1. (4 pt) Suppose we have B-tree nodes with room for three keys and four pointers, as in the examples of this section. Suppose also that when we split a leaf, we divide the pointers 2 and 2, while when we split an interior node, the first 3 pointers go with the first (left) node, and the last 2 pointers go with the second (right) node. We start with a leaf containing pointers to records with keys 1, 2, and 3. Then add in order, records with keys 4, 5, 6, and so on. At the insertion of what key will the B-tree first reach four levels?
Draw the final tree.

Answer. Since we are always adding at the right extreme of the B-tree, once split, the first (left) of the pair never changes. Thus, the leaf blocks will each have two keys only: (1,2), (3,4), (5,6), and so on.
At the second level, we divide pointers 3 to the left, and 2 to the right, so each except the rightmost node at that level has three pointers; the leftmost has two. The same holds for any other level.

Thus, when we first get four levels, the root has two pointers. At the third level, there are blocks with 3 and 2 pointers, respectively. At the second level, there are five blocks, with 3, 3, 3, 3, and 2 pointers. These 14 pointers go to leaf blocks with two pointers each, for a total of 28 pointers to records.

Thus, when the record with key 28 is inserted, the 4th level is created.

2. (3 pt)
Redo the example in slides titled "Attempt at using B-trees for MD-queries" under the assumption that the range query asks for a square in the middle that is $n \, x \, n$ for some *n* between 1 and 1000. How many disk I/O's are needed? For which values of *n* do indexes help?

Answer.

We obtain 1,000,000/(1000/n) pointers to follow from each index.
Doing pointer intersection, we obtain 1,000,000/((1000/n)*(1000/n)) pointers to finally follow, i.e. we need that many I/Os once we obtain the pointers.

Now, what is the number of I/Os for obtaining the pointers from each index?
We need to traverse 2*(1,000,000/(1000/n)) /200 leaves (blocks) +2 (for intermediate B-tree level).

Total: 1,000,000/((1000/n)*(1000/n)) + 2*(1,000,000/(1000/n))/200 +2

For this number to be less or equal to the number of blocks of the relation, 10,000, we need to solve:

1,000,000/((1000/n)*(1000/n)) + 2*(1,000,000/(1000/n))/200 +2 = 10,000

n^2 + 10n + 2 = 10,000
n= 95,   i.e. n<=95 in order to be beneficial to use the two B-Tree indexes.

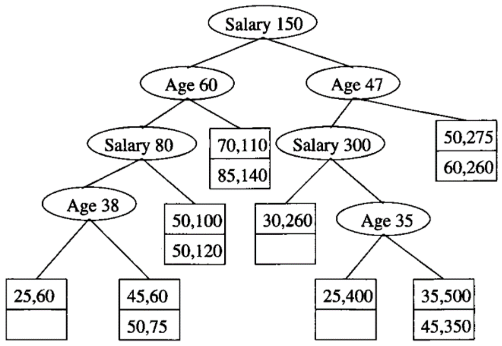
3. (4 pt)
Suppose we store a relation R(x,y) in a grid file. Both attributes have a range of values from 0 to 1000. The partitions of this grid file happen to be uniformly spaced; for x there are partitions every 20 units, at 20, 40, 60, and so on, while for y the partitions are every 50 units, at 50, 100, 150, and so on.
a) How many buckets do we have to examine to answer the range query
SELECT * FROM R
WHERE 310 < x AND x < 400 AND 520 < y AND y < 730;

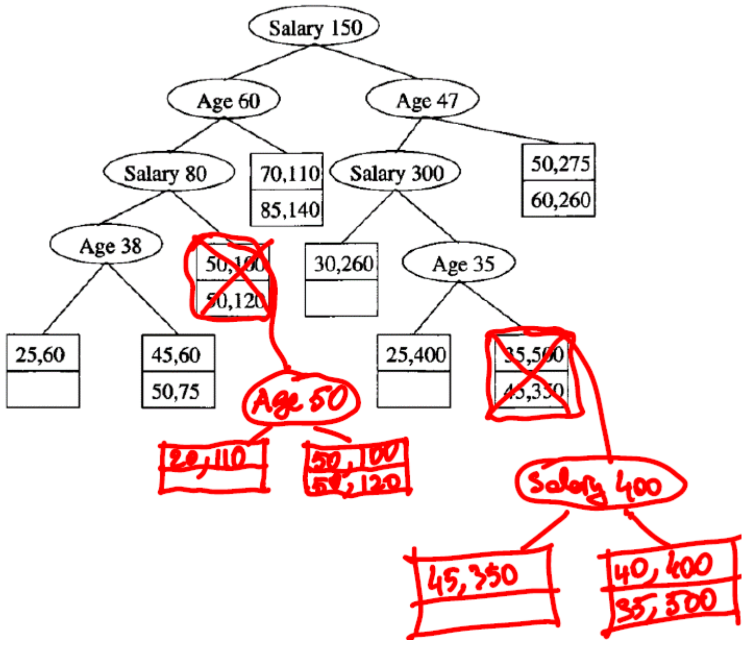
Answer. The range query takes in parts of 5 stripes in each of the dimensions, so the total number of buckets to be searched is 5*5 = 25.

b) We wish to perform a nearest-neighbor query for the point (110,205).
We begin by searching the bucket with lower-left corner at (100,200) and upper-right corner at (120,250), and we find that the closest point in this bucket is (115,220). What other buckets must be searched to verify that this point is the closest?

Answer. The distance to the closest point found so far is a little more than 15. Points closer than that to the target point (110,205) can be found in **five** neighboring rectangles, those with lower-left corners at (80,200), (80,150), (100,150), (120,150), and (120,200).

4. (2 pt) Show a possible evolution of the tree of in figure if we insert the points (20,110) and then (40,400).





5. (4 pt) Build an R-tree index with M=3 using the following sequence of rectangles:
(1,1,3,3) (2,2,4,4) (1,4,3,6) (5,4,7,6) (6,3,8,5) (7,2,9,4) (8,5,10,7) (8,6,9,7)
Redraw the tree each time an insertion is done.

After inserting (1,1,3,3)

```
(1 1 3 3)[0]
```

After inserting (2,2,4,4)

```
(1 1 3 3)[0]  (2 2 4 4)[1]
```

After inserting (1,4,3,6)

```
(1 1 3 3)[0]  (2 2 4 4)[1]  (1 4 3 6)[2]
```

After inserting (5,4,7,6)

```
(1 1 4 6)[0]  (5 4 7 6)[1]
[0]  (1 1 3 3)[0]    (2 2 4 4)[1]    (1 4 3 6)[2]
[1]  (5 4 7 6)[0]
```

After inserting (6,3,8,5)

```
(1 1 4 6)[0]  (5 3 8 6)[1]
[0]  (1 1 3 3)[0]    (2 2 4 4)[1]    (1 4 3 6)[2]
[1]  (5 4 7 6)[0]    (6 3 8 5)[1]
```

After inserting (7,2,9,4)

```
(1 1 4 6)[0]  (5 2 9 6)[1]
[0]  (1 1 3 3)[0]    (2 2 4 4)[1]    (1 4 3 6)[2]
[1]  (5 4 7 6)[0]    (6 3 8 5)[1]    (7 2 9 4)[2]
```

After inserting (8,5,10,7)

```
(1 1 4 6)[0]  (5 2 9 6)[1]  (8 5 10 7)[2]
[0]  (1 1 3 3)[0]    (2 2 4 4)[1]    (1 4 3 6)[2]
[1]  (5 4 7 6)[0]    (7 2 9 4)[1]    (6 3 8 5)[2]
[2]  (8 5 10 7)[0]
```

After inserting (8,6,9,7)

```
(1 1 4 6)[0]  (5 2 9 6)[1]  (8 5 10 7)[2]
[0]  (1 1 3 3)[0]    (2 2 4 4)[1]    (1 4 3 6)[2]
[1]  (5 4 7 6)[0]    (7 2 9 4)[1]    (6 3 8 5)[2]
[2]  (8 5 10 7)[0]   (8,6,9,7)[1]
```

6. (6 pt) Consider the following fragment from a collection of documents. Assume that these three documents are the only documents in the collection that contain the words “dog” or “cat”.

Document id	**Document Text**
234569	The phrase "fight like cats and dogs" reflects a natural tendency for the relationship between the two species to be antagonistic. However, sometimes the two species can be friends.
234578	Dogs and cats can have a bad relationship.
234839	Cats are furry.
234879	Dogs are man's best friend.

1. Write the inverted index posting lists for terms “cat” and “dog”.

   cat : 234569:1, 234578:1, 234839:1
   dog: 234569:1, 234578:1, 234879:1

2. Write the compressed form of the posting list for dog.

   dog: 234569:1, 9:1, 301:1

   In binary:
   234,569 = b111001010001001001
   1 = b1
   9 = b1001
   301 = b100101101

   VB encoding:
   234,569: 00001110 00101000 11001001
   1: 10000001
   9: 10001001
   1: 10000001
   301: 00000010 10101101
   1: 10000001

3. Calculate the cosine similarity of each of the above documents with query
   q: cat and dogs.
   Use stemming and remove stop words from the documents and query.
   Use only TF (ignore IDF).

   We compute the document vector magnitudes as follows.

Document id	**Document Text**	Maxf	Document vector magnitudes
234569	The phrase "fight like cats and dogs" reflects a natural tendency for the relationship between the two species to be antagonistic. However, sometimes the two species can be friends.	2 (species, two)	Sqrt(phrase 1+ fight 1+ cat 1+ dog 1+ reflects 1+ natural 1+ tendency 1+ relationship 1+ two 2+ species 2+ antagonistic 1+ friend 1) = Sqrt((1/2)^2+ (1/2)^2+ (1/2)^2+ (1/2)^2+ (1/2)^2+ (1/2)^2+ (1/2)^2+ (1/2)^2+ (2/2)^2+ (2/2)^2+ (1/2)^2+ (1/2)^2) = 2.12
234578	Dogs and cats can have a bad relationship.	1	Sqrt(4) = 2
234839	Cats are furry.	1	Sqrt(2) = 1.41
234879	Dogs are man's best friend.	1	Sqrt(4) = 2

Query magnitude: Sqrt(2) = 1.41
Cos(q, d_{234569}) = (.5+.5)/(2.12*1.41) = 0.33
Cos(q, d_{234578}) = (1+1)/(2*1.41) = 0.71
Cos(q, d_{234839}) = 1/(1.41*1.41) = 0.5
Cos(q, d_{234879}) = 1/(2*1.41) = 0.35