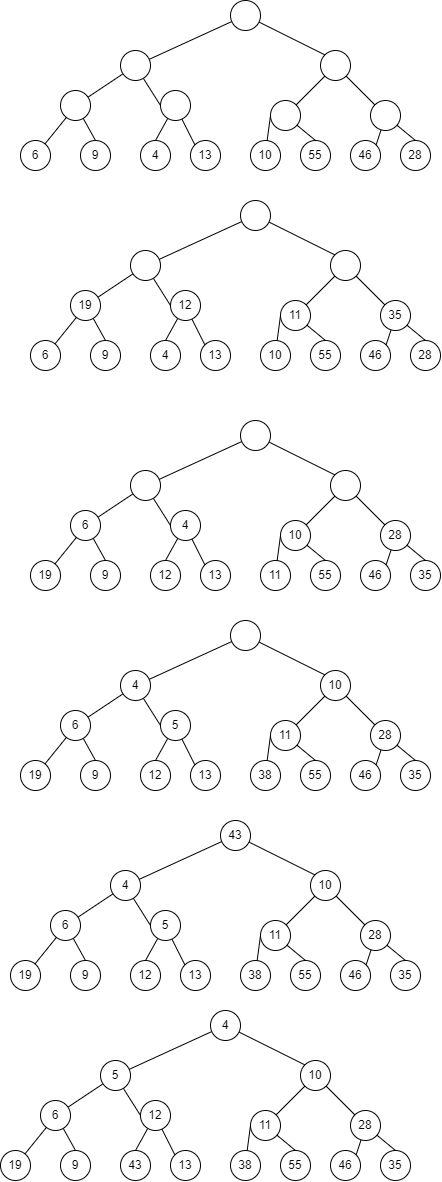
2022

Q1  
a)  
Min: height 10 (having going all way right)  
Max: 4 (having proper binary tree)  
  
b)  
1.  
i 31,21, 4,2,10,29,48,43,49  
ii 2,10,4,29,21,43,49,48,31  
iii 2, 4,10,21,29,31,43,48,49  
iv 31, 21, 4, 2, 4, 10, 4, 21, 29, 21, 31, 48, 43, 48, 49, 48, 31

2. 2

3. Yes  
4. On the left of 29 or right of 10   
5. You would go one right from 31, then go as left as possible until you reach 34. Then replace the value 31 with 43, keeping the 21 and 48 as children of 43 now.

Q2  
a) Priority Queue ADT stores a collection of Entries. An entry is a Key and Value.  
The goal of a priority queue is to have the most used/important entries being the first/fastest to access. Therefore the key indicates how important/priority of that entry ie the lower the key value, the higher the priority of it. The value signifies the actual object stored in the Entry.  
  
In a sorted sequence:   
The insert: O(n)  
The remove min + min: O(1)  
  
b)  
2 distinct entries, can have the same key  
Keys can be arbitory objects, that define the order   
The 3 math things.  
c)  




Q3  
a)

Put (key, value)  
  
Go through all positions p in list  
if p.element.getKey == key  
 V temp = p.element.getValue  
 list.set(p, (k,v))

Return temp  
  
list.addLast((k,v))  
size++;  
return null

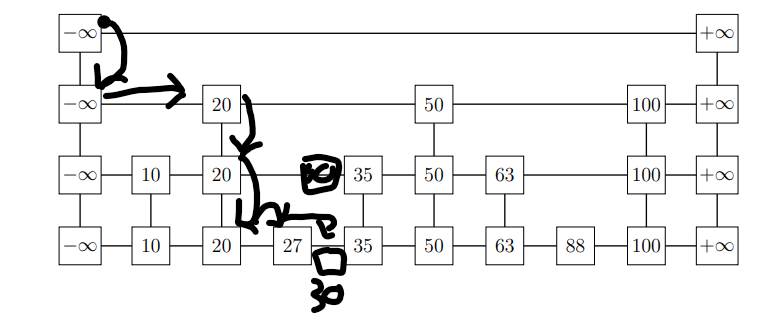
b)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 35 |  | 30 | 10 |  | 5 | 6 | 20 | 27 | 40 |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

c)  
8/12

Q4  
a)  
For List: Worst case for get/remove, that element does not exist and must traverse entire list.  
Put: can add to first/end of list  
  
For Hash: Worst case for get/remove, that element does not exist and must traverse entire list, where every value is on the same array square List.  
Put: add to square, else if collide, add to first/end of list

|  |  |  |
| --- | --- | --- |
|  | list | Hash |
| Get | O(n) | O(n) |
| Put | O(1) | O(1) |
| remove | O(n) | O(n) |

b)  


We know that the number of nodes on a new level, is around half the previous level. This is due to there being a 50/50 chance of moving up or not.  
  
So we can see that te total nodes will be n/2^0 + n/2^1 + … 1 < 2n  
Therefore, we can see there will be a height of log2n + 2  
The +2 counts the first level, and the level with no values on it.  
  
This can be simplifies to O(logn)

Q5