(((exam #3 - make-up solution)))

1.
a)
After insert 75.

```
 a (  30     50     )
    /     |         \
b(10 15) c(35,40 45) d(55 60 70 75)
```

After insert 65

```
   (  30     50     65     )
    /     |       \       \
(10 15)(35 40 45) (55 60) e(70 75)
```

b)
We have the same assumption as in text that there is enough internal memory to hold all nodes accessed during an operation.
Insert 75: 3 disk accesses
    (2 to read a and d, and 1 to write updated node d)
Insert 65: 5 disk accesses
    (2 to read a and d, and 3 to write nodes d, e and a)

c)

delete 15.

```
       ( 35   50  )
        /   |    \
 (10 30) (40 45) (55 60 70)
```

delete 55.

```
       ( 35   50  )
        /   |    \
 (10 30) (40 45) (60 70)
```

delete 35.

```
       (  50  )
        /     \
(10 30 40 45) (60 70)
```

2. Splay tree

(a)

```
                  10
                  /
                 /
                9
               /
              /
             8
            /
           /
          4
         /  \
        /    \
       2      6
      / \    / \
     /   \  /   \
    1    3  5   7
```

(b) split with respect to 7

```
   (splay)
                        10               10                    7
                        /               /                     / \
                       /               /                     /   \
                      9               7                     6     10
        ===>         /    ===>       / \      ===>         /     /
                    /               /   \                 /     /
                   8               6     8               4     8
                  /               /       \             / \     \
                 /               /         \           /   \     \
                7               4           9         2     5     9
               /               / \                   / \
              /               /   \                 /   \
             6               2     5               1     3
            /               / \
           /               /   \
          4               1     3
         /  \
        /    \
       2      5
      / \
     /   \
    1    3
```

(Split)

```
               6            7             10
              /                          /
             /                          /
            4                          8
           / \                          \
          /   \                          \
         2     5                          9
        / \
       /   \
      1     3
```

3. Patricia

node: <label>:<bitnumber>:<key>.

After inserting 0100:

```
                        a:0:0100
                          /
                         a
```

After inserting 1011:

```
                        a:0:0100
                          /
                      b:1:1011
                       /    \
                      a      b
```

After inserting 1001:

```
                        a:0:0100
                          /
                      b:1:1011
                       /    \
                      a   c:3:1001
```

```
                            /    \
                           c      b

        After inserting 0111:

                        a:0:0100
                          /
                      b:1:1011
                      /        \
                d:3:0111     c:3:1001
                 /     \      /    \
                a       d    c      b

        After inserting 1010:

                        a:0:0100
                          /
                      b:1:1011
                      /        \
                d:3:0111     c:3:1001
                 /     \      /    \
                a       d    c    e:4:1010
                                   /     \
                                  e       b

        After inserting 0001:

                        a:0:0100
                          /
                      b:1:1011
                      /        \
                f:2:0001     c:3:1001
                 /     \      /    \
                f   d:3:0111  c    e:4:1010
                     /     \       /     \
                    a       d     e       b

part b:

        After deleting 0100

                        d:0:0111
                          /
                      b:1:1011
                      /         \
                f:2:0001     c:3:1001
                  /     \      /    \
                f       d    c    e:4:1010
                                    /     \
                                   e       b
```

3. Suffix tree

(a) S = aaab, Suffix(S) = { b, ab, aab, aaab }

```
                  B
               a/ \b
                /   \
               B     E
            a/ \b
             /   \
            B     E
         a/ \b
          /   \
        E     E

                    cf) E = end node,
                        B = branch node
```

(b)
Among the branch nodes that are parents of end nodes,
choose the one whose length of the labels on the edges leading to
that branch node is largest.
Then the sequence of labels on the path from root to the chosen one
is the longest repeating substring.

The longest repeating substring = aa

4

```
   (10,12) [0,24)                 [0,24) (10,12)               [0,24) (3,4)
                     I (8,16)           /        I(3,4)                /
                      ====> [0,12) (8,16)         ====>     [0,12) (10,12)
                                                                        \
                                                                  (8,16) [6,12)
```

```
               [0,24) (3,4)
                     /     \
I(23,6) [0,12) (10,12)      (23,6) [12,24)
====>              \
                  (8,16) [6,12)
```

```
                    [0,24) (3,4)
                         /       \
I(13,18)[0,12) (10,12)              (23,6) [12,24)
====>            \                    /
                (8,16) [6,12)     (13,18) [12,18)
```

```
                    [0,24) (3,4)
                         /         \
I(17,9) [0,12) (10,12)               (23,6) [12,24)
====>            \                    /
                (8,16) [6,12)     (17,9) [12,18)
                                   /
                                (13,18) [12,15)
```

5.

Assume we have N records with 2 keys, x and y.
A 2-D range tree is a radix tree using the range of x, while in each node we keep a sorted list based on y. The radix tree always splits the records in half, using a median x key value as the discriminator.

Preprocessing time P:
we need to build a sorted list on y, and another on x. Since we have the sorted list on x, we can find the discriminator easily. And since we have the sorted list on y, we can split the list in linear time on each level (there are logN levels overall). So the preprocessing time is:

P = NlogN + NlogN + N*logN = O(NlogN)

Space required S:

In each node we store an array of the records. At each level the number of records is N. So the space required is:

S = N*logN = O(NlogN)

Query time Q:

Upon each query, we start at the root, decide which branch to take by comparing the boundaries of x with the discriminator. Once we arrive a node, we use binary search on y in the array. Then we report those records that fit the query. So the query time is:

Q = logN*logN+F = O(sqr(logN)+F)

where F is the number of records reported.