Igor Hamburger

Question 1:

In Lower Bounds we search for the best case for a time complexity of a function. Since we’re interested in the asymptomatic behavior of the growth of the function, the factors (-3n^2) and

(-logn) can be ignored since they’re less relevant considering that: Omega(g(n)) = c \* n, for c and n0 satisfying 0 <= cg(n) < f(n) for all n >= n0.

Question 2:

[3, 4, 16, 17, 29, 18, 35, 19]

First we remove the root of the Minimum Heap (2), than to add a new element to the heap we first add it to the end of the list and we do Percolate Up, if the ‘father’ of the new element has a bigger value, we change both positions until the new element is in the right place in the Minimum Heap.

Question 3:

A –

After calling the my\_mistery function on the given array, it will be sorted: [5, 7.5, 15, 20, 30]

B – Since we have two for loops, the external one has a time complexity of O(n), and the inner one also has a time complexity of O(n), therefore the time complexity for the whole function is O(n^2).

Question 4:

Class Node:

def \_\_init(self, data, next = None):

self.data = data

self.next = next

Class LinkedList:

def \_\_init(self):

self.head = None

self.total\_value = 0

self.size = 0

def add\_to\_lst(self, x):

if self.head == None:

self.head = x

self.total\_value += x.data

self.size += 1

else:

cur = self.head

while cur.next != None:

cur = cur.next

cur.next = x

self.total\_value += x.data

self.size += 1

def get\_avg\_value(self):

return self.total\_value // self.size

With this algorithm its possible to get the sum of the values of the list just by calling the function get\_total\_value that return the total value that is updated at every call of the function add\_to\_lst. Therefore the time complexity for this function is O(1).

But to add a Node to this function would take O(n), but a solution for this would be using a value as tail, representing the end of the list, than to add an element would take O(1) time complexity.