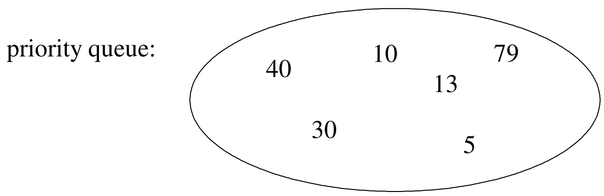
Data Structures (CS 1520) Lab 4 Name:_________________

Objective: To understand priority queue implementations in Python including being able to determine the big-oh of each operation.

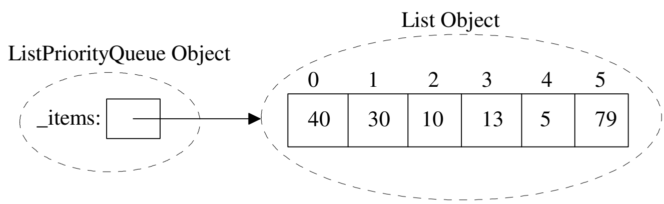
To start the lab: Download and unzip the file at: www.cs.uni.edu/~fienup/cs1520s13/labs/lab4.zip

Part A:

a) Suppose that we have a priority queue with integer priorities such that the smallest integer corresponds to the highest priority. For the following priority queue, which item would be dequeued next?



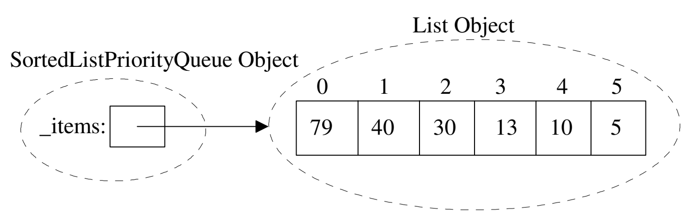
b) The `ListPriorityQueue` implementation in `lab4/list_priority_queue.py` uses an **unorder Python list**.



What would be the big-oh notation for each of the following methods: (justify your answer)

- enqueue:
- dequeue:

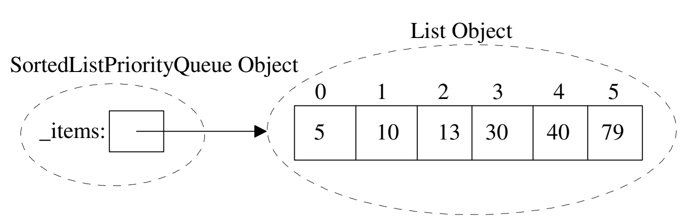
c) The `SortedListPriorityQueue` implementation in `lab4/sorted_list_priority_queue.py` uses a **Python list order by priorities** in decending order.



What would be the big-oh notation for each of the following methods: (justify your answer)

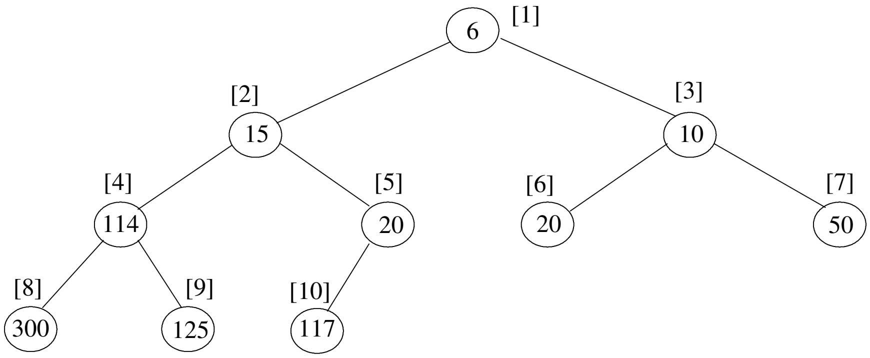
- enqueue:
- dequeue

d) Why would it be a bad idea to implement a priority queue using a **Python list order by priorities** in reverse (ascending) order?



Answer the above questions, then raise your hand. Explain your answers to an instructor or TA.

<u>Part B</u>: (Lecture 7 and) Section 6.6 discusses a very “non-intuitive”, but powerful list/array-based approach to implement a priority queue, call a binary heap. The list/array is used to store a *complete binary tree* (a full tree with any additional leaves as far left as possible) with the items being arranges by *heap-order property*, i.e., each node is ≤ either of its children. An example of a *min* heap “viewed” an a complete binary tree would be:



	0	1	2	3	4	5	6	7	8	9	10
Python List actually used to store heap items	not used	6	15	10	114	20	20	50	300	125	117

a) For the above heap, the list/array indexes are indicated in [ ]'s. For a node at index *i*, what is the index of:

- its left child if it exists:
- its right child if it exists:
- its parent if it exists:

Recall the General Idea of insert(newItem):

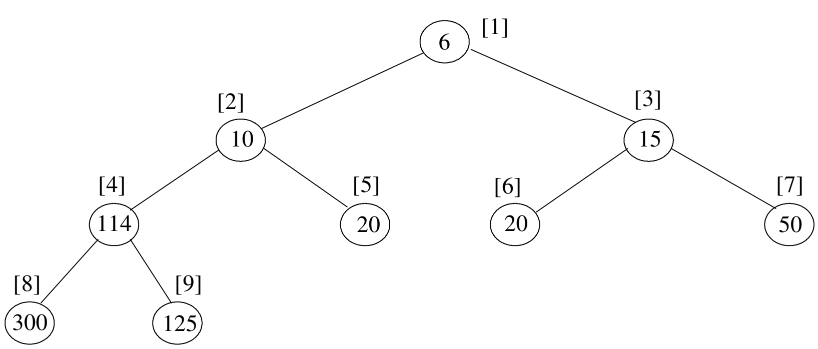
- append newItem to the end of the list (easy to do, but violates heap-order property)
- restore the heap-order property by repeatedly swapping the newItem with its parent until it *percolates up* to the correct spot

b) What would the above heap look like after inserting 30 and then 8? (show the changes on above tree)

c) What is the big-oh notation for inserting a new item in the heap?

Now let us consider the `delMin` operation that removes and returns the minimum item. Recall the General Idea of `delMin()`:

- remember the minimum value so it can be returned later (easy to find - at index 1)
- copy the last item in the list to the root, delete it from the right end, decrement size
- restore the heap-order property by repeatedly swapping this item with its smallest child until it *percolates down* to the correct spot
- return the minimum value



d) What would the above heap look like after `delMin`? (show the changes on above tree)

Answer the above questions, then raise your hand. Explain your answers to an instructor or TA.

Part C: Run the `lab4/timePriorityQueues.py` program that enqueues 20,000 random integers followed by dequeueing all 20,000 integers from various priority queues discussed above. Complete the following timing table from the output of `timePriorityQueues.py`.

Priority Queue Implementation	**Execution Time in Seconds**	
	Enqueuing 20,000 Random ints	**Dequeuing 20,000 ints**
Unsorted Python list		
Sorted Python list in descending order		
"Reverse" sorted Python list in ascending order		
Binary heap stored in a Python list		

b) Why does it take more time to enqueue 20,000 items in the "unsorted" Python list version than dequeue 20,000 in the sorted Python list version?

c) Why does it take more time to dequeue 20,000 items in the heap version than enqueue 20,000 in the heap version?

d) Why is the heap implementation of the priority queue considered "better" than the other three?

After you have answered the above questions, raise your hand and explain your answers.

If you have extra time, complete lab 3 or work on homework #2!