Question 4 solutions:

In order to support duplicate items in the vEB trees, my solution is to add two interger variables: v.n_max and v.n_min which denotes the number of maximum and numbers of minimum in each cluster respectively. By default, these two variables are set to be 0. By using this method, we will have two variables count the numbers of duplicate values and thus the vEB trees can support duplicate keys.

```
// the explanations of the program are listed below//
Insert function:
vEB-Tree-Insert(v,x,number=1):                    // by default, the number of
    if v.min = Nil:                               //items to be inserted is 1.
        vEB-Empty-Tree-Insert(v,x,number)
    else if x=v.min:                              //……①
            v.n_num = v.n_num + number
        else:
            if x<v.min:                           //……②
                exchange x with v.min
                exchange number with v.n_min
            if v.u > 2:                           //……③
                if vEB-Tree_minimum(V.cluster[high(x)]) == NiL
                    vEB-Tree-Insert(v.summary,,high(x))
                    vEB-Empty-Tree-Insert(v.cluster[high(x)], low(x),number)
                else:
                    vEB-Tree_Insert(v.cluster[high(x)],low(x),number)
            if x > v.max:                         //……..④
                v.max = x
                v.n_max= 1
            if x==v.max:
                v.n_max++

vEB-Empty-Tree-Insert(v,x,number=1)        // if a cluster is empty, update the
    v.min = x                              // value and numbers of v.min and
    v.max = x                              // v.max respectively.
     v.n_min = number
    v.n_max = number
```

Explanations:

1. If x is the same with current minimum, then just update the numbers of current minimum value and stop further going down.
2. If x is smaller than the current minimum, according to the property of vEB tree, swap the value and numbers of current minimum value with this insert value (that means let the insert value become the new minimum value in this cluster).

Then, we need to continue insert back the old minimum value.

3. If we do not reach the leaf of the tree, first check if we need to update the summary cluster, then keep inserting recursively
4. If x is equal to the current maximum, update the number accordingly. If x is larger than the current maximum value. Reset the number and value of maximum value to this new inserted one.

For other functions, we also need to check the number of minimum and maximum in the cluster before we do the delete or decide the successor or predecessor.

Since assign the n_max and n_min in each step are all o(1), so still the running time fulfill:

$T(2^m) = T(2^{m/2}) + O(1)$, thus the running time will not change.

For other functions, basically, we also only need to check the n_max and n_min and update them accordingly, those operations are all O(1) , thus the running time should still be the same.