CMPSC 132: Programming and Computation II

Lab 7 (10 points) Due date: November 12nd, 2021, 11:59 PM

Goal: The goal of this lab is for you to gain a deeper understanding of the binary heap data structure by working to implement a minimum heap and using it to implement a sorting algorithms based on heap operations.

General instructions:

- The work in this assignment must be your own original work and be completed alone.
- The instructor and course assistants are available on Teams and with office hours to answer any questions you may have. You may also share testing code on Teams.
- A doctest is provided to ensure basic functionality and may not be representative of the full range of test cases we will be checking. Further testing is your responsibility.
- Debugging code is also your responsibility.
- You may submit more than once before the deadline; only the latest submission will be graded.

Assignment-specific instructions:

- Download the starter code file from Canvas. Do not change the function names or given starter code in your script.
- You are not allowed to use the heap functions from the Python library, the index() method or any built-in sorting methods.
- You are not allowed to modify the given constructor.
- If you are unable to complete a function, use the pass statement to avoid syntax errors

Submission format:

- Submit your code in a file named LAB7.py file to the Lab 7 Gradescope assignment before the due date.
- As a reminder, code submitted with syntax errors does not receive credit, please run your file before submitting.

Section 1: The MinBinaryHeap 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 minimum binary heap using a Python list

Attributes

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

Methods

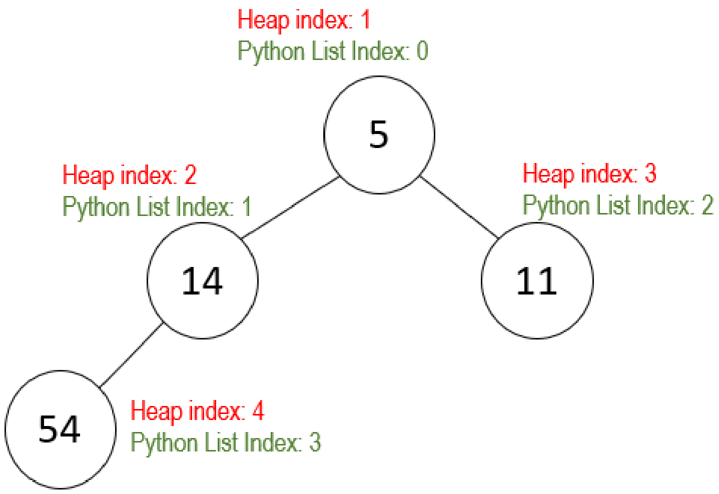
Type	Name	Description
int or float	_parent(self, index)	Gets the value of the parent of the node at an index
int or float	_leftChild(self, index)	Gets the value of the left child of the node at an index
int or float	_rightChild(self, index)	Gets the value of the right child of the node at an index
None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

Section 1: The MinBinaryHeap class

Please 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 adjust your formulas for this (do not leave an empty space in the 0th index of `self._heap`).



```
>>> x = MinBinaryHeap()
>>> x._heap = [5, 14, 11, 54]  # force load a minimum heap
>>> x._parent(4)  # Heap index 4 (value: 54) has a parent at index
14                 # 4//2 =2, which is list index 1 (x._heap[1])
>>> x._leftChild(2)  # Heap index 2 (value: 14) has a left child at index
54                    # 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
11                    # 2 * 1 + 1 = 3, which is list index 2 (x._heap[2])
```

Section 1: The MinBinaryHeap class

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

Returns the value of the parent of the element at the specified index. Please see the above note about heap and list indices.

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 the specified child

_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.

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

getMin(self) **(0.25 pts)**

A property method that returns the minimum value in the heap. You are not allowed to use the min method.

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

insert(self, item) **(3 pts)**

Inserts an item into the heap while maintaining the minimum 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

deleteMin(self) **(4 pts)**

Removes the minimum 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 has been implemented for you already, your task is to implement the general case. As a reminder, according to the minimum heap property, the minimum element is the root node. When percolating down, if both children have the same value, swap 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 minimum 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.

Section 2: The heapSort function

As discussed in our video lecture, 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 MinBinaryHeap 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 ascending order. It does not modify the original list.

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

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