ASSIGNMENT: HW6

NAME: PRASANTH BHAGAVATULA

CWID: A20355611

1. If we are not to store the max value in the ~~table~~ cluster, and just store it in the max value, then there will be change only in the vEB-TREE-INSERT and vEB-TREE-DELETE, only ~~the~~ and the ~~vEB-TREE-IN~~ vEB-EMPTY-TREE-INSERT remains same.

vEB-~~TR~~EMPTY-TREE-INSERT(V, x)

    V.min = x
    V.max = x

vEB-TREE-INSERT (V, x)

    if V.min == NIL
        vEB-EMPTY-TREE-INSERT(V, x)
    else if x < V.min
        exchange x with V.min
    else if x > V.max
        exchange x with V.max
        ~~else if V.u > 2~~
            ~~if vEB-TREE-MINIMUM~~
    if x != V.min and x != V.max
        if V.u > 2
            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))
            else vEB-TREE-INSERT(V.cluster[high(x)], low(x))

In the INSERT algorithm, we adde one more condition to swap the element if that is greater than max and the we compare the element with 'min' and 'max' values. If and only if the value is different from ~~max~~ min and max, ~~the~~ we will insert the value into the cluster.

Similarly, in case of DELETE algorithm we need to add condition to check whether the deleting value is same as ~~no~~ 'max' value and then we need to find the next available max value and place it. The ~~algo~~ algorithm is as follows

```
VEB-TREE-DELETE(V,x)
if V.min == V.max
    V.min = NIL
    V.max = NIL
else if V.u == 2
    if x == 0
        V.min = 1
    else V.min = 0
    V.max = V.min
else if x == V.min
        first-cluster = VEB-TREE-MINIMUM (V.summary)
        x = index(first-cluster, VEB-TREE-MINIMUM(V.cluster[first-cluster]))
        V.min = x
    else if x == V.max
        last-cluster = VEB-TREE-MAXIMUM(V.summary)
        x = index(last-cluster, VEB-TREE-MAXIMUM(V.cluster[last-cluster]))
        V.max = x
    VEB-TREE-DELETE(V.cluster[high(x)], low(x))
```

ASSIGNMENT: HW6
NAME: PRASANTH BHAGAVATULA
CWID : A 20355611

18. if VEB-TREE-MINIMUM (V.cluster[high(x)]) == NIL
19. VEB-TREE-DELETE (V.summary, high(x))
20. ~~if~~

Since, we are checking for 'max' element before and then proceeding, we don't need to ~~delete~~ find the 'max' element again.

ASSIGNMENT : HW6

NAME: PRASANTH BHAGAVATULA

CWID : A20355611

2\. (a) The space requirements $P(u)$ for van Emde Boas tree ~~follo~~ characterizes the following recurrence relation.

$$P(u) = (\sqrt{u}+1)\,P(\sqrt{u}) + \Theta(\sqrt{u})$$

Each vEB-tree structure will have '$\sqrt{u}$' clusters ~~and~~ pointing to '$\sqrt{u}$' vEB-tree structures of universe size $\sqrt{u}$ and one summary structure of universe size '$\sqrt{u}$'. So, there will be a total of $(\sqrt{u}+1)$ vEB-tree structures of universe size '$\sqrt{u}$'. The pointers pointing to sub-structures also needs to be stored which are '$\sqrt{u}$' in number and so, we can represent

$$P(u) = (\sqrt{u}+1)\,P(\sqrt{u}) + \Theta(\sqrt{u})$$

The first part represents the space requirements for the sub-tree structures and the second part $(\Theta(\sqrt{u}))$ represents the space requirement for ~~pointing~~ storing the pointers to sub-structures.

ASSIGNMENT: HW6

NAME: PRASANTH BHAGAVATULA

CWID: A20355611

2 (b). Consider the assumption that the constant in '$\theta(\sqrt{u})$' be '1'.

If we look at the vEB-sub-tree structure with universe size as 2, there are no sub-tree structure further to this and so no array of pointer are also required.

Hence $P(2) = 0$

Consider, the value of $P(4)$

$$P(4) = (\sqrt{4}+1)\, P(\sqrt{4}) + \theta(\sqrt{4})$$

$$\Rightarrow P(4) = 3\,P(2) + 2$$

$$\Rightarrow P(4) = 2$$

$$\Rightarrow P(4) = 4 - 2$$

For all, $k = l^2 \geq 4$,

$$P(k) = (l+1)\, P(l) + l$$

$$\Rightarrow P(k) = (l+1)(l-2) + l$$

$$\Rightarrow P(k) = l^2 - 2l + l - 2 + l$$

$$\Rightarrow P(k) = l^2 - 2$$

$$\Rightarrow P(k) = k - 2$$

$\therefore$ Hence $P(u)$ grows linearly with respect to universe size

and $P(u) = O(u)$

ASSIGNMENT: HW6

NAME: PRASANTH BHAGAVATULA

CWID: A20355611

2.(c) If we store all the ~~array of~~ pointers to sub-structures in in a single array outside of vEB structure, the space requirement ~~recursion~~ recurrence will be as below.

$$P(u) = (\sqrt{u}+1)\,P(\sqrt{u}) + O(1)$$

Even in this case, the vEB-sub-structure of universe size '2' will still have the no pointers and so $P(2) = 0$. Consider the value for $P(4)$

$$P(4) = (\sqrt{4}+1)\,P(\sqrt{4}) + O(1)$$

$$\Rightarrow P(4) = 3P(2) + O(1)$$

$$\Rightarrow P(4) = O(1)$$

We will write the coefficient of '$O(1)$' as $4c_1 + c_2$, which is equivalent to '1'. Similarly $P(16)$ will be

$$P(16) = (\sqrt{16}+1)\,P(\sqrt{16}) + O(1)$$

$$\Rightarrow P(16) = 5P(4) + O(1)$$

$$\Rightarrow P(16) = 6\,O(1)$$

and so $\Rightarrow 16c_1 + c_2 = 6$

Solving both the equations $4c_1 + c_2 = 1$ and $16c_1 + c_2 = 6$, we get $c_1 = 5/12$ and $c_2 = -8/12$

So, we can clearly express

$$P(u) = \left(\left(5/12\right)u - \left(8/12\right)\right) O(1)$$

$$\Rightarrow P(u) = \underline{O(u)}$$

~~This is similar to~~ th

The other way to consider is that there are '$\sqrt{u}$' vEB-sub-tree structures and each one will point to $\sqrt{u}$' sub-tree structures. So a total of "u' subtree structures and if we look at each sub tree structure, we ~~don't~~ nee and since we are not storing the array of pointers, it ~~would~~ the space requirement would be $O(1)$ and t so, the total space requirement would be $O(u)$

The space requirement earlier is also $O(u)$, but the constants will be reduced in the new approach.

ASSIGNMENT: HW6
NAME: PRASANTH BHAGAVATULA
CWID: A20355611

(3) We can implement the mentioned problem and operations using vEB tree. Initialize operation needs to create and insert all elements of S into the vEB tree. The decrease key operation will compare the values of 'x' and 'v(s)' and if the condition is satisfied, we simply delete v(s) and insert 'x'. The minimum (x) will simply return the minimum value from the vEB tree and check if it is less than 'x' and then return it.

INITIALIZE (S)

1. for all $s \in S$
2. vEB-TREE-INSERT (V, v(s))

The above operation will simply insert all values of (v(s)) of $s \in S$ into a vEB tree. Here while creating vEB tree, the size of universe is considered i.e. 'u' is considered as the size of universe of root.

DECREASE-KEY (s, x)

1. If $x < v(s)$
2. vEB-TREE-DELETE (V, v(s))
3. vEB-TREE-INSERT (V, x)

If the condition is satisfied, we need change v(s) to x and so we delete v(s) from vEB tree and insert x into the vEB tree.

MINIMUM (x)

~~1. VEB-TREE-MINIMUM (V)~~

1. min = VEB-TREE-MINIMUM (V)
2. If min <= x
3. return min
4. else return NULL

We will just find the minimum value of the VEB tree and see, if it is less than x and ~~they~~ if it is less than (or) equal to x, we will simply return the value, else we return NULL, indicating that there is no element in S, such that $v(s) \leq x$ where $s \in S$

The INITIALIZE operation calls the VEB-TREE-INSERT operation and so, it takes $O(\log \log u)$ time to execute. The DECREASEKEY operation calls the VEB-TREE-DELETE and VEB-TREE-INSERT operations and so, it also takes $O(\log \log u)$. The MINIMUM operation uses only VEB-TREE-MINIMUM operation and so, it takes $O(1)$ time to execute.

Since, all these operations are executed using VEB trees, the space requirements would be $O(u)$, which is the size of the universe.