Advanced Data Structures (COP5536)

Solution - Exam 3, Sample 3

1. (a) A splay tree

```
1
 \
  2
   \
    3
     \
      4
       \
        5
         \
          6
           \
            10
```

(b) split with respect to node with element 10

(Splay)

```
1                 1                   1                     10
 \                 \                   \                   /
  2                 2                   2                 2
   \                 \                   \               / \
    3                 3                   10            1   4
     \    ==>          \    ==>          /      ==>        / \
      4                 4               4                 3   6
       \                 \             / \                   /
        5                 10          3   6                 5
         \               /               /
          6             6               5
           \           /
            10        5
```

(Split)

```
  2                 10                 empty
 / \
   1    4
       / \
     3    6
         /
        5
```

2. (a)

```
           4
        /     \
      2         6
     / \       / \
    1   3     5   7
```

(b) delete 6

i) replace node 6 with 7 and delete leaf node 7

```
           4
         /    \
      2         7
     / \       / \
    1   3     5  ( )
```

ii) combine nodes 5,7, and ( )

```
           4
        /     \
      2        ( )
     / \        |
    1   3      5,7
```

iii) combine 2,4, and ( )

```
         2,4
      /   |   \
    1     3    5,7
```

3.

(a)

Node structure: (x_lower, x_upper) (x, y)

```
EnumerateRectangle(p,x_left, x_right, y_top)
{
    If(y > y_top) return null;

    If (x_lower<=x && x<=x_right) report(x,y);
        middleX=(x_lower+x_upper)/2;
        if (x_left<middleX)
            EnumerateRectangle( p->left, x_left, x_right, y_top);
        if(x_right>=middleX)
            EnumerateRectangle( p->right, x_left, x_right, y_top);

}
```

(b) Start from root, hight logK, when enumearte a node s++.

So total O(logK+S).

4.

(a) Assume k1->x, k2->y, k3->z. first sort the n records on x(k1). Then build a tree like this:

* Each node stores roughly n/2^(l-1) elements which is used to constructure two-dimensional range tree on x, y in that node.

* Z_i is the median value at that node which approximately devide the node's elements into two half parts.

* Each node in the tree represents a range in the2-dimensional.

(b) Start at the root, recursively visite node in such way compare the z-range of the query l_z and u_z to the range of the node.

* If the node's range is entirely within the query then search the 2-d range tree in that node using same way on y and at last search the sorted x for all points in the query x-range and return .

* If the query range is entirely below the median recursively, visite the left subtree.

* If it is above, recursively visite the right subtree.

* If the query range on z overlaps the median, visite both.

(c)

Processing Time:

At each level, there are total N elements and take NlogN processing time(two dimensional tree). Hight is logN, total N(logN)^2.

Query Time:

Search on Z at most 2logN, then search on y take 2logN, then on X take logN and report the result take F. Total O((logN)^3+F)

5.

If X is the NW or SW child of its parent, then its east-neighbor is NE or SE child of the parent, repectively. If X is the NE or SE child of its parent, we have to recursively find the east neighbor.

The algorithm is below:::

```
Procedure East_Neigbhor(X)
{
    if X is the root then retrun null;
    if X = NW-child of parent(X)
       return NE_child of parent(x);
    if X = SW-child of parent(X)
       return SE-child of parent(x);
    j <-- East_Neighbor(parent(X));  //recursive call
    if j = null or leaf
        return j;
    else if (X is NE-child of parent (X))
        return NW-child of j;
    else if (X is SE-child of parent(X))
        return SW-child of j;
}
```