1. Si consideri un array $A = [1, 2, 3, \ldots, n]$ di n interi distinti ordinati in modo crescente. L'obiettivo è costruire da A un *heap binario massimo*, applicando l'algoritmo classico `BuildMaxHeap`, che esegue la procedura `MaxHeapify` sui nodi interni, partendo dal basso verso l'alto.
   (a) Calcolare il numero di scambi (swap) applicati durante l'esecuzione dell'algoritmo per $n = 10$.
   (b) Fornire una **stima asintotica** del numero di scambi effettuati da `BuildMaxHeap` in funzione della dimensione n dell'array. Motivare la risposta.

1. Costruzione del max-heap con `BuildMaxHeap` per `n = 10`

L'array iniziale è

indice	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
valore	1	2	3	4	5	6	7	8	9	10

`BuildMaxHeap` visita i nodi interni nell'ordine 5, 4, 3, 2, 1 (numerazione 1-based).
Ad ogni chiamata di `MaxHeapify` riportiamo solo gli scambi (`swap`) effettuati.

nodo	**prima situazione**	**scambio**	**dopo lo scambio**	**swap tot.**
5	`5 (10)`	$5 \leftrightarrow 10$	`1 2 3 4 10 6 7 8 9 5`	**1**
4	`4 (9)`	$4 \leftrightarrow 9$	`1 2 3 9 10 6 7 8 4 5`	**2**
3	`3 (7)`	$3 \leftrightarrow 7$	`1 2 7 9 10 6 3 8 4 5`	**3**
2	`2 (9,10)`	$2 \leftrightarrow 10$ $2 \leftrightarrow 5$	`1 10 7 9 2 6 3 8 4 5` `1 10 7 9 5 6 3 8 4 2`	**5**

1	`1 (10,7)`	$1 \leftrightarrow 10$
		$1 \leftrightarrow 9$
		$1 \leftrightarrow 8$
			`10 1 7 9 2 6 3 8 4 5`
			`10 9 7 1 5 6 3 8 4 2`
			`10 9 7 8 5 6 3 1 4 2`
				8

Alla fine dell'algoritmo sono avvenuti **8 scambi**.

2. Stima asintotica del numero di scambi per un array ordinato crescente di lunghezza `n`

Altezza dei nodi e swap di `MaxHeapify`

- Sia $H = \lfloor \log_2 n \rfloor$ l'altezza dell'heap completo che conterrà i n elementi.
- Un nodo di profondità d (root = 0) ha altezza $H-d$.
- Nel caso peggiore (array crescente) quando invoco `MaxHeapify` quel nodo **scende sempre fino a una foglia**, compiendo esattamente $H-d$ swap.

Somma degli swap su tutti i nodi interni

- (1) Ci sono 2^d nodi alla profondità d; quindi il numero totale di scambi
$$S(n) = \sum_{d=0}^{H-1} 2^d (H-d)$$
- (2) Riscriviamo la somma ponendo $k=H-d$ [nb: k = h-d -> d = H-k]:
$$S(n) = \sum_{d=0}^{H} 2^{H-k} k = \sum_{d=0}^{H} 2^H + 2^{-k} k = 2^H \sum_{k=1}^{H} \frac{k}{2^k}$$

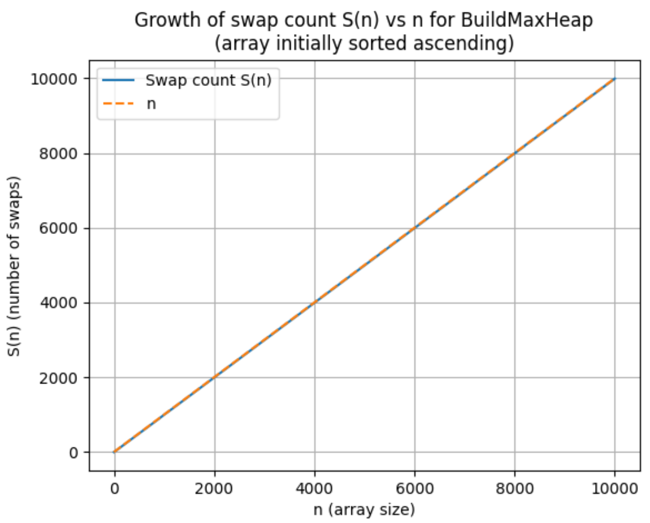
Limite superiore
Poiché la serie $\sum_{k=1}^{\infty} k/2^k$ vale 2, dalla (2):

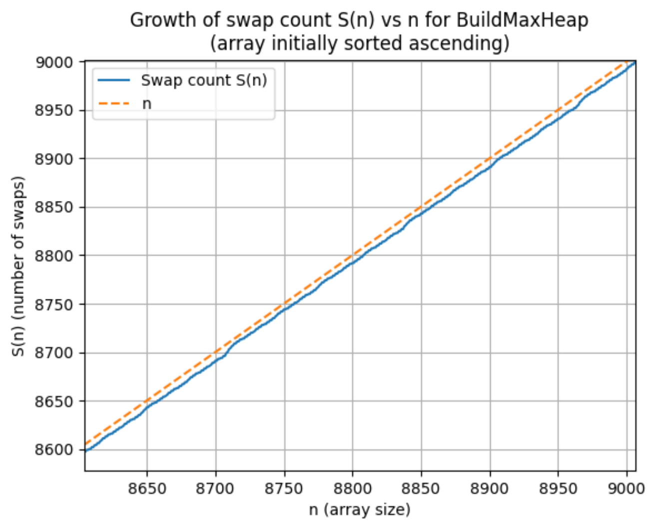
$$S(n) \leq 2^H * 2 = 2^{H+1}$$

Con $2^H \leq n < 2^{H+1}$ *otteniamo* $S(n) = O(n)$

Extra: codice Python (per chi non fosse convinto può usare il seguente codice python. Si vede che ogni volta che si aumenta n, il numero di swap massimo sarà O(n) cioè entrambi i numeri cresceranno in maniera asintoticamente lineare, o comunque O(n) rappresenta un limite asintotico per S(n))

Dimostrando anche la presenza di un limite inferiore Ω(n), dal momento che abbiamo già dimostrato un limite O(n) si può quindi dire che la complessità è Θ(n), come è chiaro vedendo un plot di come crescono n e S(n) (pagina 3)





```python
def build_max_heap(arr):
    n = len(arr)
    swap_count = 0

    def max_heapify(i, heap_size):
        nonlocal swap_count
        largest = i
        left = 2 * i + 1
        right = 2 * i + 2

        if left < heap_size and arr[left] > arr[largest]:
            largest = left
        if right < heap_size and arr[right] > arr[largest]:
            largest = right

        if largest != i:
            arr[i], arr[largest] = arr[largest], arr[i]
            swap_count += 1
            max_heapify(largest, heap_size)

    for i in range(n // 2 - 1, -1, -1):
        max_heapify(i, n)

    return swap_count, arr

n = 10000
array = list(range(1, n + 1))  # [1, 2, 3, ..., n]
swaps, heap = build_max_heap(array.copy())
print(f"Numero di swap: {swaps}")
```