HEAP 2

Topic:

- Library called ‘**heapq**’

We use the library heapq whenever we solve problems on heap. There is no need to build heap every time. It’s for knowledge and interview purpose to learn and understand how to make a heap.

In order to use heapq library, we need to import it into our python file using `import heapq`.

```
import heapq
```

Few methods that are important for us:

Heappush	→ to add some element into the heap
Heappop	→ to remove element from the heap
Heapify	→ to make a random array (min-heap).

This will always create a min heap

Description:

Heaps are arrays for which a[k] < a[2*k+1] and a[k] <= a[2*k+2] for all k, counting elements from 0. For the sake of comparison, non-existing elements and considered to be infinite. The interesting property of a heap is that a[0] is always its smallest element.

Usage:

`heap = []`	# creates an empty heap
`Heappush (heap, item)`	# pushes a new item on the heap
`item = heappop(heap)`	# pops the smallest item from the heap
`item = heap[0]`	# smallest item on the heap without popping it
`heapify(x)`	# transforms list into a heap, in-place, in linear time
`item = heapreplace(heap, item)`	# pops and returns smallest item, and adds new item.

Creating n Empty Heap:

```
import heapq

if __name__ == "__main__":
    heap = []
    heapq.heappush(heap, 2)
    heapq.heappush(heap, 12)
    heapq.heappush(heap, 222)
    heapq.heappush(heap, 1)
    heapq.heappush(heap, 5)

    # a min heap will be created with items 2, 12, 222, 1, 5
    print(heap)
```

OUTPUT:

```
[1, 2, 222, 12, 5]
```

To pop the first element of an array, we use **heapqpop(heap)**

```
import heapq

if __name__ == "__main__":
    heap = []
    heapq.heappush(heap, 2)
    heapq.heappush(heap, 12)
    heapq.heappush(heap, 222)
    heapq.heappush(heap, 1)
    heapq.heappush(heap, 5)

    # a min heap will be created with items 2, 12, 222, 1, 5
    print(heap)

    x = heapq.heappop(heap)
    print(x)
```

OUTPUT:

```
[1, 2, 222, 12, 5]
1
```

```
    print(heap)

    x = heapq.heappop(heap)
    print(x)
    x = heapq.heappop(heap)
    print(x)
```

OUTPUT:

```
[1, 2, 222, 12, 5]
1
2
```

Using “heapify”

```
import heapq

if __name__ == "__main__":
    heap = [2,3,11,31,44,556,11]
    heapq.heapify(heap)
    print(heap)
```

OUTPUT:

[2, 3, 11, 31, 44, 556, 11], this is a min heap.

Now, how to change this into max heap?

If we put a negative value and when we are popping the elements if we multiply it with ‘-1’, we will get max heap.

```
import heapq

if __name__ == "__main__":
    heap = []
    heapq.heappush(heap, -2)
    heapq.heappush(heap, -12)
    heapq.heappush(heap, -222)
    heapq.heappush(heap, -1)
    heapq.heappush(heap, -5)

    # a min heap will be created with items -2, -12, -222, -1, -5
    print(heap)

    x = heapq.heappop(heap)
    print(-1*x)
    x = heapq.heappop(heap)
    print(-1*x)
```

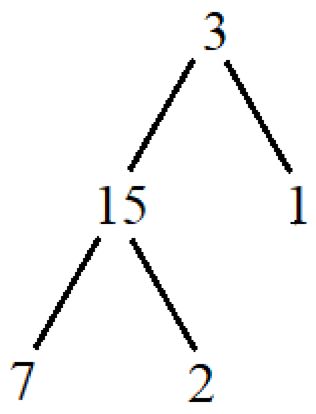
OUTPUT:

```
[-222, -5, -12, -1, -2]
222
12
```

Q) Given an array, find the kth smallest element

[3, 15, 1, 7, 2]

Create a heap, that will create a min heap and the 1st time we pop we get the smallest element and then on the 2nd pop we get the 2nd smallest element.



3 15 1 7 2

k = 3

Our first step would be to create a heap of this array. Then, if we pop once we will get the first smallest because this would always make a **min heap**. If we pop twice, then we will get the second smallest element. So if we pop ‘k’ times, we will get the kth smallest element.

Heap only guarantees you that the top element is the smallest element.

CODE:

```python
import heapq

if __name__ == "__main__":
    l = [5, 6, 2, 12, 3, 9]
    k = 2

    heap = []
    for item in l:
        heapq.heappush(heap, item)

    cnt = 0
    ans = None

    while cnt < k:
        ans = heapq.heappop(heap)
        cnt += 1

    print(ans)
```

OUTPUT:

```
3
```

Time complexity to create a heap is ***O(n)*** and we are pop ‘k’ times, so the time complexity would be ***O(klogn)***.

$$O(n) < O(klogn)$$

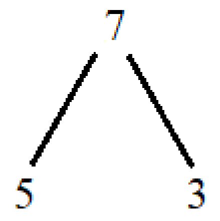
So, the time complexity would be ***O(klogn)***

Space Complexity: the size of the heap is ‘n’, so we are using ***O(n)*** space.

If there are millions of values in a heap, then we will be using huge space. ‘n’ is of the huge range, and we have a very small k. For a small k, and huge ‘n’ we will be still using n space, which is very big.

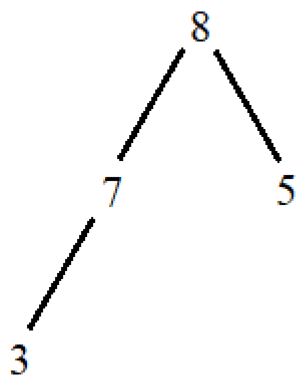
Q) Create a max heap of size k, where k = 3.

[3, 5, 7, 8, 1, 2, 15].



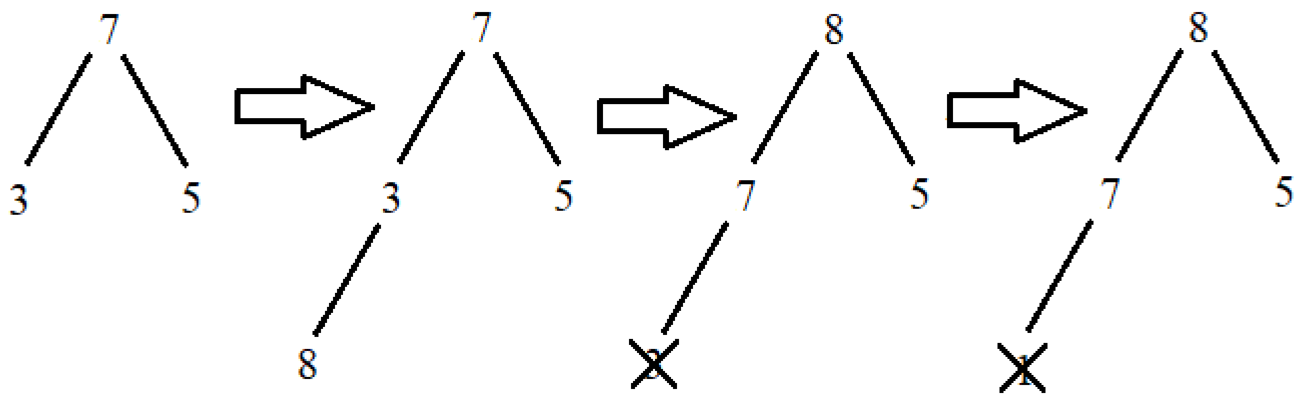
Make sure the size of the heap is always equal to 3, we create the first heap using the first three elements.

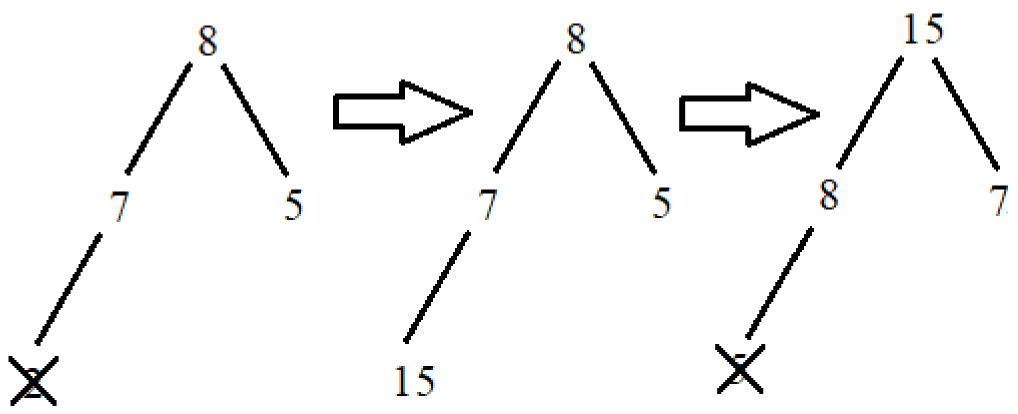
If we add 8 to this heap,



Once this is done, we will pop (3) since it is the size of heap > 3 so to get only k = 3 elements in the heap.

We will add the next element 2, since the size = 4 > 3, last element is popped.





Q) Min Cost of Ropes (geekforgeeks)

https://practice.geeksforgeeks.org/problems/minimum-cost-of-ropes-1587115620/1

4, 3, 2, 6

We are given 4 ropes of length, 4, 3, 2 and 6.

Now, we have to connect al these, ropes and find the total length = 15.

If we connect 4 & 3, the cost to pay is 7. Once this is connected, we will have 3 ropes with lengths, 7, 3 and 6. The cost = 7.

The cost of connecting 7 & 6 would be 13 and cost is increased with 13 i.e., 7 + 13 and we are left with only 2 ropes of length 13 & 2.

The cost of connecting 13 & 2 would be 15 and the cost is increased with 15. So 7 + 13 + 15 = **32**.

But we have to merge the ropes with the min cost.

If we connect the smallest length ropes at each step. So that,

4, 3, 2, 6, we sort it to get 2, 3, 4, 6, and merge, 2 & 3 to get a rope of len = 5 and we will have 3 ropes with lengths 5, 4, 6,

5, 4, 6, we sort to get 4, 5, 6, and merge 4 & 5 to get a rope of length 9 and we will have 2 ropes with lengths 9, 6

9, 6 we sort to get 6, 9 and merge 6 & 9 to get a rope of length **15**.

Time Complexity = Time Complexity for sorting is nlogn and we are doing this, n-1 times. So, the total time complexity would be ***O((n-1) * (nlogn))***

If we merge 2 & 3 = ***5****. Then 4 & 5 =* ***9****. Then 9 & 15 =* ***29****.*

Example 01:

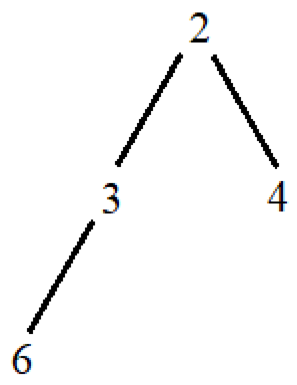
[2, 2, 1, 5, 3, 4]. Sorted array = [1, 2, 2, 3, 4, 5]

[3, 2, 3, 4, 5] ➔ sorted array ➔ [2, 3, 3, 4, 5]

[5, 3, 4, 5] ➔ sorted array ➔ [3, 4, 5, 5]

[7, 5, 5] ➔ sorted array ➔ [5, 5, 7]

The data structure which will always give us the minimum element is heap. So if we create a min heap of these elements,



if we pop once from this heap, we will get 2. If we pop twice, we will get x = 3 (2nd smallest element). This will leave 4 & 6 in the heap.

We can add the smallest element (2) and the second smallest element (3) which will give us a rope of length 5.

We will push it back to the heap, and we will get it, [4, 6, 5]

Now, we will pop the two smallest elements (4 & 5) and will return the sum (9) to the heap giving us [9, 6]

Since there are only two elements, we will add them and the total length of the rope would be 15.

Example 02:

[1, 2, 2, 3, 4, 5] ➔ We will pop two values, (1 & 2) and the cost is 0 initially. We will add 1 & 2 and will get 3 which will be added back to the heap and the cost = 3.

[3, 2 3, 4, 5] ➔ Again, we will pop it twice and we will get 2 & 3. We will merge them and will add that to the cost (5) and push the value back to the heap.

[5, 3, 4, 5] ➔ Again, we will pop it twice and we will get 3 & 4. We will merge them and will add that to the cost (7) and push the value back to the heap.

[7, 5, 5] ➔ Again, we will pop it twice and will get 5 & 5. We will merge them and will add that to the cost (10) and push the value back to the heap.

[10, 7] ➔ Again, we will pop it twice and will get 10 & 7. We will merge them and will add that to the cost (17) and push the value back to the heap.

Now that we have 17, only one element, so the program is done and it will be the cost.

CODE:

```python
import heapq

def minCost(a, n):
    heap = []
    for x in a:
        heapq.heappush(heap, x)

    cost = 0
    while len(heap) != 1:
        x = heapq.heappop(heap)
        y = heapq.heappop(heap)
        cost += x + y
        heapq.heappush(heap, x + y)

    return cost
```