Task 5.1 Message : NSA IS CIA IS IRA
Key : LA CHI CIA
Encrypted Message: LA PW IRA CIA WIFI

Task 5.2 Find the value of i % l, call it *keyIndex*.
$w_{ci} = w_{mi+keyIndex\%l}$

Task 5.3 $w_{mi} = w_{c(l+(i-keyIndex))\%l}$

Task 7.1 Figure 1

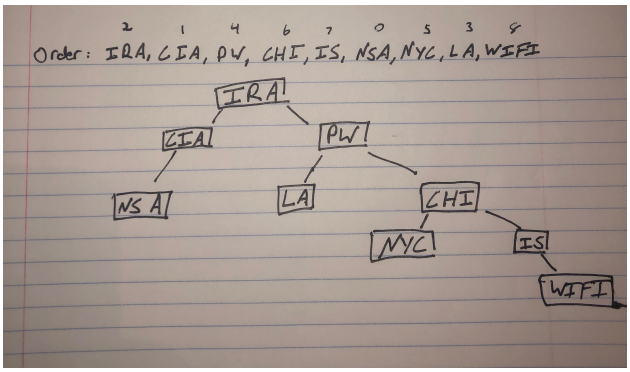

Figure 1: Task 7.1

Task 7.2 First Blank : 0
Second Blank: num_trees(i) * num_trees(n-i-1)

Task 7.3 Field to be added: int size;
The field represents the size of the tree rooted to a given node. Every time a new node is inserted into a tree the size of that tree is incremented

Task 7.4 The Solution has a runtime that is proportional to Log N (where N is the number of Nodes in the tree) because (in the average case) that is the number of steps it takes to reach a leaf of the tree from the root of the tree.
First Blank:
T → adress = addr;
T → distance = dist;
T → price = price;
T → left = NULL;
T → right = NULL;
T → size = 1;
Second Blank:
T → left = insert(T → left, addr, dist, price);
T → size ++;
Third Blank:
T → right = insert(T → right, addr, dist, price);
T → size ++;

Task 7.5 Search Protocol of range(T, d1, d2);
If T == NULL, return 0. Otherwise Proceed to step 0
Step 0 Look at the distance stored in the tree. If is greater than d2, run the search protocol on the left subtree (return range(T → left), d1, d2)). If it is less than d1, run the search protocol on the right tree (return range(T → right, d1, d2)). If it is greater than or equal to d1 and less than or equal to d2 proceed to step 1.
Step 1 Store the size of the tree in a value called $size_{total}$ and proceed to step 1a.
Step 1a Move down the left edge of the tree until you encounter a node with a distance that is less than d1. If no such node is encountered move on to step 1b. Otherwise, subtract the size field of this node from $size_{total}$ and move on to step 1b.
Step 1b Move down the right edge of the tree until you encounter a node with a distance that is greater than d2. If no such node is encountered move on to step 2. Otherwise, subtract the size field of this node from $size_{total}$ and move on to step 2.
Step 2 Return $size_{total}$

This solution runs in O(log N) time because it requires going down two paths of the tree in its worst case, which each take 0 (Log N) time (The height of a tree is Log(N))

Task 7.6 I don't know Claude you tell me

Task 8.1 Figure 2

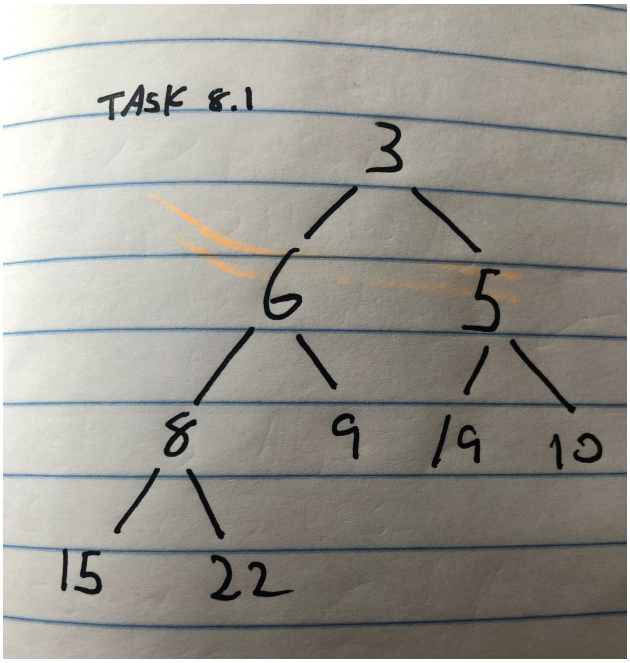

Figure 2: Task 8.1

Task 8.2 Figure 3

Task 8.3 Figure 4

Task 8.4 **First Blank:** p < (H → priorities) [idx] || idx > size

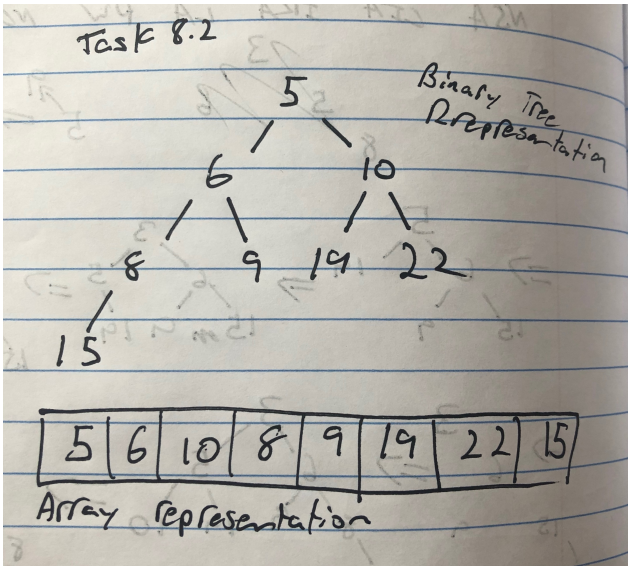

Figure 3: Task 8.2

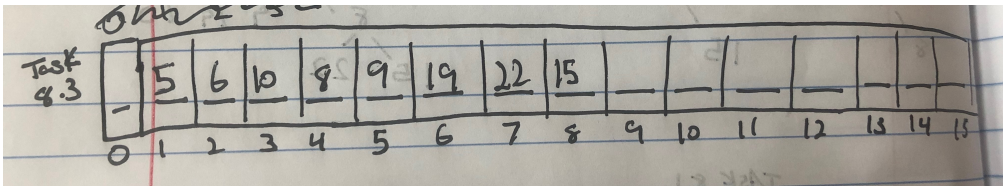

Figure 4: Task 8.3

Second Blank: p == (H → priorites) [idx]

Third Blank:

Fourth Blank: return heap_search(H, p, 2*idx) || heap_search (H, p, 2*idx+1)

Task 8.5 The data structure has three fields. The first one is a median value. The last two are two heaps, a min heap and a max heap. Every element in the min heap is greater than or equal to the median. Every element in the max heap is less than the median.
Track the size of both heaps.
Finding the median involves the follwing steps:
Get the value of the median field stored the the structure. This has O(1) time because it is just looking up a value at a known location.
Deleting the median involves the following steps:
If the sizes of the two heaps are equal, make the median field equal the min value of the min heap. Delete the min value of the min heap. If the size of the min heap is greater than the size of the max heap, make the median field equal the min value of the min heap. Delete the min value of the min heap. If the size of the max heap is greater than the size of the min heap, make the median field equal the max value of the max heap. Delete the max value of the min heap. Delete only takes O(Log N) time because it only involves deleting root value from heaps, which has O ((Log N)/2) complexity.