# BINARY HEAPS

*Mahsa Ziraksima*
*ACIBADEM UNIVERSITY*
*Fall 2025*

# SUMMARY

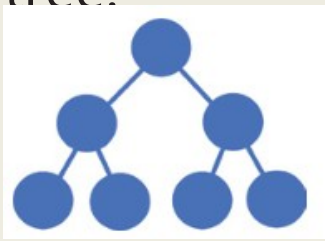
- Performance
- Connecting Ropes with Minimal Cost

# HEAPS

- A priority queue is an abstract data type that describes a data structure where each piece of data has a priority.
- It removes the data with the highest priority first, followed by the subsequent highest priority data.
- A **heap** is a tree-based data structure in which each node keeps track of two pieces of information:
  - *its value or key, and*
  - *its priority.*

# BINARY HEAP

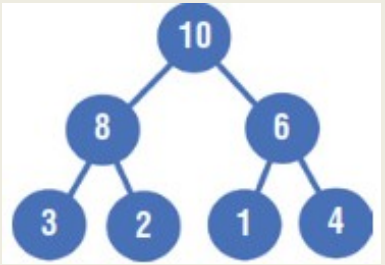
- A binary heap is a heap that you create using a binary tree.



- It has two types:

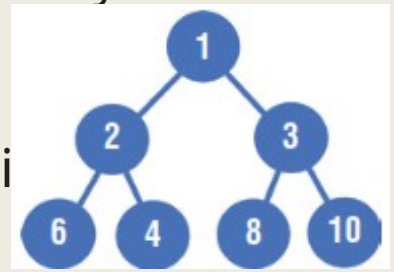
  –*max heaps*

  parent node's priority is always greater than or equal to any child node's priority, and the node with the highest priority is the tree's root.




  –*min heaps*

  parent node's priority is always less than or equal to any child node's priority, and the node with the lowest priority is the root of the tree.




- the ordering applies only to a parent node and its chi

  There is no sorting between sibling nodes

# HEAPIFYING

- It is the process of creating a heap from a data structure like an array.

```
["R", "C", "T", "H", "E", "D", "L"]
```


- Computer scientists often store heaps in arrays, by distributing the keys into a list based on their position in the tree.
- For any node, k, its left child's index is 2k + 1, and the right child's index is 2k + 2


```
from heapq import heapify, heappop, heappush


a_list = ['R', 'C', 'T', 'H', 'E', 'D', 'L']
heapify(a_list)
print(a_list)

heappop(a_list)
print("After popping")
print(a_list)

a_list = ['D', 'E', 'L', 'H', 'R', 'T']
heapify(a_list)
print(a_list)
while len(a_list) > 0:
    print(heappop(a_list))
print("After popping")
print(a_list)

a_list = ['D', 'E', 'L', 'H', 'R', 'T']
heapify(a_list)
heappush(a_list, "Z")
print(a_list)
```

```
['C', 'E', 'D', 'H', 'R', 'T', 'L']
```

```
After popping
['D', 'E', 'L', 'H', 'R', 'T']
```

```
['D', 'E', 'L', 'H', 'R', 'T']
D
E
H
L
R
T
After popping
[]
```

```
['D', 'E', 'L', 'H', 'R', 'T', 'Z']
```

# PERFORMANCE

- finding the maximum or minimum value in a max or min heap is constant time.
- Removing the minimum node from a min heap or the maximum node from a max heap is logarithmic because after you remove the item, you have to balance the remaining nodes.
- Inserting data into a heap is logarithmic.
- searching for data in a heap is O(n).
- Heaps are helpful anytime you have to execute tasks according to priority.

# CONNECTIN ROPES WITH MINIMAL COST

- Challenge: given a list of different rope lengths, find a way to connect all of the ropes, two at a time, in the order that results in the lowest total cost. The cost of connecting two ropes is their sum, and the total cost is the sum of connecting all the ropes.

```
[5, 4, 2, 8] # 8 + 2 = 10
[5, 4, 10] # 10 + 4 = 14
[5, 14] # 5 + 14 = 19
# 10 + 14 + 19 = 43
```

```
[5, 4, 2, 8] # 4 + 2 = 6
[5, 8, 6] # 6 + 5 = 11
[8, 11] # 8 + 11 = 19
# 6 + 11 + 19 = 36
```

# CONNECTIN ROPES WITH MINIMAL COST

```
from heapq import heappush, heappop, heapify


def find_min_cost(ropes):
    heapify(ropes)
    cost = 0
    while len(ropes) > 1:
        sum = heappop(ropes) + heappop(ropes)
        heappush(ropes, sum)
        cost += sum
    return cost

ropes = [5, 4, 2, 8]
print(find_min_cost(ropes))
```

36