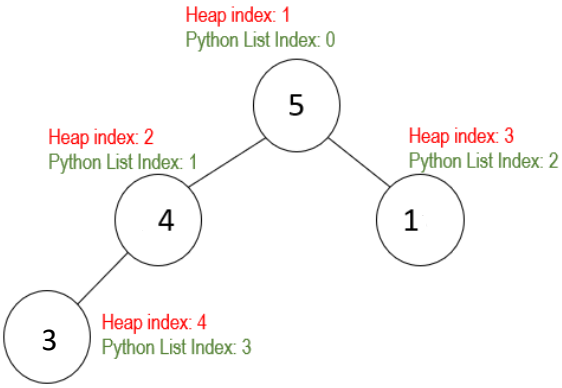
Lab 7 (10 points) **Due date:** April 14th, 2023, 11:59 PM

+3 Extra Credit Points

Before you start…

Keep in mind these tips from your instructor:

- Watch the lectures in Module 6.1 before you start this assignment
- Do not compare yourself to others: An assignment might take you hours, compared to other fellow students, but every student learns in a different way. Please reach out to the course staff id you are stuck and need help
- Outline a strategy before you start typing code and share it with the course staff for additional support
- Note that the index passed into the `_parent`, `_leftChild` and `_rightChild` methods are 1-indexed (as heap indices usually are), but a Python list is 0-indexed. Make sure to subtract 1 in your final index to make up for that difference. Do not leave an empty space in the 0th index of `self._heap`.




```
>>> x = MaxBinaryHeap()
>>> x._heap = [5, 4, 1, 3]  # force load a maximum heap
>>> x._parent(4)  # Heap index 4 (value: 3) has a parent at index
4                 # 4//2 =2, which is list index 1 (x._heap[1])
>>> x._leftChild(2)  # Heap index 2 (value: 4) has a left child at index
3                    # 2 * 2 = 4, which is list index 3 (x._heap[3])
>>> x._rightChild(1) # Heap index 1 (value: 5) has a right child at index
1                    # 2 * 1 + 1 = 3, which is list index 2 (x._heap[2])
```

- Ask questions using our Lab 7 channel in Microsoft Teams

The MaxBinaryHeap class **(8 pts)**

As discussed in the first part of module 6, a binary heap is a complete binary tree that satisfies the heap ordering property and can be implemented with an array. In this assignment, you will be implementing a maximum binary heap using a Python list

Attributes

Type	Name	Description
list	_heap	A list containing the elements of the binary heap

Methods

Type	Name	Description
int or float	_parent(self, index)	Gets the value of the parent of the node at an index
int or float	_leftChild(self, index)	Gets the value of the left child of the node at an index
int or float	_rightChild(self, index)	Gets the value of the right child of the node at an index
None	insert(self, item)	Inserts an item to the maximum heap
int,float	getMax(self)	Gets the maximum value in the heap
int or float	deleteMax(self)	Removes the maximum element of the heap

Special methods

Type	Name	Description
str	__repr__(self), __str__(self)	String representation of this object
int	__len__(self)	The number of elements in the heap

_parent(self, index) **(0.25 pts)**

Returns the value of the parent of the element at the specified index. Please see the notes on page 1 about heap and list indices. **You must use the formulas given in the lectures. No credit will be given if you use 0-indexed adjusted formulas**

Input		
int	index	The heap index to find a child of

Output	
int or float	Value of that element's parent
None	None is returned if that element does not have a parent

_leftChild(self, index) and _rightChild(self, index) **(0.25 pts each)**

Returns the value of the left/right child of the element at the specified index. Please see the above note about heap and list indices. **You must use the formulas given in the lectures. No credit will be given if you use 0-indexed adjusted formulas**

Input		
int	index	The heap index to find a child of

Output	
int or float	Value of that element's left/right child
None	None is returned if that element does not have the specified child

getMax(self) **(0.25 pts)**

A property method that returns the maximum value in the heap. You are not allowed to use the max method or any sorting methods.

Output	
int or float	The minimum value in the heap
None	None is returned if the heap is empty

insert(self, item) **(2 pts)**

Inserts an item into the heap while maintaining the maximum heap property. It must use the _parent(self, index) method, so make sure to test it before using it in this method.

Input		
int or float	item	The value to add to the heap

deleteMax(self) **(3 pts)**

Removes the maximum element of the heap and returns the value of the element that was just removed. The special cases where the heap is empty or contains only one element have been implemented for you already, your task is to implement the general case. As a reminder, according to the maximum heap property, the maximum element is the root node. When percolating down, if both children have the same value, swap (if needed) with the **left child**. It must use the _leftChild and _rightChild methods, so make sure to test them before using them in this method

Output	
int or float	The maximum value of the heap before deletion

__len__(self)

Gets the number of elements in this heap by overloading the len function. This method has already been implemented for you.

Output	
int	Number of elements in the heap

__repr__(self), __str__(self)

Gets the string representation of this object. These methods have already been implemented for you.

Output	
str	Representation of this object as a string.

Additional Grading Functionality **(2pts)**

The grading script will perform a series of mixed heap operations and compare the final status of your heap. The last 2 points are based on the correctness of the heap after the mixed operations are completed. Verify that all methods work correctly when method calls are performed in a mixed and random manner.

Section 2: The heapSort function

As discussed in our video lectures in module 7, heap sort is a comparison-based sorting technique based on the Binary Heap data structure. It builds a heap with the given list and removes the root node until the heap is empty, producing a sorted list. Using your code from Section 1, implement the function heapSort that takes a list of numbers and returns a new list that is sorted.

As a reminder, you are not allowed to use the sorted() method or the sort operator. Your code will not get credit if you use them. Your function must use an instance of the MaxBinaryHeap class to complete the sorting process.

heapSort(numList) **(2 pts)**

Takes a list of numbers and uses the heap sort algorithm to return a list with the elements of numList sorted in descending order. It does not perform an in-place sorting, this means, your code must return a new sorted list without modifying the original one. You are only allowed to use the MaxBinaryHeap operations described in this assignment (len, insert and deleteMax).

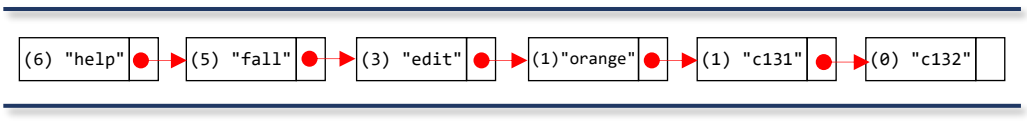
Input		
list	numList	A sequence of numerical values. You cannot make assumptions about its size

Output	
list	Values of numList sorted in descending order using the heap sort algorithm

Section 3: The PriorityQueue class **EXTRA CREDIT (3 pts)**

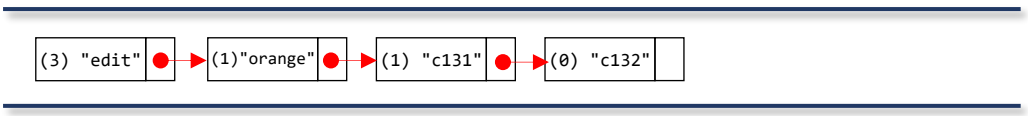
A priority queue, which was introduced in Recitation #8, works like a normal queue except each item is assigned a priority and the items with a higher priority are dequeued first. If we use a linked list-based implementation of the priority queue (where the highest priority is the largest integer), the following code fragment will result in the queue shown in the figure below:

```
>>> q = PriorityQueueLL()
>>> q.enqueue("fall", 5)
>>> q.enqueue("orange", 1)
>>> q.enqueue("edit", 3)
>>> q.enqueue("c132", 0)
>>> q.enqueue("c131", 1)
>>> q.enqueue("help", 6)
```



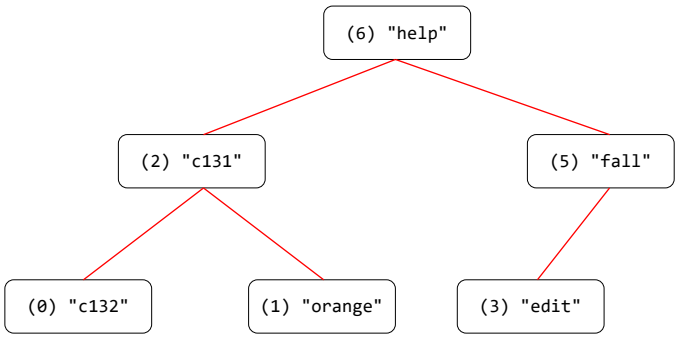
The first item to be removed will be the first item with the highest priority, "help". Notice when items "orange" and "c131" are enqueued, "c131" follows "orange" in the queue even though they have the same priority since items with equal priority still obey the FIFO principle.

```
>>> q.dequeue()
"help"
>>> q.dequeue()
"fall"
```



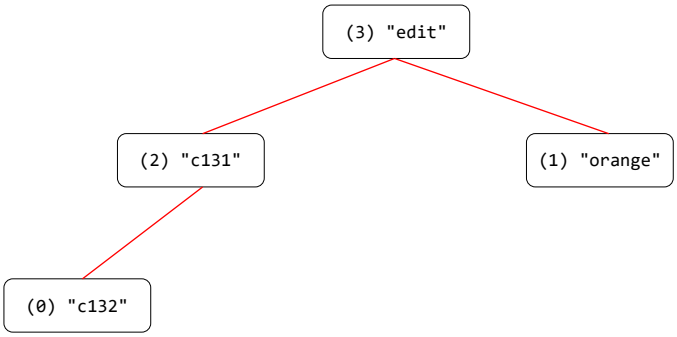
A binary max-heap can also be used to implement the general priority queue where the priority is unique. The ordering of the heap nodes is based on the priority associated with each item in the queue. For example:

```
>>> q = PriorityQueue()
>>> q.enqueue("fall", 5)
>>> q.enqueue("orange", 1)
>>> q.enqueue("edit", 3)
>>> q.enqueue("c132", 0)
>>> q.enqueue("c131", 2)
>>> q.enqueue("help", 6)
```



Since higher integer values indicate a higher priority, the item with the highest priority will always be in the root of the max-heap. When that item is dequeued, the item with the next highest priority will work its way to the top as the percolate-down operation is performed.

```
>>> q.dequeue()
"help"
>>> q.dequeue()
"fall"
```



In this assignment, you will be implementing a PriorityQueue using a maximum binary heap. As stated in Recitation #8, the heap does not play fair when several tasks are of equal priority, so for this implementation, you can assume all priorities are unique integers, and values are comparable and of the same data type.

Attributes

Type	Name	Description
MaxBinaryHeap	_items	A max heap containing the elements of queue

Methods

Type	Name	Description
any	peek(self)	Gets the value of the root node
bool	isEmpty()	Determines if the queue has elements
None	enqueue(self, value, priority)	Adds an element to the queue based on priority
any	dequeue(self)	Removes the value with the highest priority

Special methods

Type	Name	Description
str	__repr__(self), __str__(self)	String representation of this object
int	__len__(self)	The number of elements in the heap

IMPORTANT: Each method requires between 1 to 3 lines of code (based on your coding style). To achieve such goal, you must reuse the code from the MaxBinaryHeap by invoking its methods. 2 pts of the score from this class comes for correct functionality (using MaxBinaryHeap methods), the other point comes from the syntax of the code you write. No modifications need to be made in your MaxBinaryHeap to accommodate this class; tuples are also comparable data types:

```
>>> (5, 'alma') > (1, 'hello')
True
>>> (1, 45.65) < (7, 3.1416)
True
>>> (0, 567) > (7, 3)
False
```

peek(self)

Returns the value of the element at the front of the queue. It does not modify the queue. The priority is discarded.

Output	
any	Value of the element at the front of the queue
None	None is returned when the queue is empty

isEmpty(self)

Indicates whether the queue is empty or not.

Output	
bool	True when the queue has no elements, False otherwise

enqueue(self, value, priority)

Adds the tuple (priority, value) to the queue by inserting it in the proper position in the heap based on priority.

Input		
int	priority	Unique integer that represents the priority of the value
any	value	Element to be placed in the queue

Output	
None	None is returned after the operation is completed

dequeue(self)
Removes and returns the value at the front of the queue (the element with the highest priority). The priority is discarded.

Output	
any	Value at the front of the queue (no priority)
None	None is returned when the queue is empty

__len__(self)
Gets the number of elements in the queue by overloading the len function. This method has already been implemented for you.

Output	
int	Number of elements in the heap

__repr__(self), __str__(self)
Gets the string representation of this object. These methods have already been implemented for you.

Output	
str	Representation of this object as a string.