

Question 1. (4 points) Consider the following Python code.

```
i = n
while i > 1:
    print(i)
    i = i // 2

for j in range(n * n):
    print(j)
```

What is the big-oh notation $O()$ for this code segment in terms of n ?

Question 2. (4 points) Consider the following Python code.

```
for i in range(n):
    for j in range(n):
        k = 1
        while k < n:
            print(i, j, k)
            k = k + 3
```

What is the big-oh notation $O()$ for this code segment in terms of n ?

Question 3. (4 points) Consider the following Python code.

```
def main(n):
    i = n
    while i > 0:
        for j in range(n):
            doSomething(n)
        i = i // 2

def doSomething(n):
    for k in range(n):
        doMore(n*n)

def doMore(n):
    for j in range(n):
        print(j)

main(n)
```

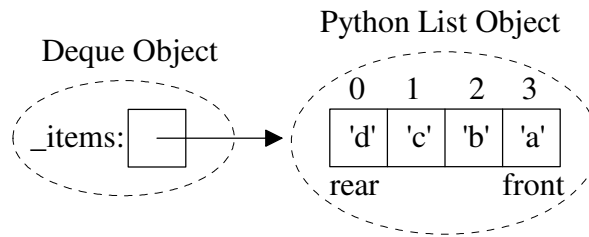
What is the big-oh notation $O()$ for this code segment in terms of n ?

Question 4. (5 points) Suppose a $O(n^5)$ algorithm takes 1 second when $n = 1000$. How long would the algorithm run when $n = 10,000$?

Question 5. (8 points) Why should medium/large size programs be written using function definitions instead of a single block of monolithic code written at the top-level (i.e., all statements written outside of any function)?

Question 6. A Deque (pronounced “Deck”) is a linear data structure which behaves like a double-ended queue, i.e., it allows adding or removing items from either the front or the rear of the Deque. One possible implementation of a Deque would be to use a built-in Python list to store the Deque items such that

- the **rear** item is **always stored at index 0**,
- the front item is always at index $\text{len}(\text{self}._items) - 1$ or -1



a) (6 points) Complete the big-oh $O()$, for each Deque operation, assuming the above implementation. Let n be the number of items in the Deque.

<code>isEmpty</code>	<code>addRear</code>	<code>removeRear</code>	<code>addFront</code>	<code>removeFront</code>	<code>size</code>

b) (9 points) Complete the code for the `addRear` method, including any precondition check needed by raising an exception if it is violated.

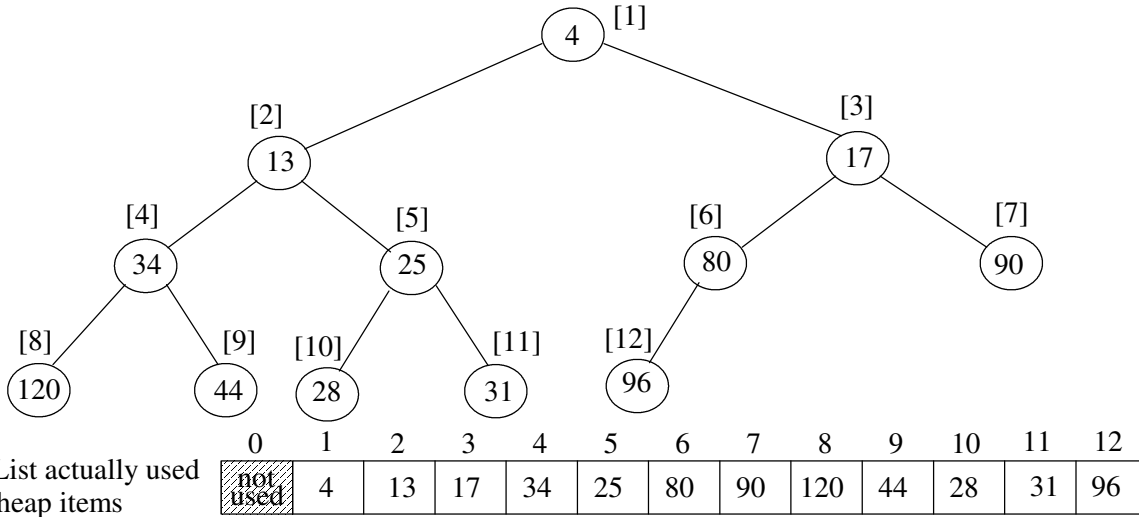
```
def addRear(self, newItem):
    """Adds the newItem to the rear of the Deque
    Precondition: none
    Postcondition: newItem has been added to the rear of the Deque"""
```

c) (10 points) Complete the method for the `__str__` operation.

```
def __str__(self):
    """Returns the string representation of the Deque.
    Precondition: none
    Postcondition: Returns a string representation of the Deque from the
    front item thru the rear item with a blank space between each item."""

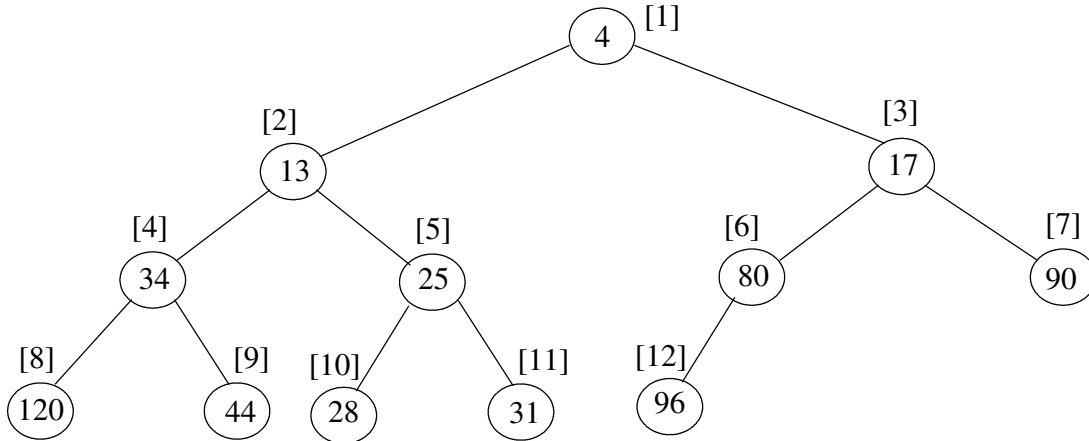
    resultStr = "(front) "
```

Question 7. Consider the binary heap approach to implement a priority queue. A Python list is used to store a *complete binary tree* (a full tree with any additional leaves as far left as possible) with the items being arranged by *heap-order property*, i.e., each node is \leq either of its children. An example of a *min* heap “viewed” as a complete binary tree would be:



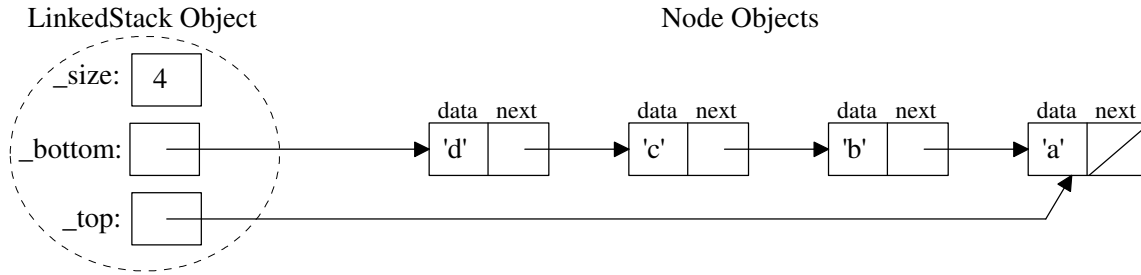
- a) (3 points) For the above heap, the list indexes are indicated in []'s. For a node at index i , what is the index of:
- its left child if it exists:
 - its right child if it exists:
 - its parent if it exists:
- b) (7 points) What would the above heap look like after inserting 10 **and then** 20 (show the changes on above tree)

Now consider the `delMin` operation that removes and returns the minimum item.



- c) (2 point) What item would `delMin` remove and return from the above heap?
- d) (7 points) What would the heap look like after `delMin`? (show the changes on tree in the middle of the page)
- e) (6 points) What is the big-oh notation for the `delMin` operation? (**EXPLAIN YOUR ANSWER**)

Question 8. The Node class can be used to dynamically create storage for each new item added to a Stack using a singly-linked implementation as in:



a) (6 points) Complete the big-oh $O()$, for each `LinkedStack` operation, assuming the above implementation. Let n be the number of items in the `LinkedStack`.

<code>isEmpty</code>	<code>size</code>	<code>pop</code>	<code>push(item)</code>	<code>__init__</code>	<code>__str__</code>

b) (12 points) Complete the `push` method for the above `LinkedStack` implementation.

```
class LinkedStack(object):
    """ Singly-linked list based Stack implementation."""

    def __init__(self):
        self._size = 0
        self._bottom = None
        self._top = None

    def push(self, item):
        """ Adds the item to the top of the Stack.
        Precondition: none """
```

```
class Node:
    def __init__(self, initdata):
        self.data = initdata
        self.next = None

    def getData(self):
        return self.data

    def getNext(self):
        return self.next

    def setData(self, newdata):
        self.data = newdata

    def setNext(self, newnext):
        self.next = newnext
```

c) (7 points) Suggest an improvement to the above implementation to speed up some of the stack operations enough to change their big-oh notation? (Justify your answer)