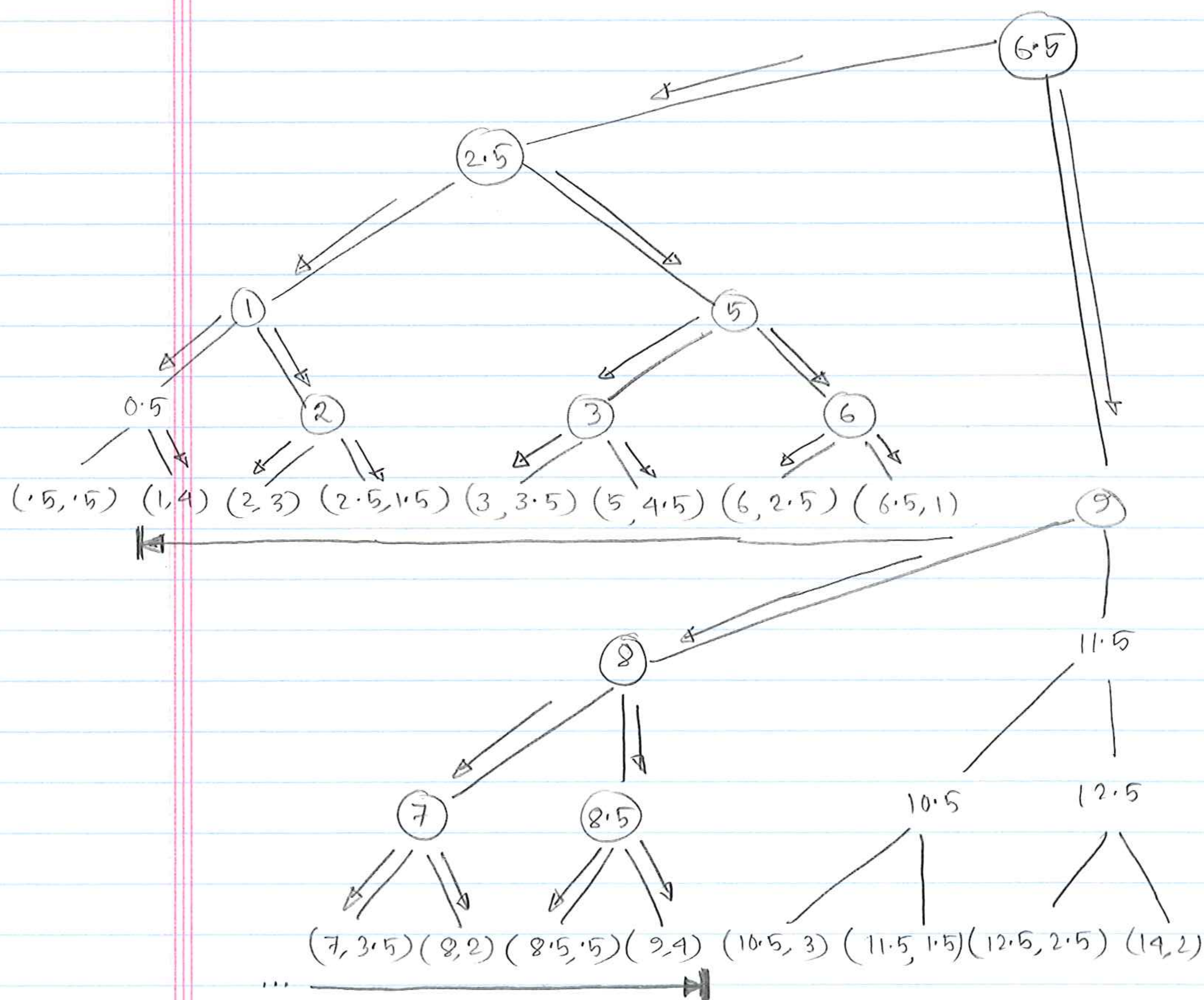
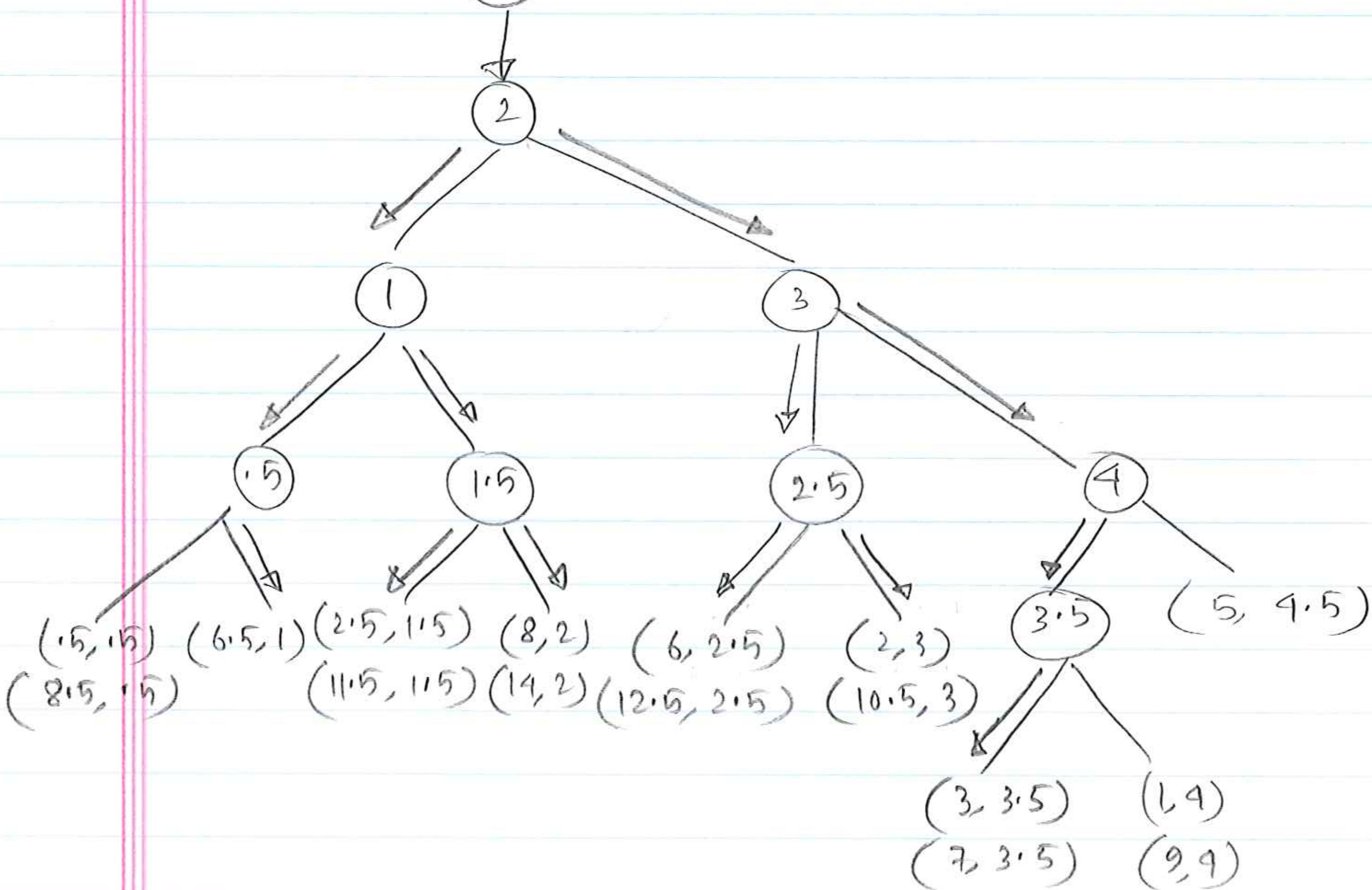
HW-6

1. a) primary range tree
(keyed by x-coordinate)
Arrow (→) and circle (○) indicate spliting and search query report

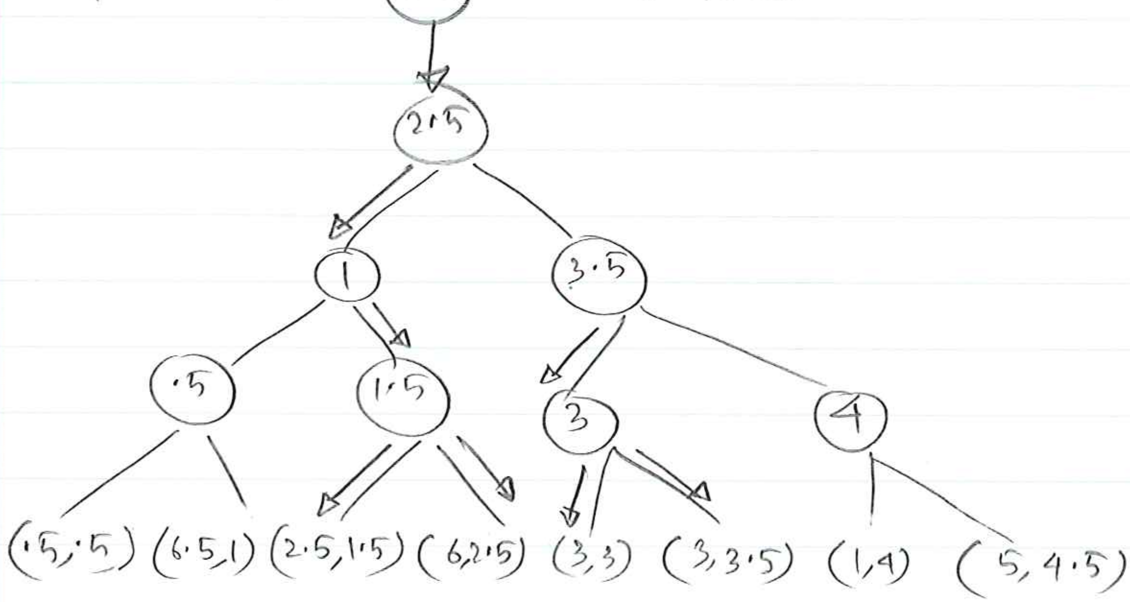


b) secondary tree
(keyed by Y-coordinate)

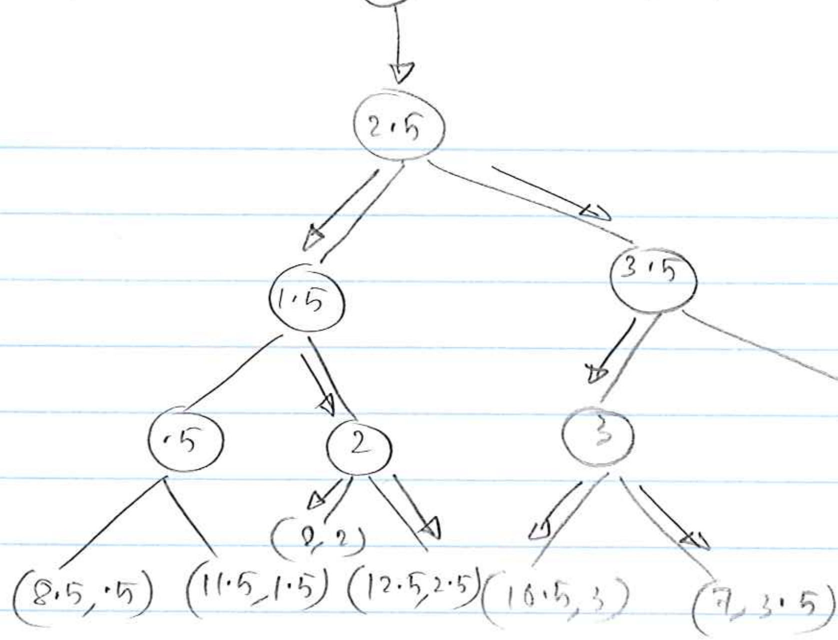
~ for node (6.5) in X-tree



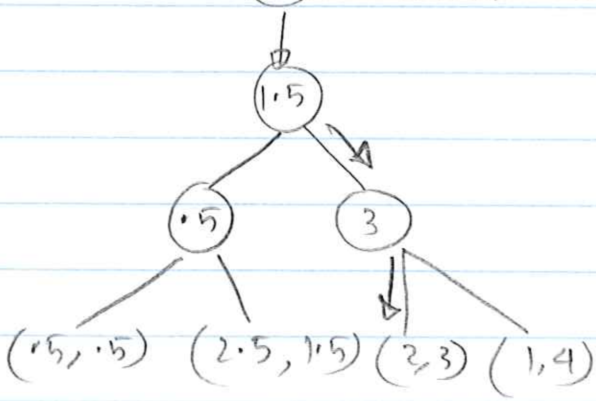
~ for node (2.5) in X-tree



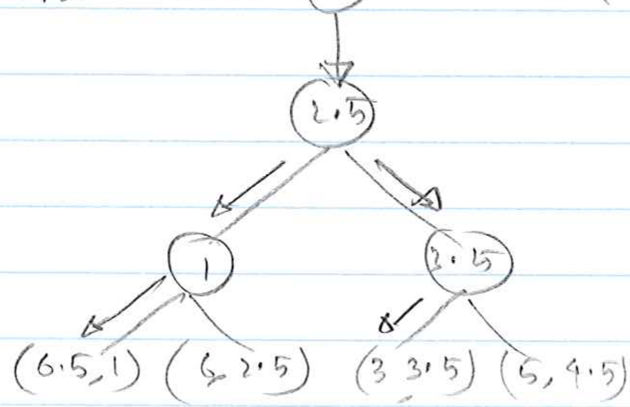
for node 9 in x-tree



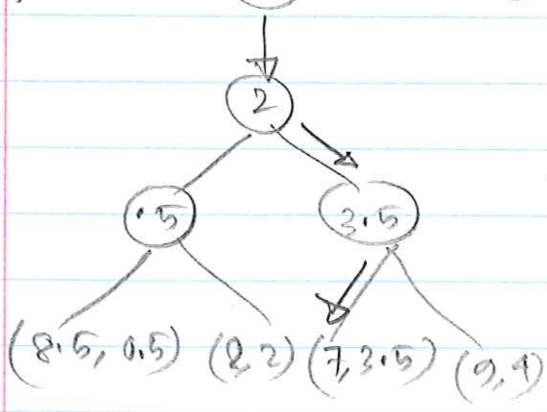
for node 1 in x-tree



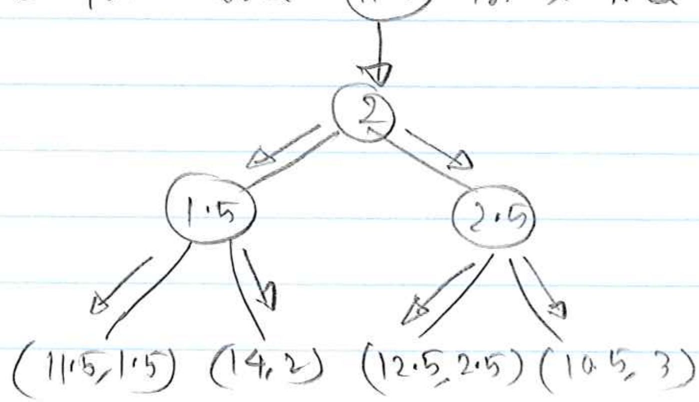
for node 5 in x-tree



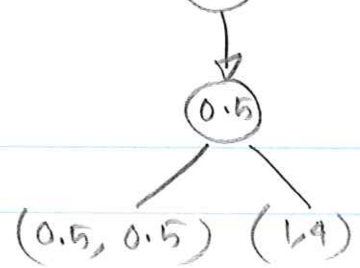
for node 8 in x-tree



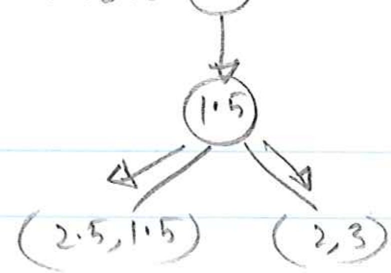
for node 11.5 in x-tree



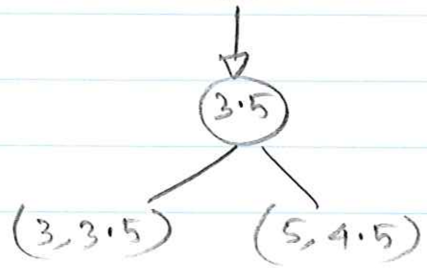
for node 0.5 in X-tree



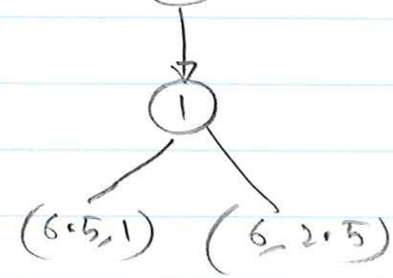
for node 2 in X-tree



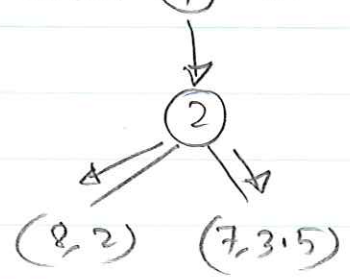
for node 3 in X-tree



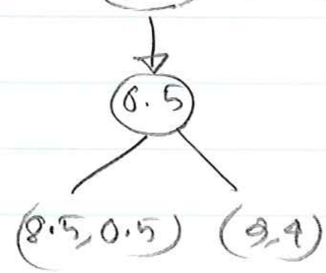
for node 6 in X-tree



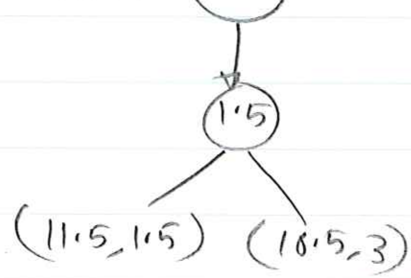
for node 7 in X-tree



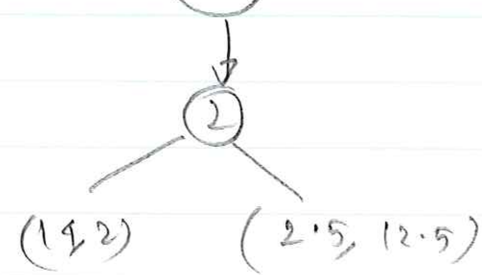
for node 8.5 in X-tree



for node 10.5 in X-tree



for node 12.5 in X-tree



c) split node: First node we found in our query range.

split node (primary range tree(X)) : 6.5
split node (secondary range tree(Y)) : 2

Search node (primary range tree (X)) : 0.5, 1, 2, 2.5, 3, 5, 6, 6.5, 7, 8, 8.5, 9
Search node (secondary range tree (Y)) : 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4

for the given query range final reporting points are (2, 3), (2.5, 1.5), (3, 3.5), (6, 2.5), (6.5, 1), (7, 3.5) (8, 2)

(2) We do not need to store all the sequences. Just need to store values currently needed for $c[i,j]$
For any time, to compute $c[i,j]$ we need:
~ earlier entries in the current row
i.e. $c[i,x]$ where $x \leq j-1$
~ earlier entries in the previous row
i.e, $c[i-1,x]$ where $x \geq j-1$

Finally x can be any values from 1 to $\min(m,n)$ with two $x = j-1$
so the total space cost $= \min(m,n) + O(1)$

In the new approach the array A will contain $\min(m, n) + 1$ of the following entries while computing $c(i, j)$.

$A[x] = c[i, x]$ for $1 \leq x < j-1$ (earlier entries of current row)

$A[x] = c[i-1, x]$ for $x \geqslant j-1$ (earlier entries of previous row)

$A[0] = c[i, j-1]$ (we have to put it in different place to avoid confliction with $c[i-1, j-1]$)

Now we will follow the following steps:

1. Initialize A to all 0
2. Compute the entries from left to right
3. While computing $c[i, j]$ for $j > 1$ required values are in
$$A[0] = c[i, j-1]$$
$$A[j-1] = c[i-1, j-1]$$
$$A[j] = c[i-1, j]$$

4. When $c[i, j]$ computation is done
   - move $A[0]$ to $A[j-1]$
   - put $c[i, j]$ in $A[0]$

Therefore in the mentioned setting the space required is $\min(m, n) + O(1)$

(3) a) For a set of n possible locations number of possible placement of toll booths is 2^n

For each possible placement running time for calculating sum of money for all the tolls and checking for regulation (can be done alongside) is $O(n)$

So running time for brute force algorithm is $O(n 2^n)$.

b) Recursive definition for $a[j]$:

Base case: $a[0] = 0$

Otherwise: $a[j] = \max\{a[\text{index of } l(j)] + t[j],\ a[j-1]\}$

c)
```
DP (a, T, L) {
    a[0] = 0;
    for i = 1 to n
        if (a[L[i]] + T[i] > a[i-1])
            a[i] = a[L[i]] + T[i];
        else
            a[i] = a[i-1];
    end for
}
```

Here, $L[i]$ = index of $l(i)$ which can be computed as the following procedure.

```
computeL (a, L) {
      L[0] = 0;
      for i = 1 to n
            for j = i-1 to 1
                  if( a[i] - a[j] >= 10 ) {
                        L[i] = j;
                        break;
                  }
            end for
      end for
}
```

d) The algorithm make one call of the procedure computeL() and another call to DP() where they take $O(n^2)$ and $O(n)$ time respectively. Hence overall running time for the algorithm is $O(n^2)$