Problem 1:

22 15 36 44 10 3 9 13 29 25

Find the minimum item, swap it with current item in index 0

3 15 36 44 10 22 9 13 29 25

Find the second biggest item, swap it with item in index 1…

3 9 36 44 10 22 15 13 29 25

3 9 10 44 36 22 15 13 29 25

3 9 10 13 36 22 15 44 29 25

3 9 10 13 15 22 36 44 29 25

Item is already in the right place, no swap.

3 9 10 13 15 22 36 44 29 25

3 9 10 13 15 22 25 44 29 36

3 9 10 13 15 22 25 29 44 36

3 9 10 13 15 22 25 29 36 44

Done.

Problem 2:

22 15 36 44 10 3 9 13 29 25

15 22 36 44 10 3 9 13 29 25

15 less than 22, insert 15

15 22 36 44 10 3 9 13 29 25

15 22 36 44 10 3 9 13 29 25

36 and 44 both already in the right order, keep moving for both

10 15 22 36 44 3 9 13 29 25

10 less than everything, insert it in front

3 10 15 22 36 44 9 13 29 25

3 9 10 15 22 36 44 13 29 25

3 9 10 13 15 22 36 44 29 25

3 9 10 13 15 22 29 36 44 25

3 9 10 13 15 22 25 29 36 44

Continue inserting each item into the right place until the end.

Problem 3:

```
Algorithm update_last (T):
    {we need to go to the rightmost deepest node in the tree, and set that to last}
    current <- T.root
    {start at root of the tree}
    n <- len(T)
    h <- floor(log2(len(T))
    {find the height of the tree}
    full <- 2^h
    remaining <- n – full
    pointer = 2^(h-1)
    {use these pointers to figure out what direction to go}
    if h = 0:
        T.last <- T.root
        return
    else:
        While  pointer > 0:
            if pointer > remaining
                current <- current.left
                {go left}
            else:
                current <- current.right
                remaining <- remaining – pointer
                {go right and update pointer}
            pointer // 2
        T.last <- current
```

return

Problem 4:

Algorithm return_less_than_k (T, k):

output = []

{initialize an output array}

helper(T,k,1, output)

{run helper function and return output}

return output

Algorithm helper(T,k,i,output):

{use a helper function so public function takes less arguments}

if i > len(T):

return

{we've reached the end of the tree}

if T[i] > k:

return

{we've reached a key greater than k}

else:

output += T[i]

helper(T,k,2 * i, output)

helper(T,k,2 * i + 1)

{else we've found a match, add the key to the output and search down the right and left subtrees}

Problem 5:

0	13
1	39, 94
2	
3	
4	
5	11, 44 ,88
6	
7	
8	12, 23
9	5, 16
10	20

Problem 6:

54 28 41 18 10 36 25 38 12 90

0	
1	
2	12
3	18
4	41
5	
6	36
7	25
8	
9	
10	
11	
12	38
13	10
14	
15	90
16	28

Problem 7:

```
Algorithm find_kth_union (S,T,k):
    i -> 0
    found <- 0
    {initialize index and number of found items}
    while i < len(k) do:
        {iterate through S}
        j <- len(T) // 2:
        {do binary search of T to see if there is a match for S[i]
        while j > 0 and j < len(T):
            if T[j] = S[i]:
                found <- found + 1
                {if we find a match increment number of found items}
                if found = k:
                    {when we have found k items in the union return the kth
                     item}
                    return T[j]
            {else keep searching}
            else if S[i] < T[j]:
                j <- (j //2) – 1
            else:
                j <- j + j//2
        i <- i + 1
    return False
    {if we reach the end of both arrays without finding k matches return false}
```