Heap Sort

Group: Wenqian, Sylvia, Louise, Shirley

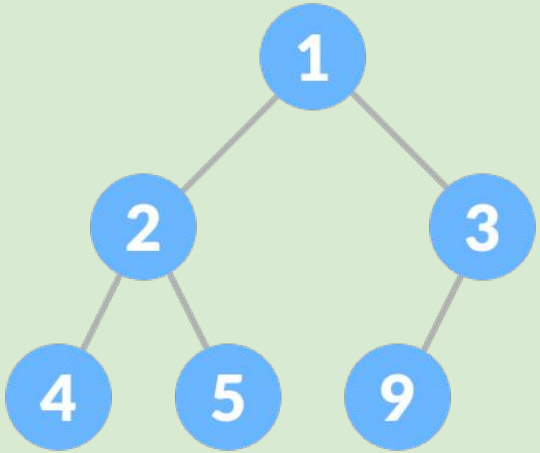
Agenda

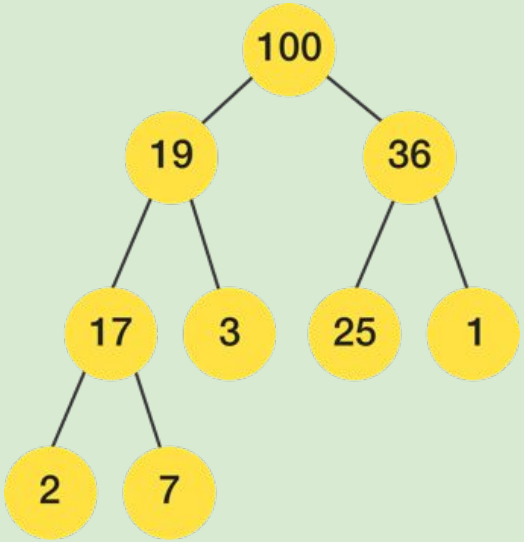
1. Understanding Heap Sort
    a. Heap
    b. Heapify
    c. Heap Sort
2. Code Analysis
    a. Emergency Room Priority Situation
3. Time Complexity

Understanding Heap Sort

What is Heap?

Heap is a data structure to manage information. Heap is a complete binary tree, which means **ALL** except the last level of nodes are completely filled, and the lowest level is filled to a certain point from the left.

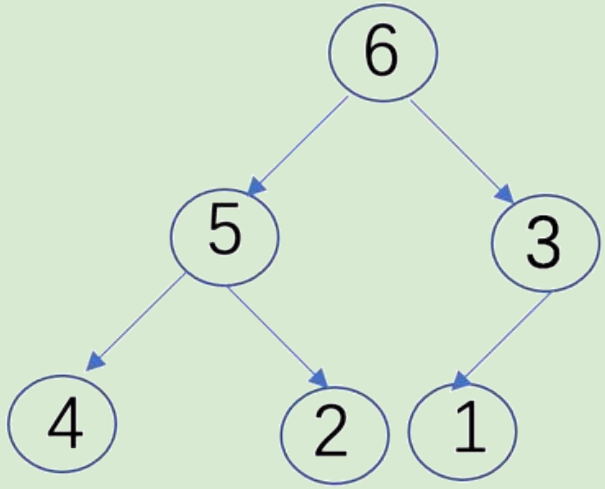




Types of Heap - Max Heap

The first type of heap is called max heap, in which all parent nodes are greater than or equal to the values of their children, subsequently the root node contains the greatest value in the heap.

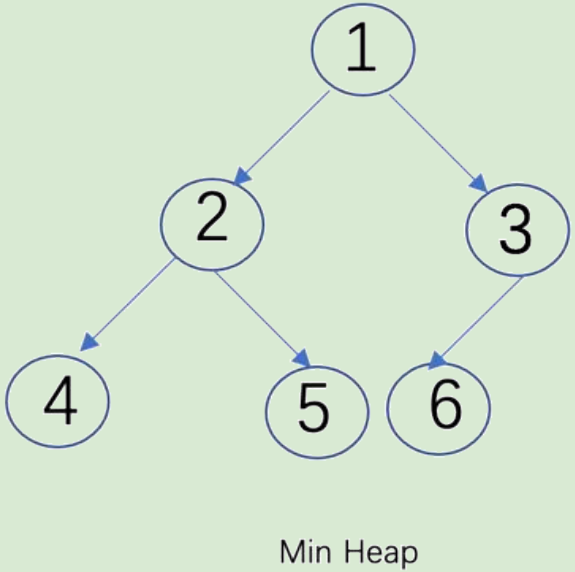
This is what we use for heap sort!



Max Heap

Types of Heap - Min heap

The other type of heap is called min heap, in which all parent nodes are less than or equal to the values of their children, subsequently the root node contains the least value in the heap.



Min Heap

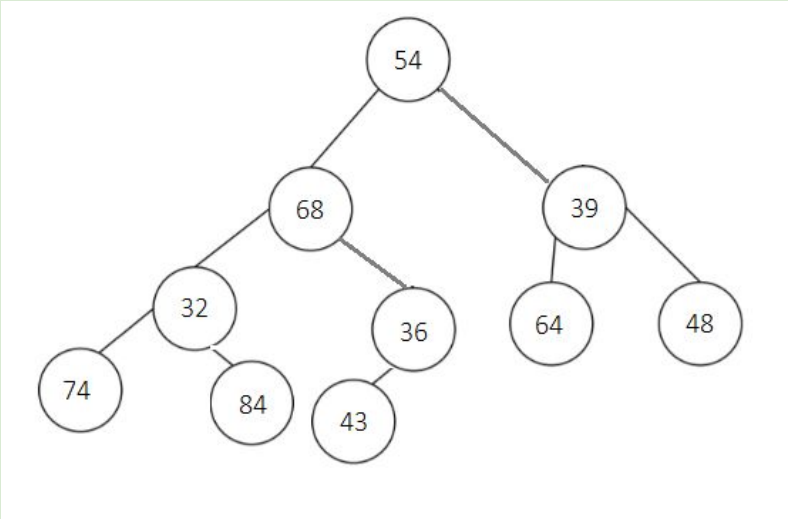
What is heapify?

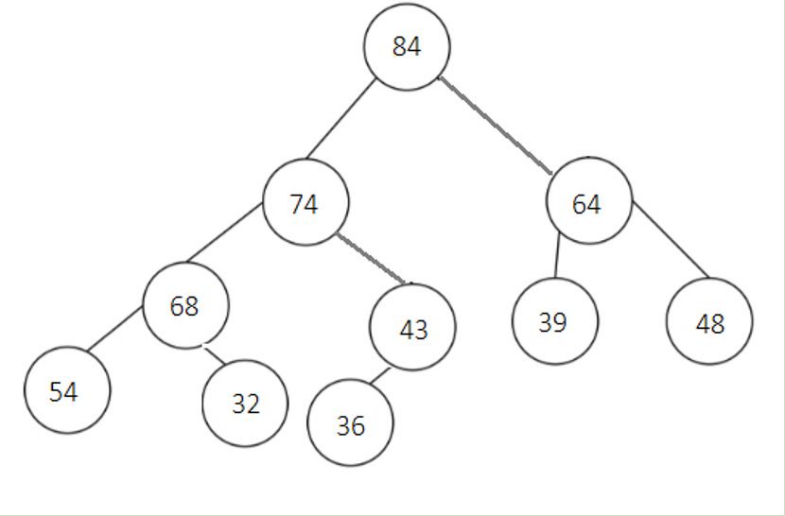
Heapify is the process of creating a heap from a given array or a binary tree. In other words, heapify is the process to rearrange the elements to maintain the property of heap data structure.

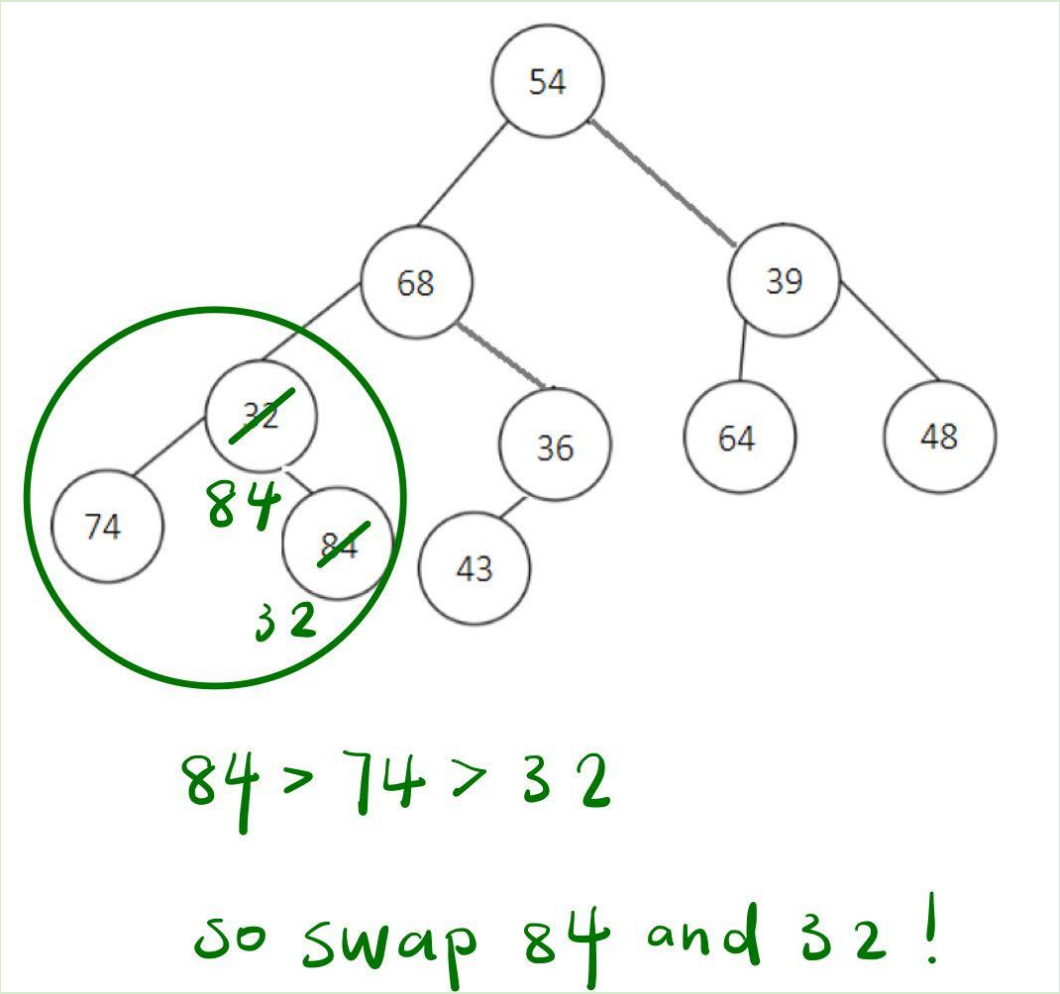
- **Max heapify**
- Min heapify

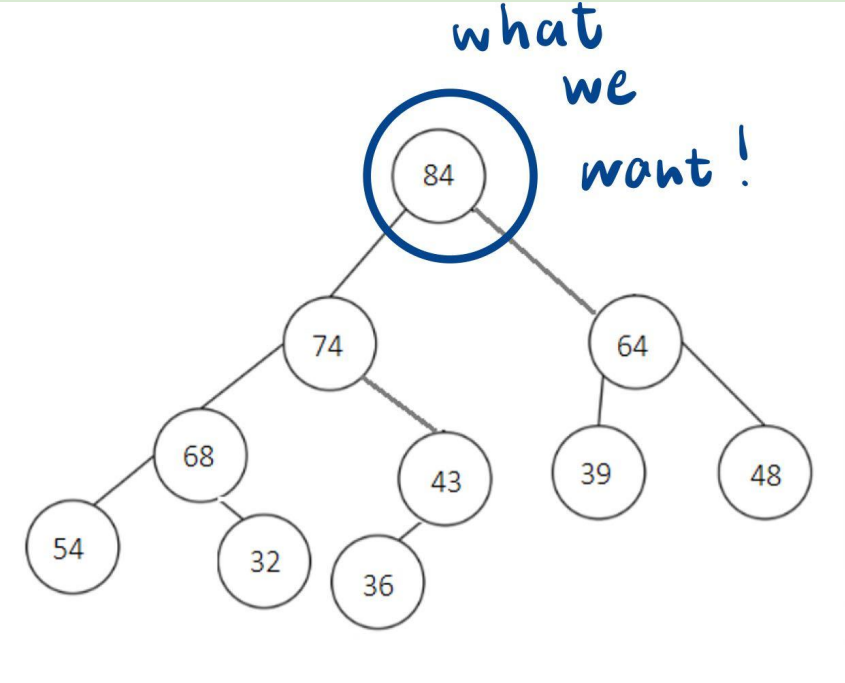
Max Heapify Example

Given array: [54,68,39,32,36,64,48,74,84,43]






54
68
39
32
36
64
48
74
84
84
43
32
84 > 74 > 32
so swap 84 and 32 !



We continue to examine all the subtrees in the heap until the heap as a whole satisfies the max heap's property. After it reaches a max heap, the value that root node contains is what we need for a heap sort.

What is heap sort?

After we explained what max-heapify is, it is much easier to understand what heapsort is. To summarize what a heap sort is, it consists of two steps:

1. Build a max heap from a given array.
2. Run max heapify until the tree is a max heap.

The basic idea of a heap sort is first to build a heap and get rid of the root, which is the max value in the heap, by swapping it with the last element of the array. The process is repeated until each element is placed in the right position.

Heap Sort Example

Given array: [54,68,39,32,36,64,48,74,84,43]

After first max-heapify: [**84**, 74, 64, 68, 43, 39, 48, 54, 32, 36]

Swap first element with the last element: [**36**, 74, 64, 68, 43, 39, 48, 54, 32, **84**]

Perform max heapify again to find largest value: [**36, 74, 64, 68, 43, 39, 48, 54, 32, 84**]

…… Repeat the steps ……

Until we sort every element and reach: **[32, 36, 39, 43, 48, 54, 64, 68, 74, 84]**

Code Analysis

Use emergency room scenario to demonstrate how heap sort can be used to efficiently sort a priority queue

Emergency Room Priority Situation

Priority score = 0.4 * (100 - age) + 0.6 * (severity of symptoms)

40% Age Component:
(100 - age) This means that younger patients will generally have a higher priority score than older patients.

60% Symptoms Component:
Severity of symptoms
Score 5 Resuscitation
Score 4 Emergent
Score 3 Urgent
Score 2 Less Urgent
Score 1 Non Urgent

(This a simplified and potentially biased model! It is not a substitute for professional medical evaluation or diagnosis.)

Information of the Ten Patients and Priority Scores

```
{'name': 'John', 'age': 25, 'priority': 5.4, 'symptoms': 'Fever, cough, body aches'},
{'name': 'Mary', 'age': 35, 'priority': 6.8, 'symptoms': 'Difficulty breathing, chest pain'},
{'name': 'Bob', 'age': 45, 'priority': 3.9, 'symptoms': 'Nausea, vomiting, diarrhea'},
{'name': 'Alice', 'age': 50, 'priority': 3.2, 'symptoms': 'Headache, dizziness, fatigue'},
{'name': 'Chris', 'age': 60, 'priority': 3.6, 'symptoms': 'Broken arm, bleeding'},
{'name': 'Emily', 'age': 30, 'priority': 6.4, 'symptoms': 'Severe abdominal pain, fever'},
{'name': 'David', 'age': 55, 'priority': 4.8, 'symptoms': 'Chest tightness, shortness of breath'},
{'name': 'Samantha', 'age': 65, 'priority': 7.4, 'symptoms': 'Unconscious, Unresponsive'},
{'name': 'Oliver', 'age': 40, 'priority': 8.4, 'symptoms': 'Seizures, confusion, disorientation'},
{'name': 'Julia', 'age': 45, 'priority': 4.3, 'symptoms': 'High blood pressure, headache, blurred vision'}]
```

Step 1: Build a max heap from a given array

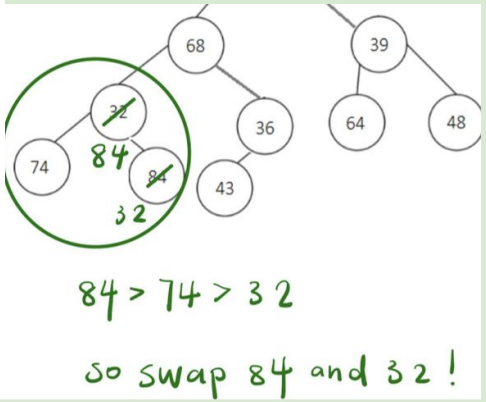
```
def heapify(arr, n, i):
    largest = i # Initialize largest as root
    left = 2 * i + 1 # left = 2*i + 1
    right = 2 * i + 2 # right = 2*i + 2

    # If left child is larger than root
    if left < n and arr[left]['priority'] > arr[largest]['priority']:
        largest = left

    # If right child is larger than largest so far
    if right < n and arr[right]['priority'] > arr[largest]['priority']:
        largest = right

    # If largest is not root
    if largest != i:
        # Swap arr[i] and arr[largest]
        arr[i], arr[largest] = arr[largest], arr[i]

        # Recursively heapify the affected sub-tree
        heapify(arr, n, largest)
```



Step 2:

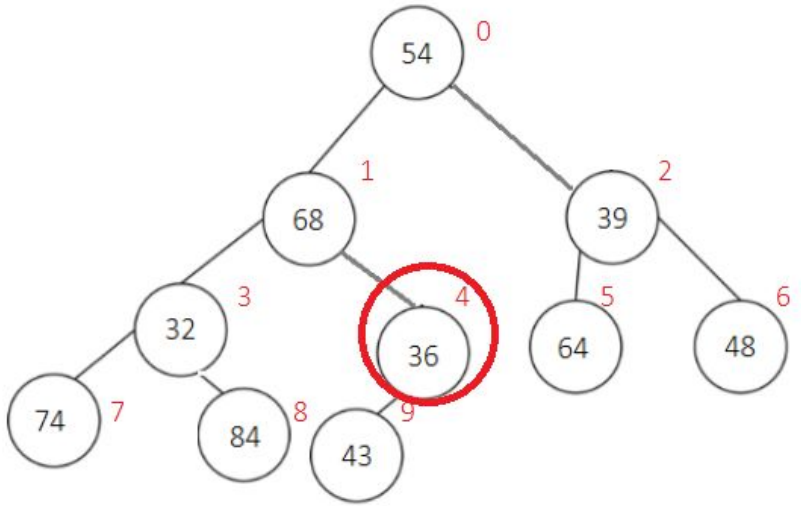
```
# Define the heap sort function
def heap_sort(arr):
    n = len(arr)

    # Build heap (rearrange array)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    # One by one extract an element from heap
    for i in range(n - 1, 0, -1):
        # Move current root to end
        arr[0], arr[i] = arr[i], arr[0]

        # call max heapify on the reduced heap
        heapify(arr, i, 0)

    return arr
```



Step 2:

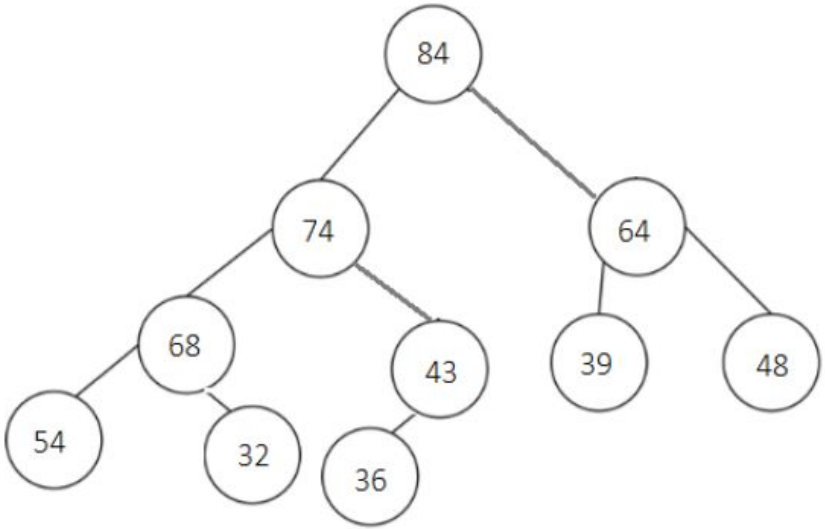
```
# Define the heap sort function
def heap_sort(arr):
    n = len(arr)

    # Build heap (rearrange array)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    # One by one extract an element from heap
    for i in range(n - 1, 0, -1):
        # Move current root to end
        arr[0], arr[i] = arr[i], arr[0]

        # call max heapify on the reduced heap
        heapify(arr, i, 0)

    return arr
```



Step 2:

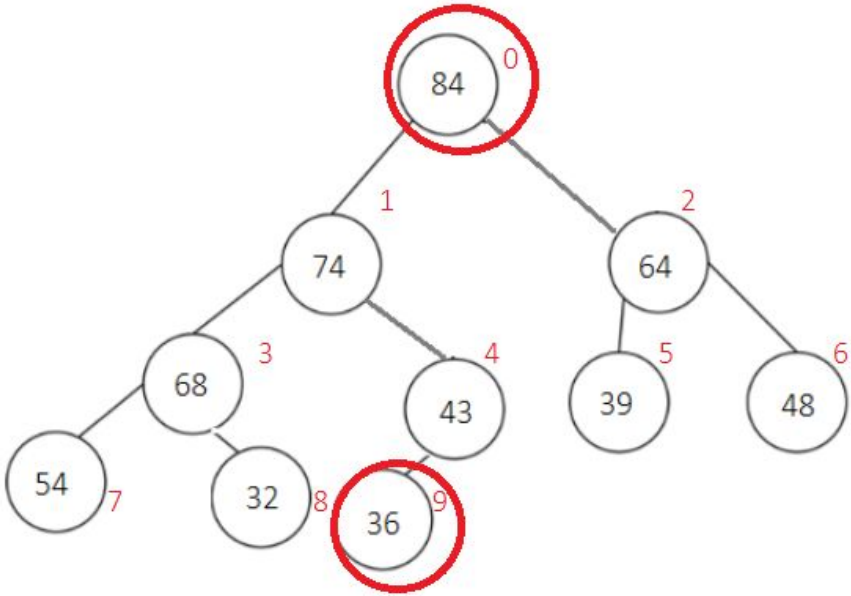
```
# Define the heap sort function
def heap_sort(arr):
    n = len(arr)

    # Build heap (rearrange array)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    # One by one extract an element from heap
    for i in range(n - 1, 0, -1):
        # Move current root to end
        arr[0], arr[i] = arr[i], arr[0]

        # call max heapify on the reduced heap
        heapify(arr, i, 0)

    return arr
```



Step 2:

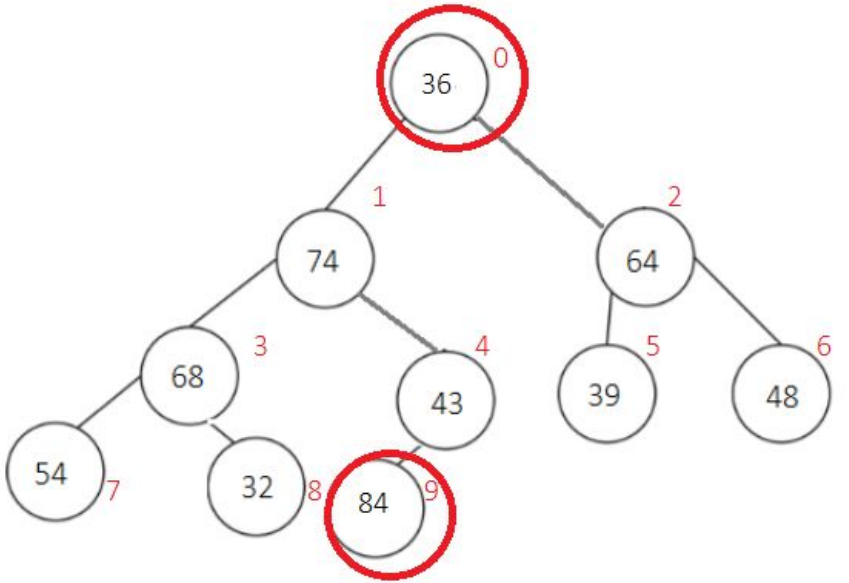
```
# Define the heap sort function
def heap_sort(arr):
    n = len(arr)

    # Build heap (rearrange array)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    # One by one extract an element from heap
    for i in range(n - 1, 0, -1):
        # Move current root to end
        arr[0], arr[i] = arr[i], arr[0]

        # call max heapify on the reduced heap
        heapify(arr, i, 0)

    return arr
```



Time Complexity

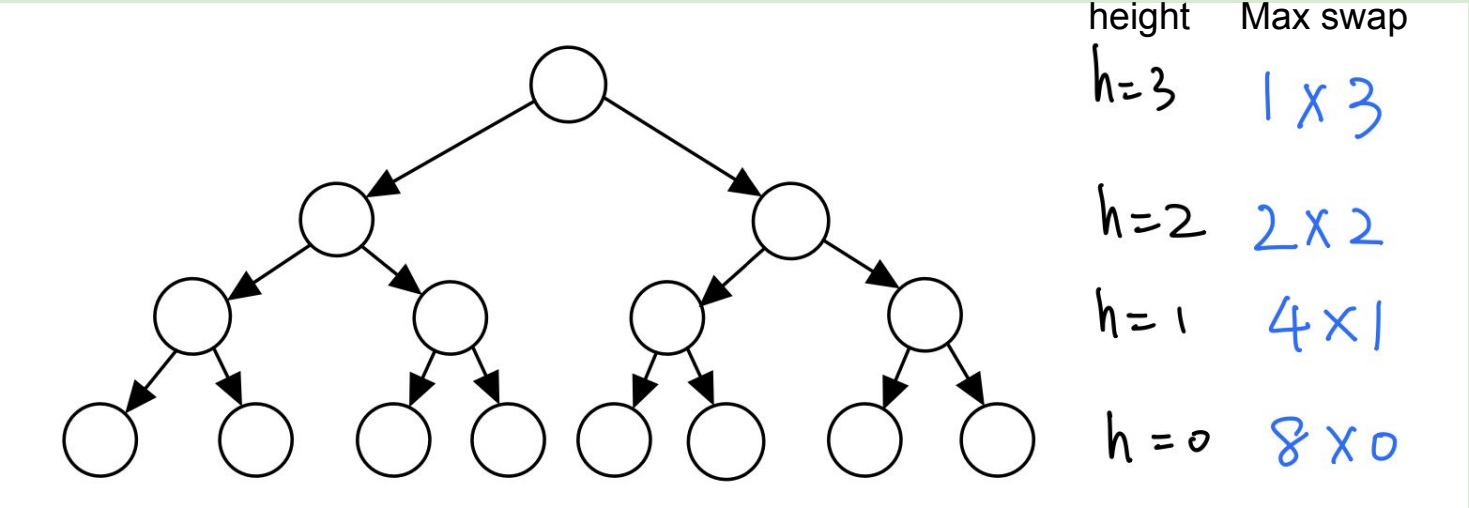
- Build Max Heap
- Max Heapify

Build Max Heap – O(n)

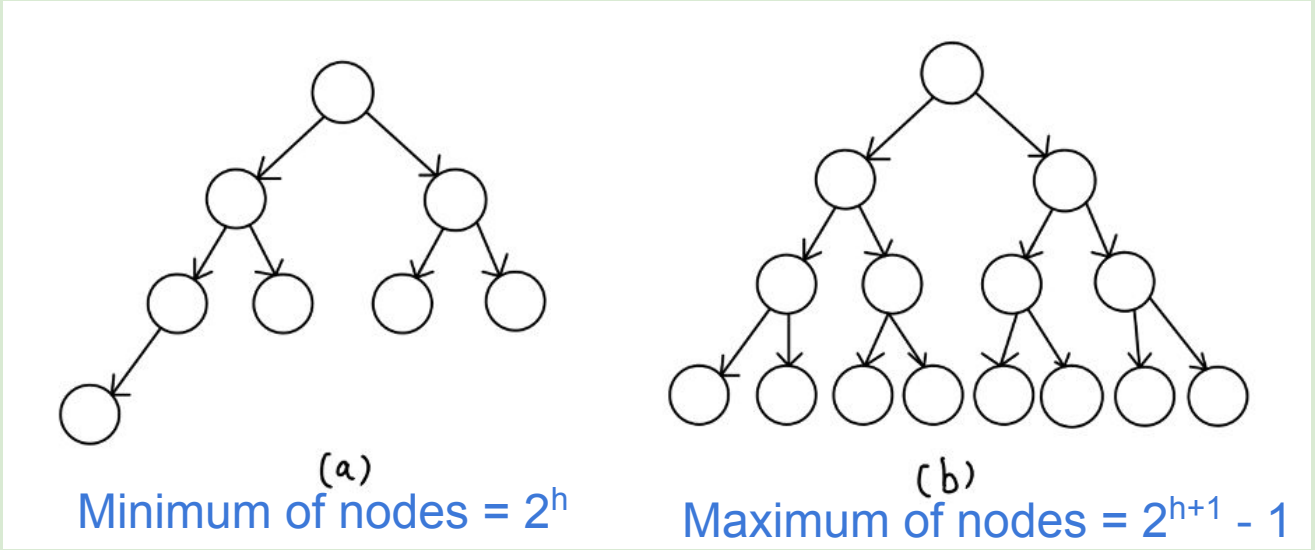
Assume Total nodes $n = 2^k - 1$

Max Swap: $S = (2^k - 1) - k$

Therefore, $S < n$



Max Heapify – O(n log n)



Minimum of nodes = 2^h

Maximum of nodes = $2^{h+1} - 1$

$2^h \leq n$ (total number of node) $\leq 2^{h+1} - 1 \longrightarrow$ $h \leq \log n < h + 1$

Total Time Complexity – O(n log n)

```
HEAPSORT(A)
1    BUILD-MAX-HEAP (A)                              O(n)
2    for i = A.length downto 2                       O(n)
3         exchange A [1] with A [i]                  O(1)
4         A.heap-size = A.heap-size – 1              O(1)
5         MAX-HEAPIFY (A, 1)                         O(log n)
```

O(n) + O(n log n) ⟶ O(n log n)

Comparison

Winner

	Heap Sort	**Bubble Sort**
Best case	O(n log n)	O(n)
Average case	O(n log n)	$O(n^2)$
Worst case	O(n log n)	$O(n^2)$


Շնորհակալություն
Teşekkür ederim
Terima kasih
Merci
ขอบคุณ
Gratiam ago
Grazie
Cảm ơn
ধন্যবাদ।
شکرًا
Спасибо
நன்றி
Mulţumesc
多謝
To-siā
Danke
धन्यवाद।
Asante
Děkuji
Ďakujem
Dankie
nanri
Köszönöm
Tak
Gràcies
Thank you
Tack
謝謝
Hvala
唔該
Dankon
Takk
Diolch
dhanyavaad
감사합니다
Kiitos
谢谢
Faleminderit
Obrigado
Paldies
Bedankt
Dziękujemy
Mèsi
ありがとう
Ви благодарам
Gracias
Lóo-lát
Σας ευχαριστούμε