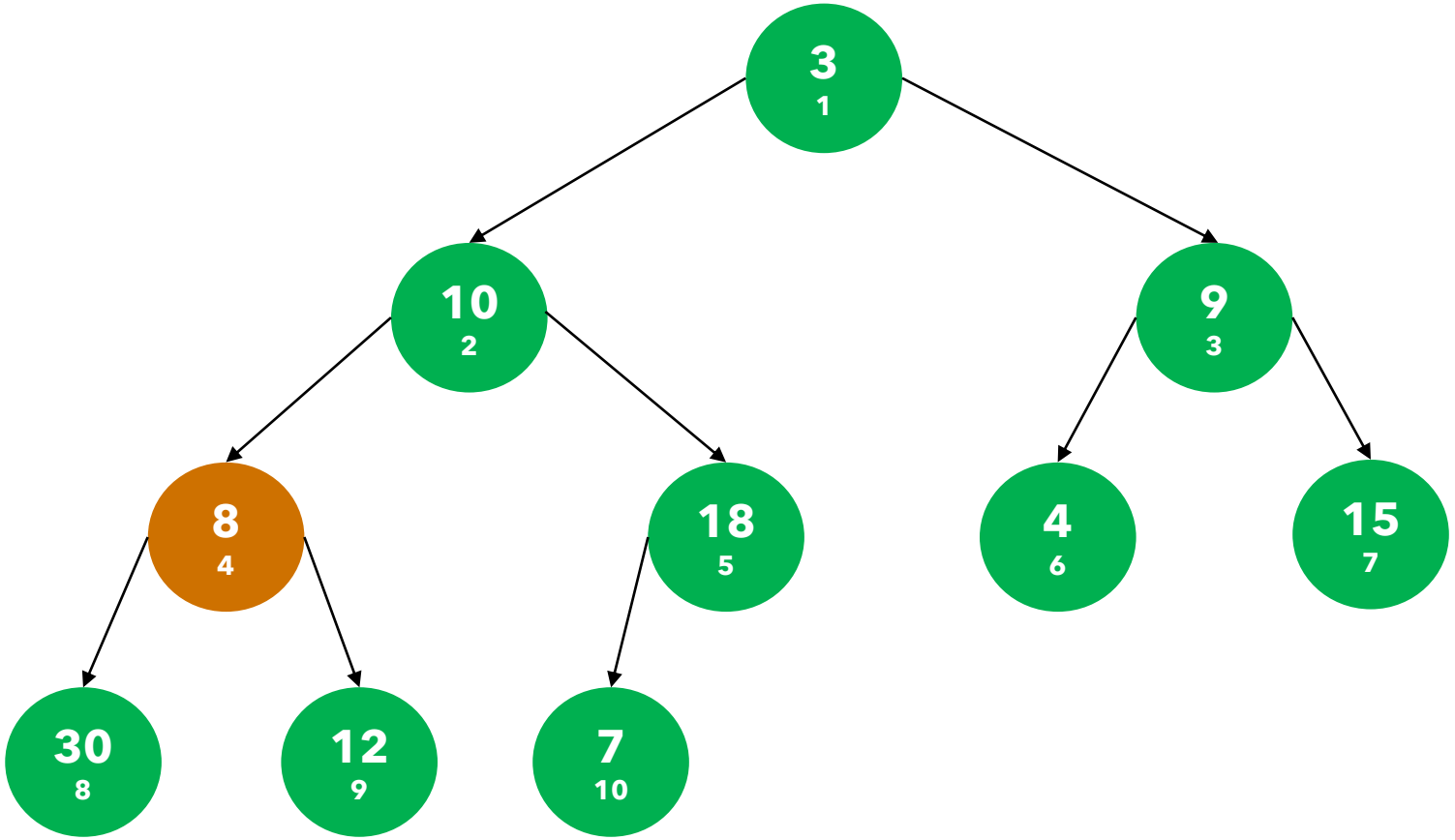


- build_max_heap permette di trasformare un array in un max heap
- si tratta di invocare max_heapify partendo dall'elemento medio dell'array per poi procedere all'indietro

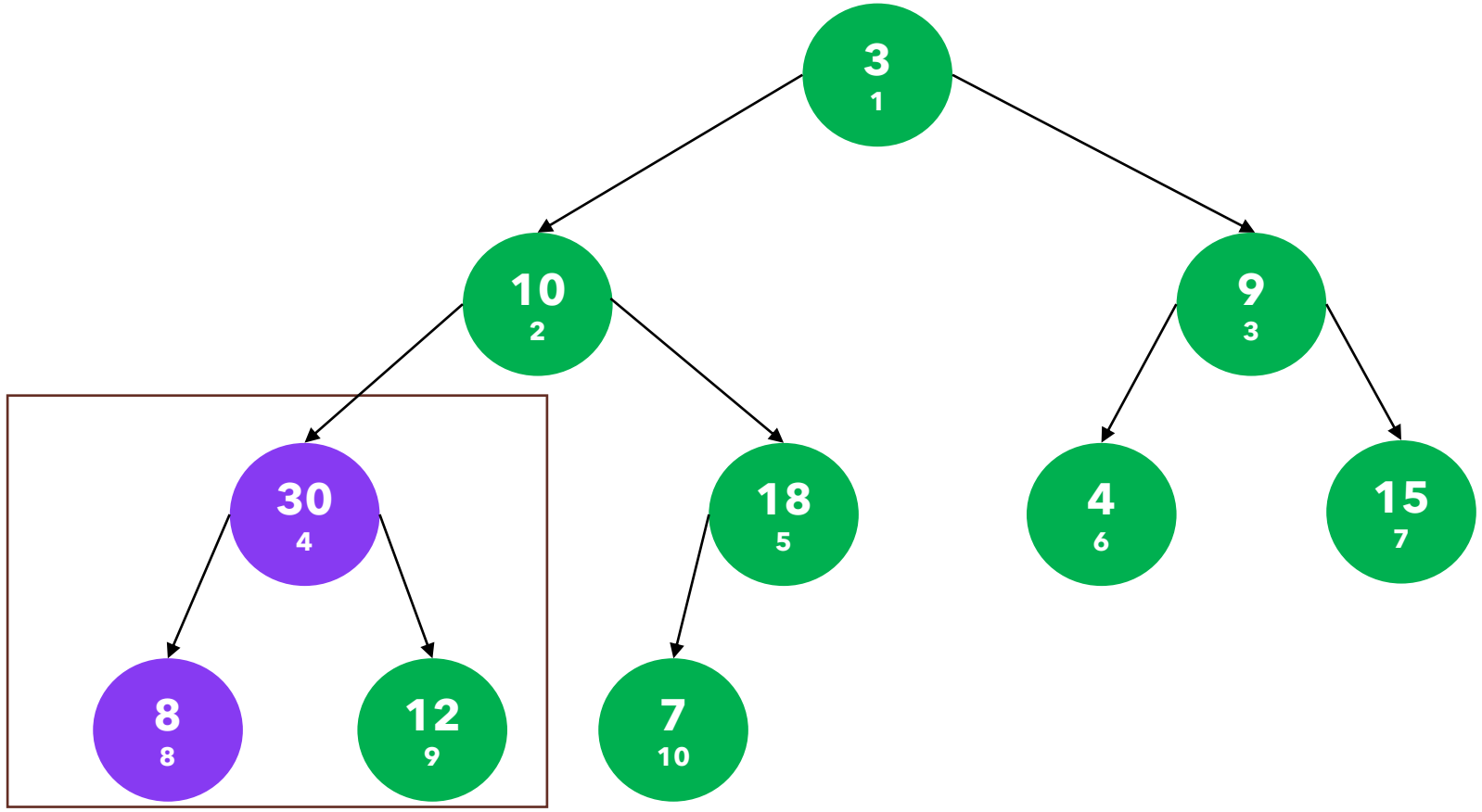
build_max_heap



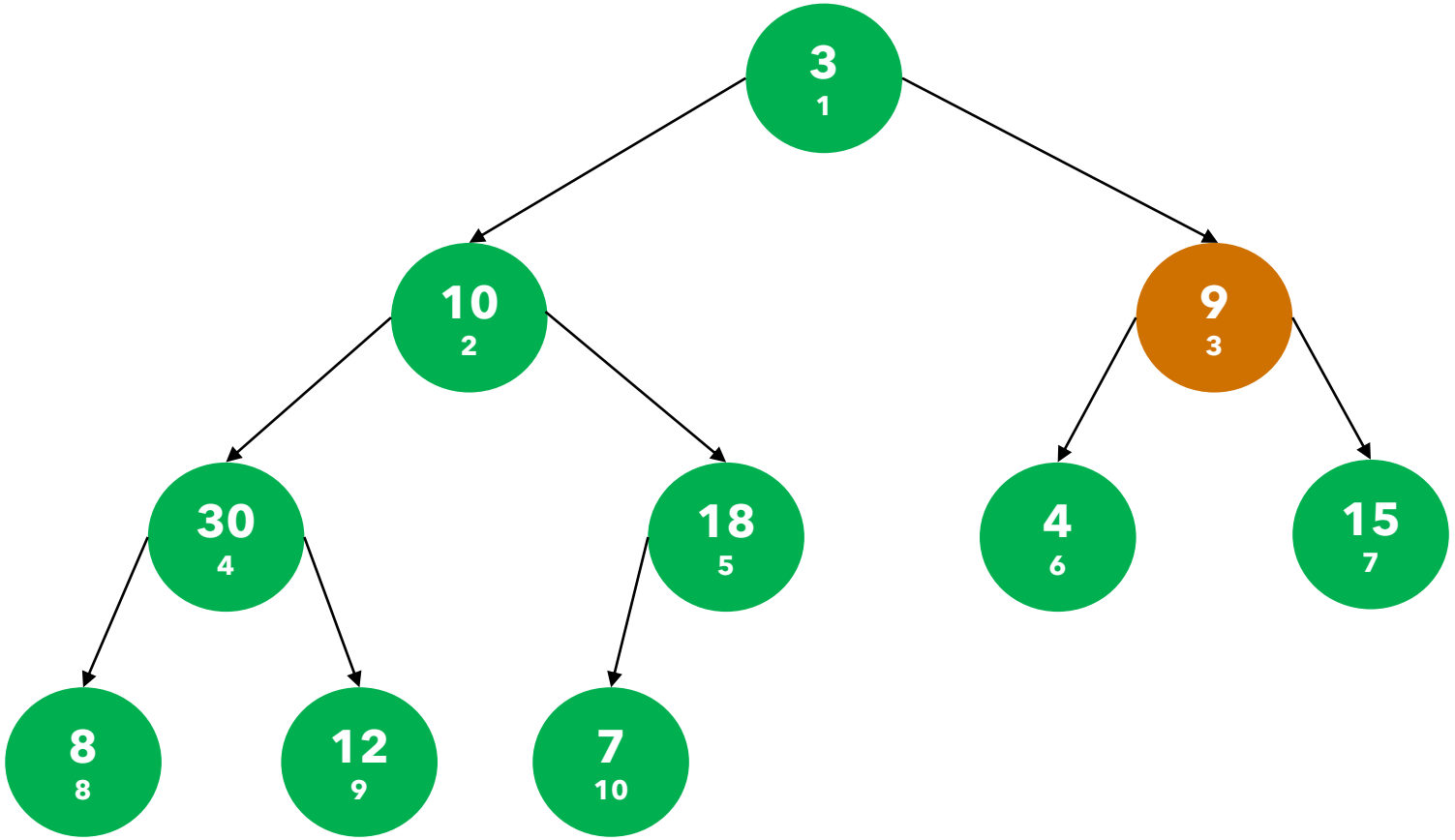
build_max_heap



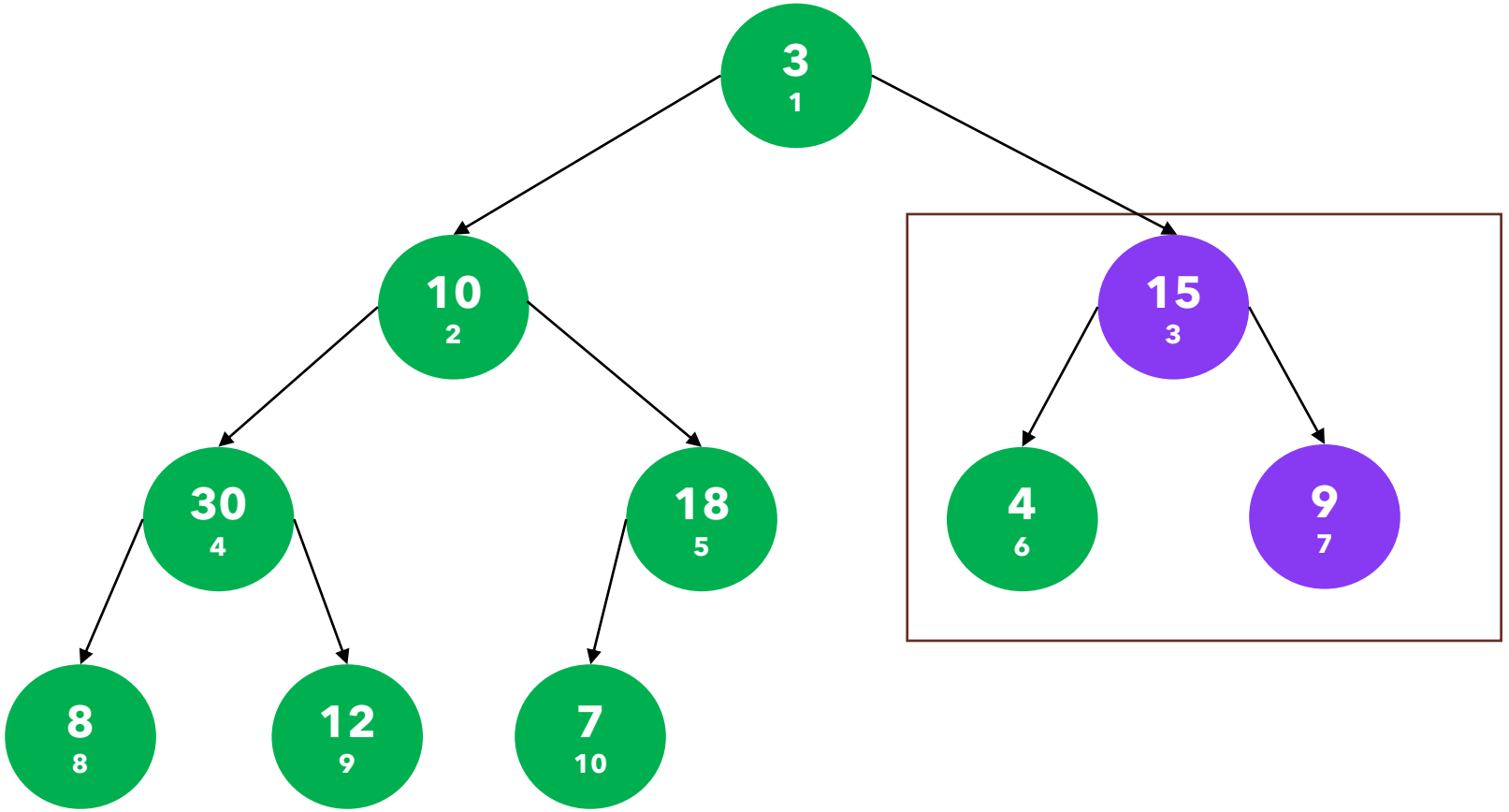
build_max_heap



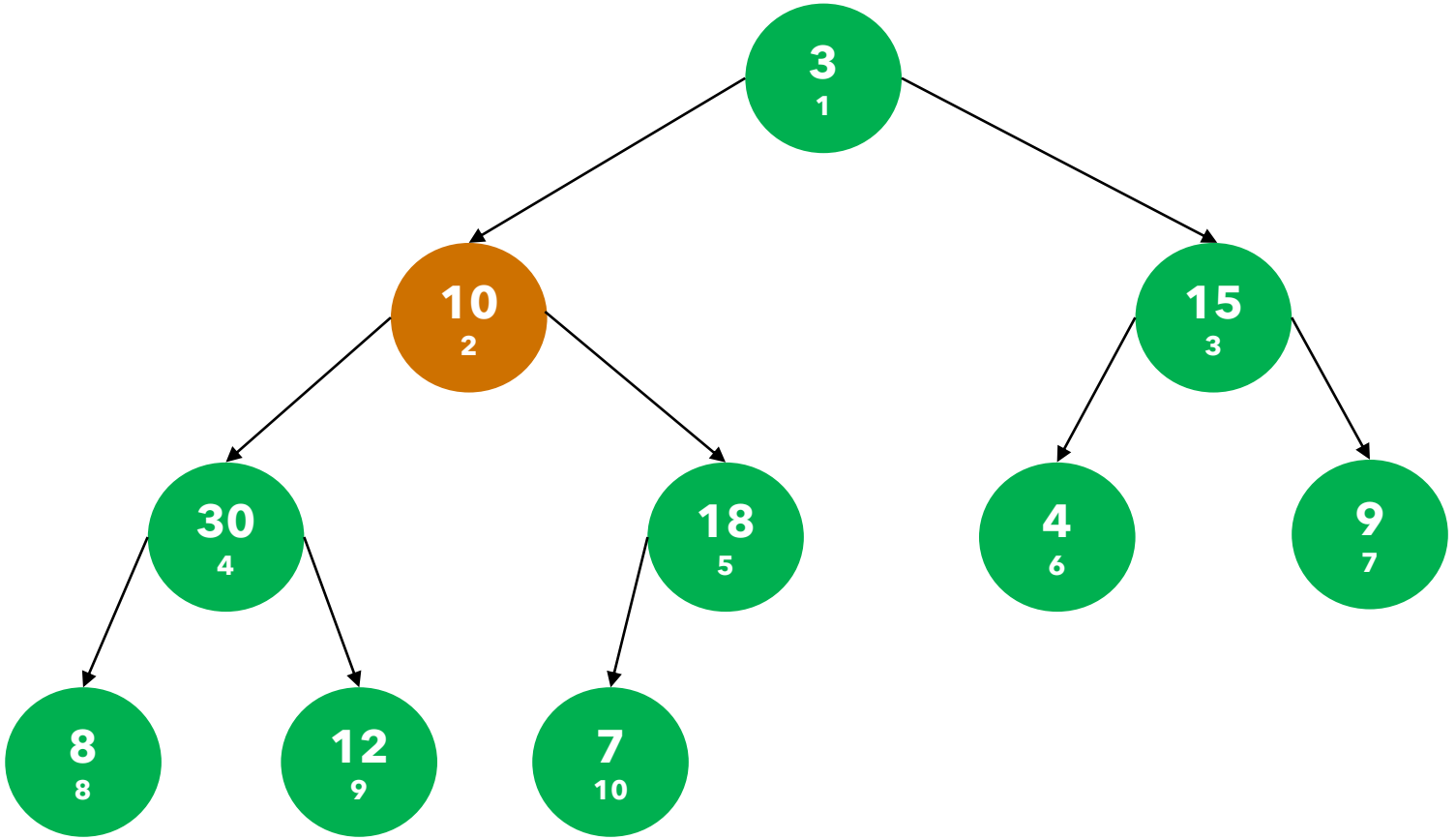
build_max_heap



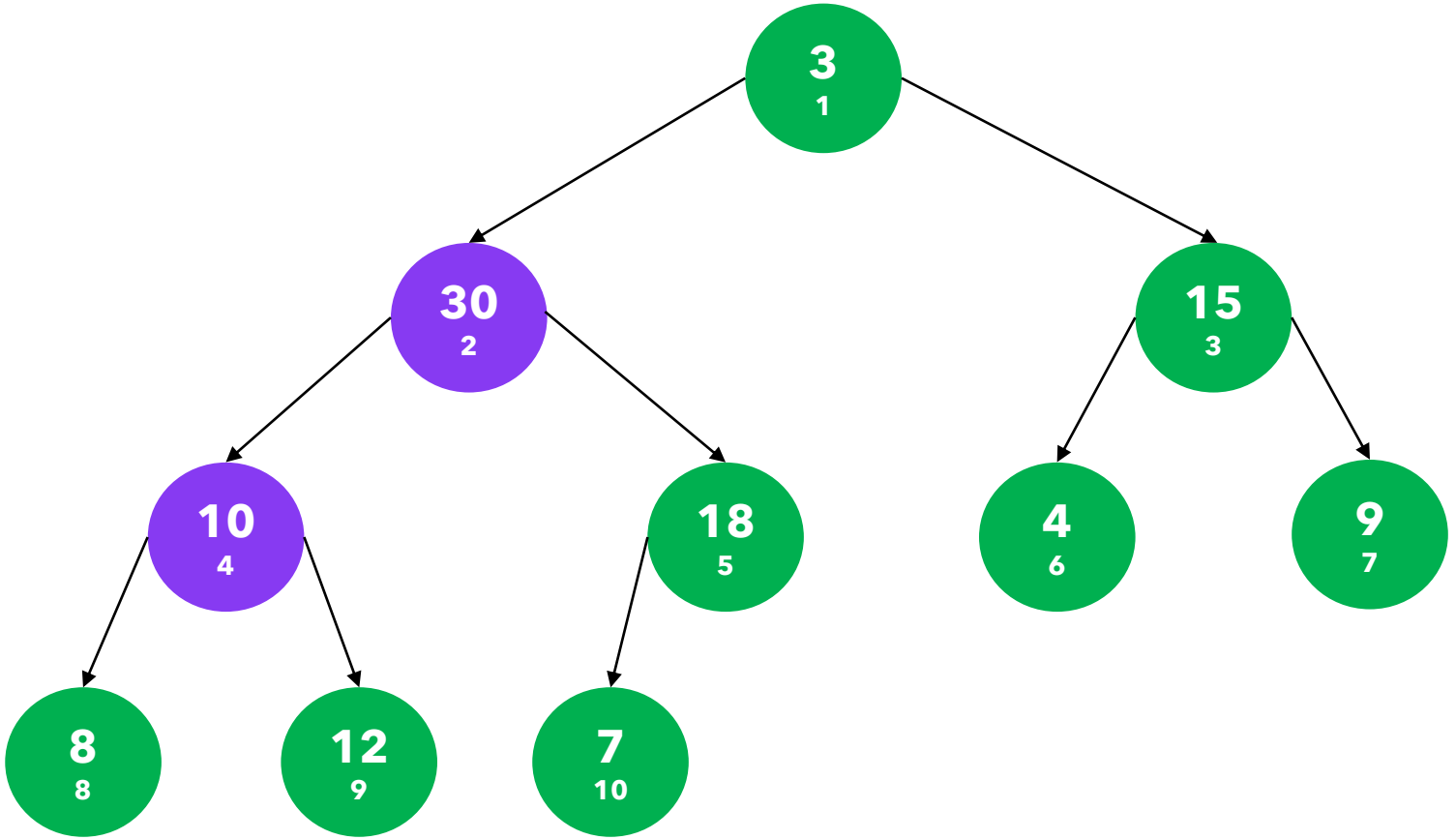
build_max_heap



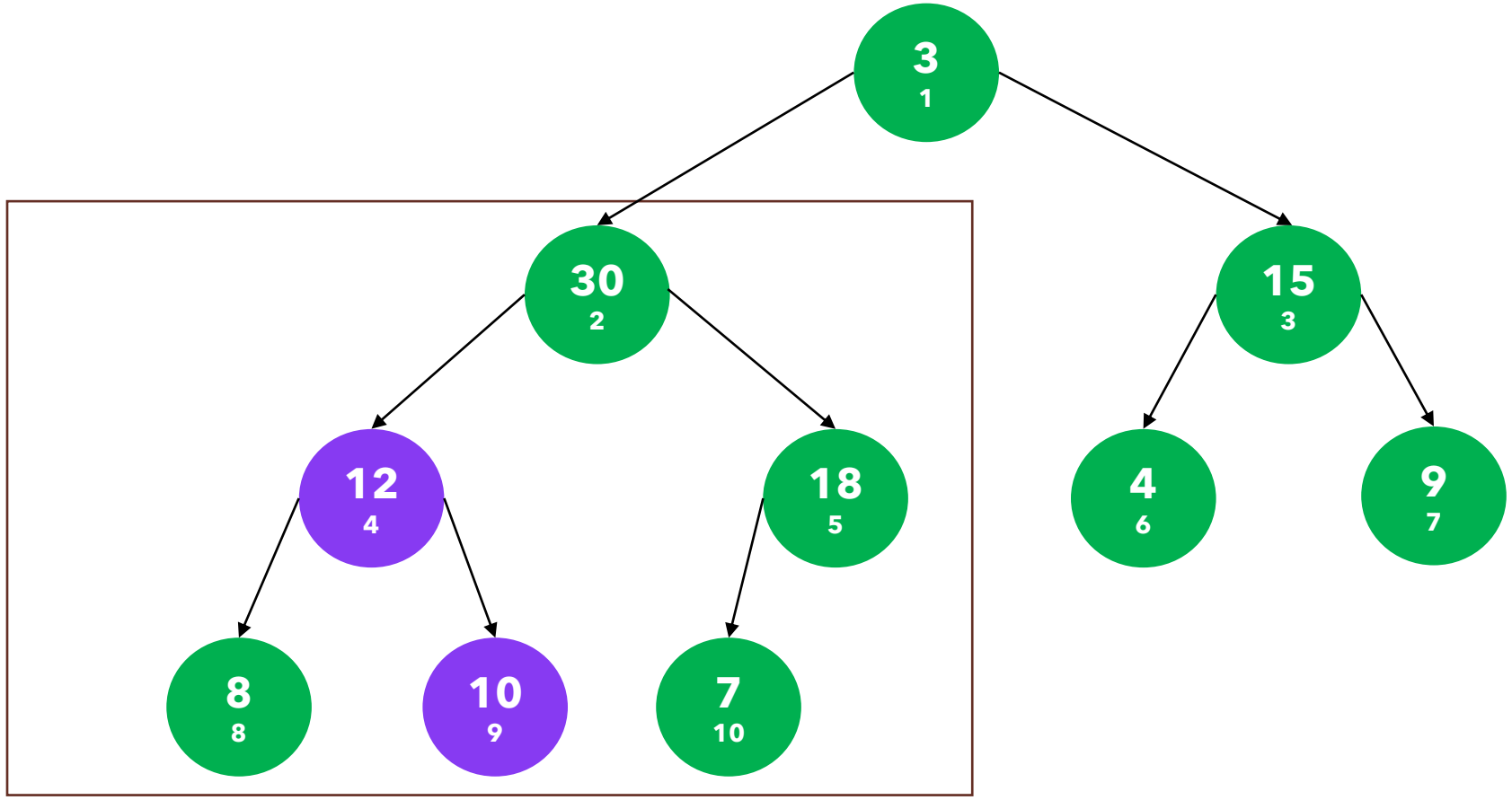
build_max_heap



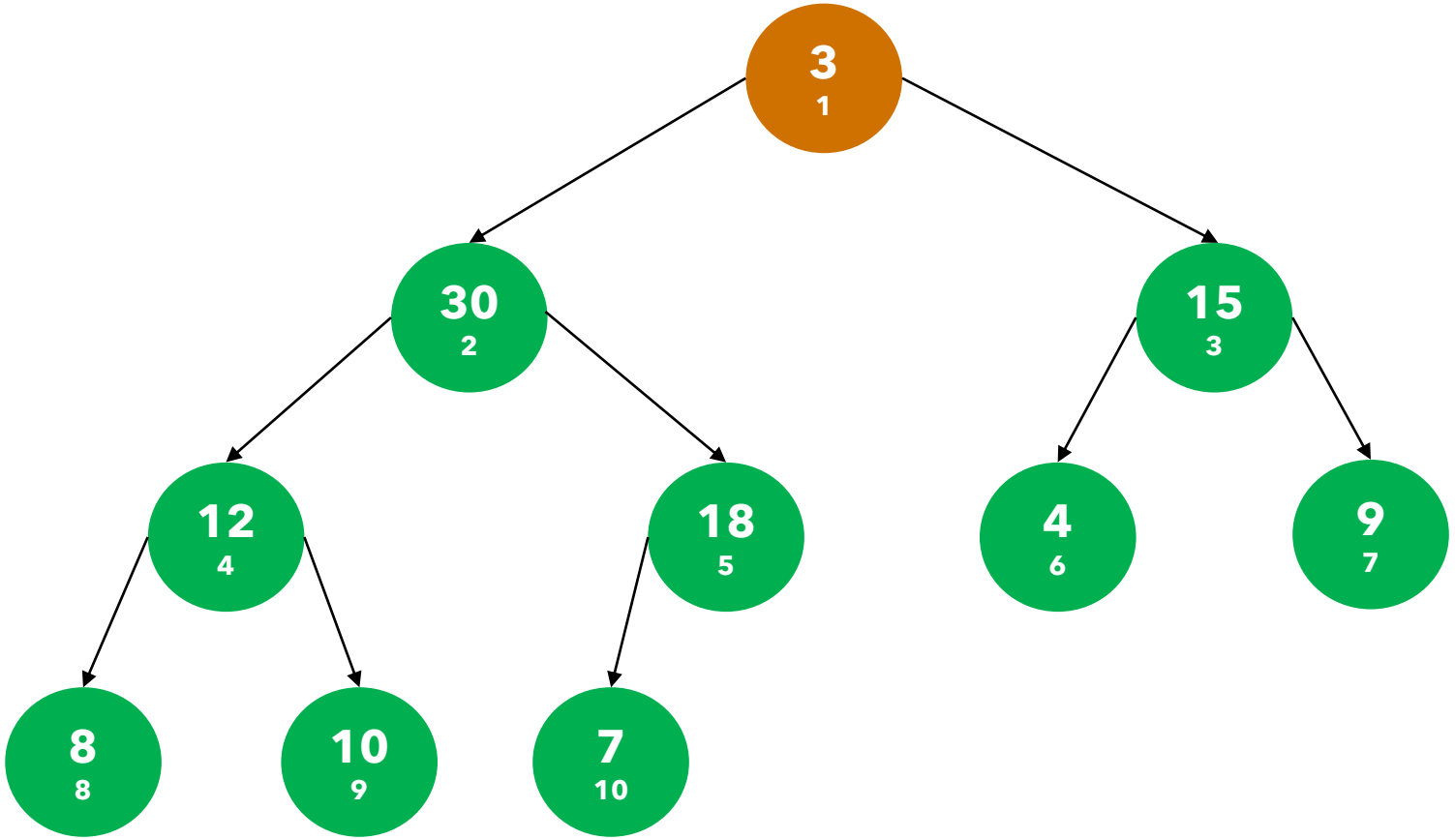
build_max_heap



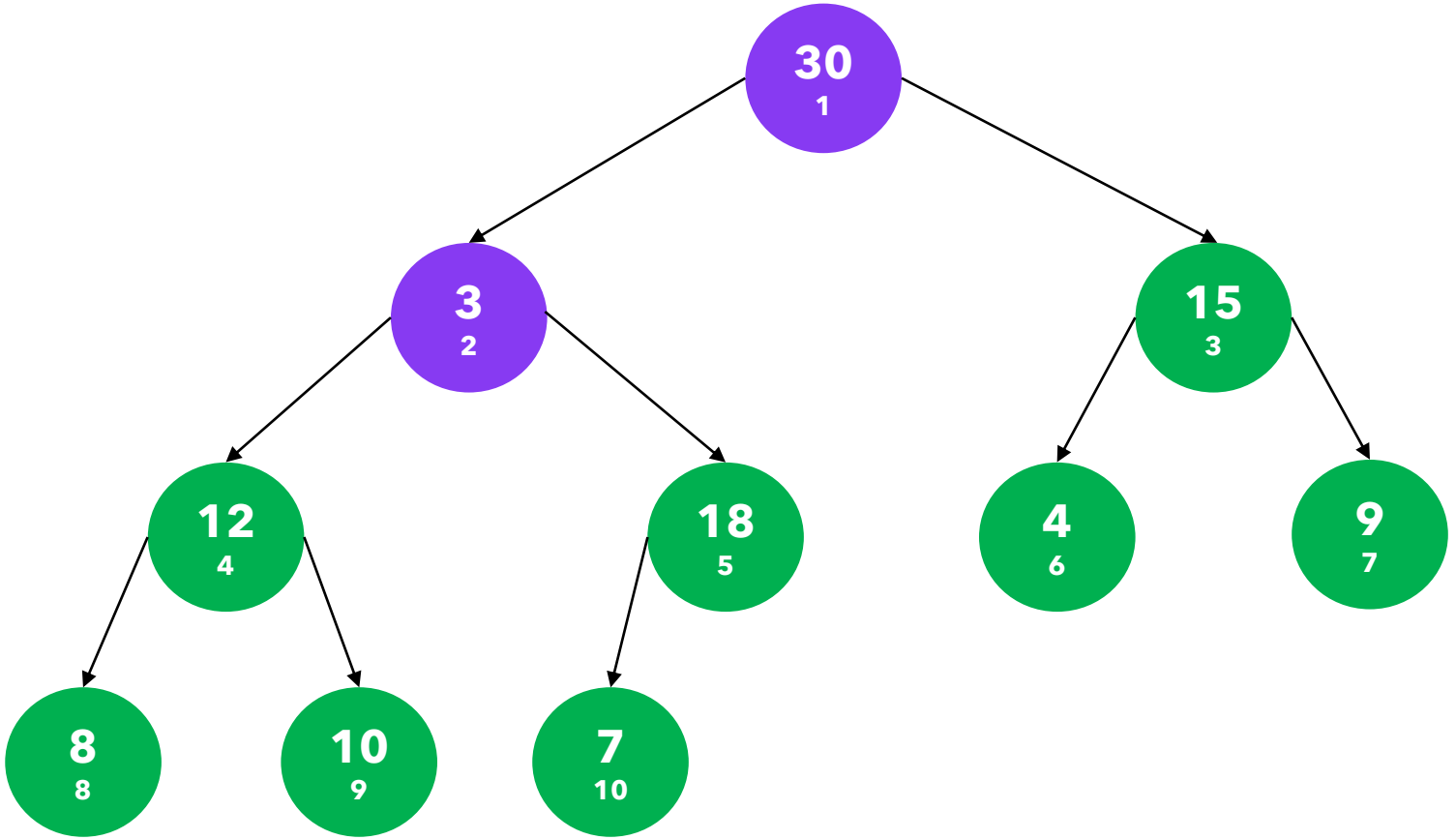
build_max_heap



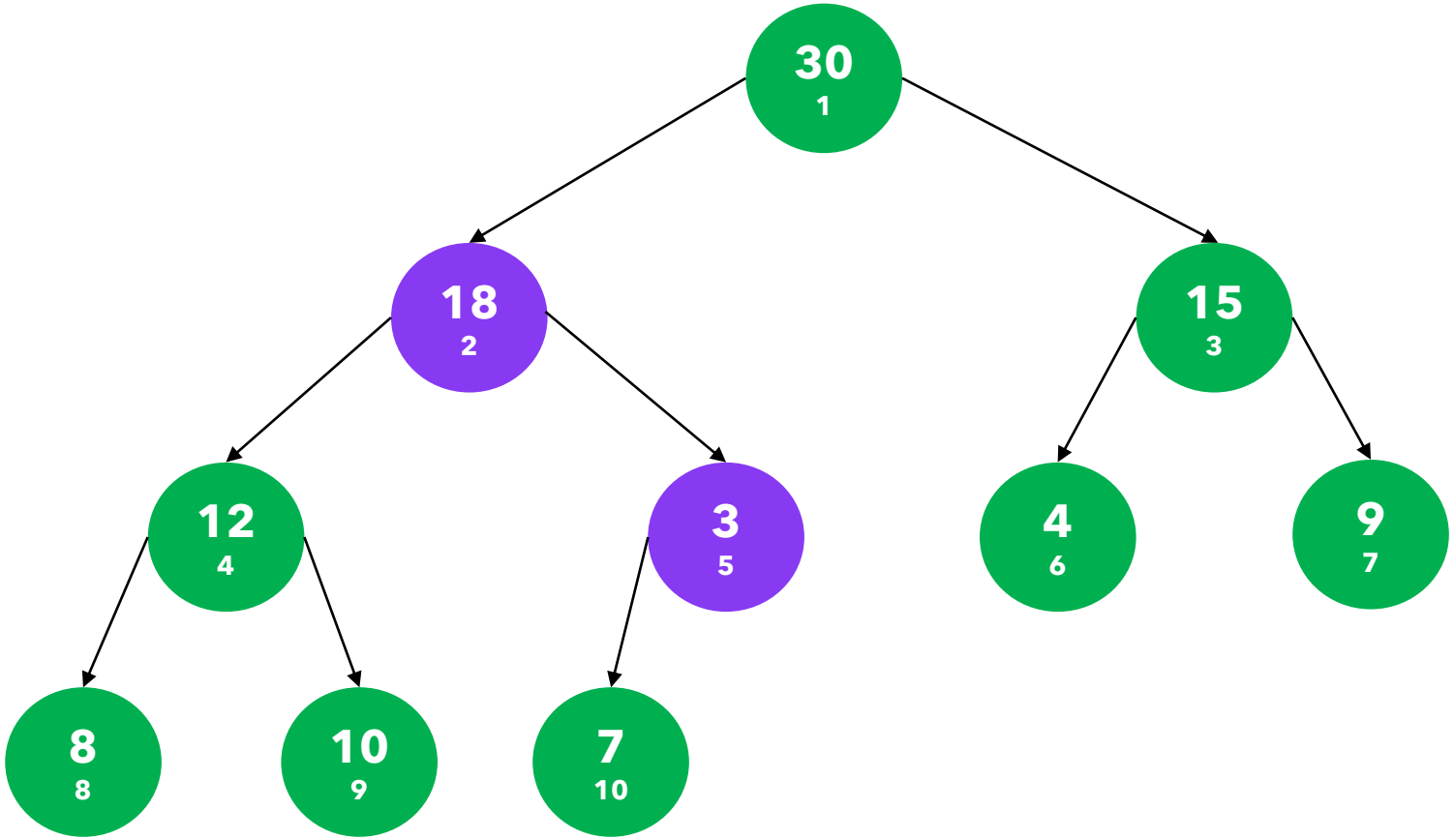
build_max_heap



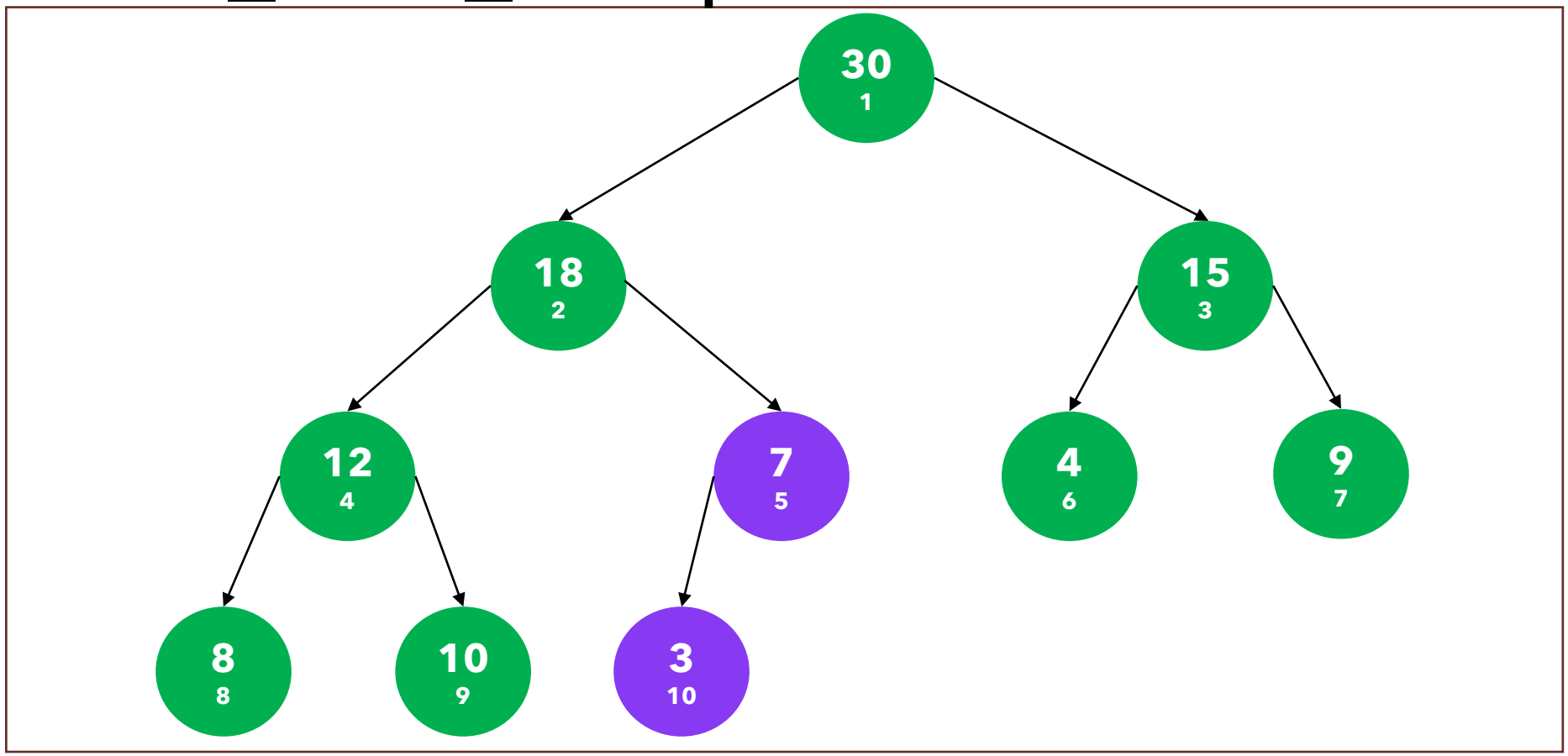
build_max_heap



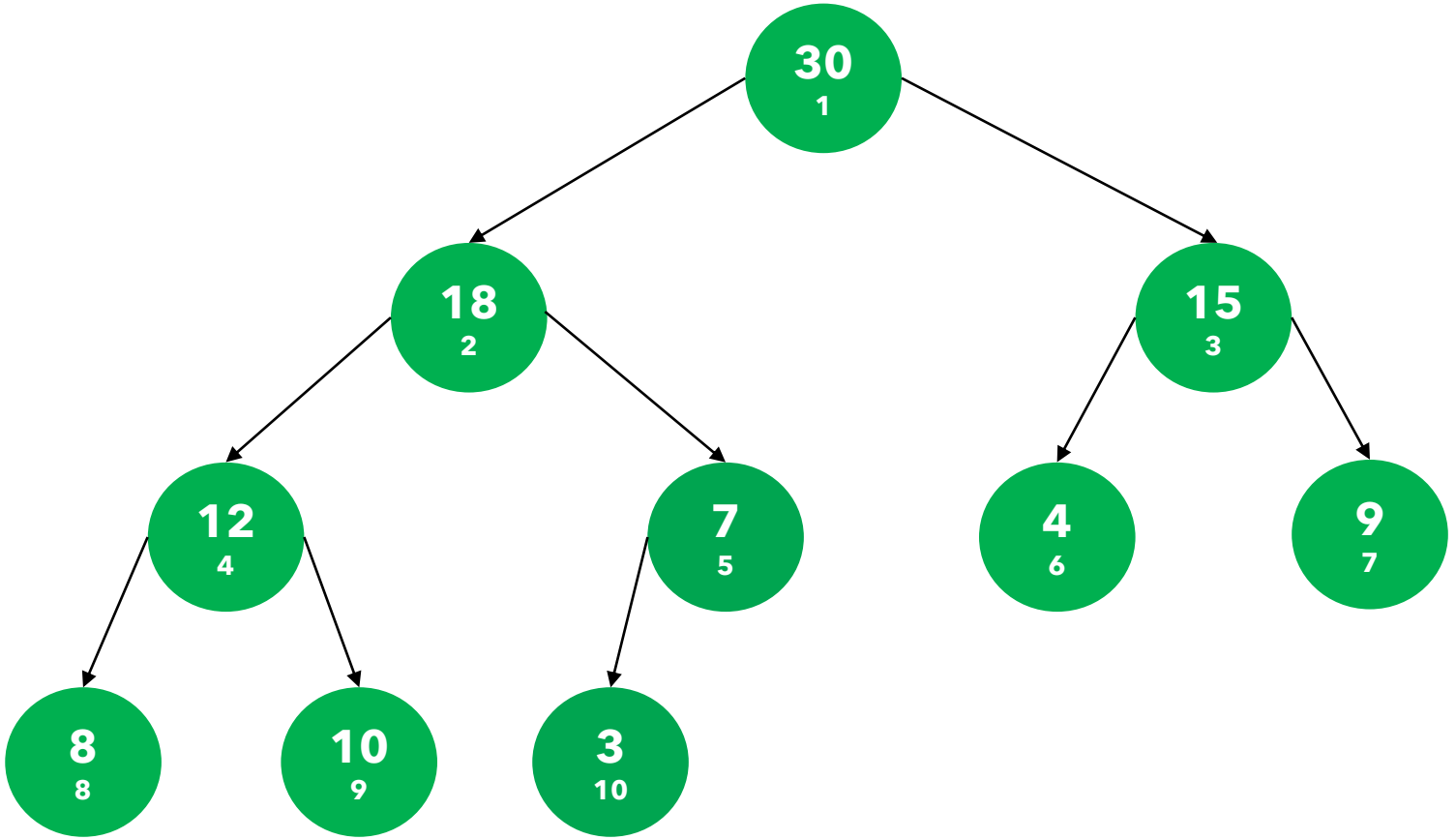
build_max_heap



build_max_heap



build_max_heap



- è possibile ordinare un array in modo efficiente, dopo che è stato trasformato in un max heap?

Heapsort

values	30	18	15	12	7	4	9	8	10	3
indexes	1	2	3	4	5	6	7	8	9	10

- questa è la rappresentazione ad array del max heap costruito nelle slide precedenti
- la costruzione di un max heap tramite le procedure `build_max_heap` e `max_heapify` è un'operazione efficiente (non lo dimostriamo, ma è intuitivo)
- **Heapsort** (inventato da J. W. J. Williams) è un algoritmo di ordinamento che si basa proprio sui max heap

Heapsort

values	*3*	18	15	12	7	4	9	8	10	*30*
indexes	1	2	3	4	5	6	7	8	9	10

- `swap(A[1], A[10])`
- in corsivo gli elementi swappati

Heapsort

values	3	18	15	12	7	4	9	8	10	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in grassetto e in grande gli elementi la cui posizione non cambierà più

Heapsort

values	3	18	15	12	7	4	9	8	10	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in viola la porzione dell'array da trasformare in heap, invocando `max_heapify(A, 1)`

Heapsort

values	18	12	15	10	7	4	9	8	3	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in azzurro la porzione dell'array appena trasformata in max heap

Heapsort

values	*3*	12	15	10	7	4	9	8	*18*	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- `swap(A[1], A[9])`
- in corsivo gli elementi swappati

Heapsort

values	3	12	15	10	7	4	9	8	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in grassetto e in grande gli elementi la cui posizione non cambierà più

Heapsort

values	3	12	15	10	7	4	9	8	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in viola la porzione dell'array da trasformare in heap, invocando `max_heapify(A, 1)`

Heapsort

values	15	12	9	10	7	4	3	8	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in azzurro la porzione dell'array appena trasformata in max heap

Heapsort

values	*8*	12	9	10	7	4	3	*15*	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- `swap(A[1], A[8])`
- in corsivo gli elementi swappati

Heapsort

values	8	12	9	10	7	4	3	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in grassetto e in grande gli elementi la cui posizione non cambierà più

Heapsort

values	8	12	9	10	7	4	3	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in viola la porzione dell'array da trasformare in heap, invocando `max_heapify(A, 1)`

Heapsort

values	12	10	9	8	7	4	3	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in azzurro la porzione dell'array appena trasformata in max heap

Heapsort

values	*3*	10	9	8	7	4	*12*	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- `swap(A[1], A[7])`
- in corsivo gli elementi swappati

Heapsort

values	3	10	9	8	7	4	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in grassetto e in grande gli elementi la cui posizione non cambierà più

Heapsort

values	3	10	9	8	7	4	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in viola la porzione dell'array da trasformare in heap, invocando `max_heapify(A, 1)`

Heapsort

values	10	8	9	3	7	4	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in azzurro la porzione dell'array appena trasformata in max heap

Heapsort

values	*4*	8	9	3	7	*10*	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- `swap(A[1], A[6])`
- in corsivo gli elementi swappati

Heapsort

values	4	8	9	3	7	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in grassetto e in grande gli elementi la cui posizione non cambierà più

Heapsort

values	4	8	9	3	7	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in viola la porzione dell'array da trasformare in heap, invocando `max_heapify(A, 1)`

Heapsort

values	9	8	4	3	7	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in azzurro la porzione dell'array appena trasformata in max heap

Heapsort

values	*7*	8	4	3	*9*	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- `swap(A[1], A[5])`
- in corsivo gli elementi swappati

Heapsort

values	7	8	4	3	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in grassetto e in grande gli elementi la cui posizione non cambierà più

Heapsort

values	7	8	4	3	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in viola la porzione dell'array da trasformare in heap, invocando `max_heapify(A, 1)`

Heapsort

values	8	7	4	3	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in azzurro la porzione dell'array appena trasformata in max heap

Heapsort

values	*3*	7	4	*8*	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- `swap(A[1], A[4])`
- in corsivo gli elementi swappati

Heapsort

values	3	7	4	**8**	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in grassetto e in grande gli elementi la cui posizione non cambierà più

Heapsort

values	3	7	4	**8**	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in viola la porzione dell'array da trasformare in heap, invocando `max_heapify(A, 1)`

Heapsort

values	7	3	4	**8**	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in azzurro la porzione dell'array appena trasformata in max heap

Heapsort

values	*4*	3	*7*	**8**	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- `swap(A[1], A[3])`
- in corsivo gli elementi swappati

Heapsort

values	4	3	**7**	**8**	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in grassetto e in grande gli elementi la cui posizione non cambierà più

Heapsort

values	4	3	**7**	**8**	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in viola la porzione dell'array da trasformare in heap, invocando `max_heapify(A, 1)`

Heapsort

values	4	3	**7**	**8**	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in azzurro la porzione dell'array appena trasformata in max heap

Heapsort

values	*3*	*4*	**7**	**8**	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- `swap(A[1], A[2])`
- in corsivo gli elementi swappati

Heapsort

values	3	**4**	**7**	**8**	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in grassetto e in grande gli elementi la cui posizione non cambierà più

Heapsort

values	**3**	**4**	**7**	**8**	**9**	**10**	**12**	**15**	**18**	**30**
indexes	1	2	3	4	5	6	7	8	9	10

- in grassetto e in grande gli elementi la cui posizione non cambierà più
- l'array è ordinato

Implementazione

- https://github.com/Cyofanni/high-school-cs-class/blob/main/python/assignments/heapsort.py