COP 3503 Recitation Worksheet: Divide and Conquer Algorithms - Solutions

1) Skyline Merge Algorithm

Write a method that takes in two skylines (as two odd-sized integer arrays), and returns an integer array representing the merged skyline. To make your code a little easier to write, you may assume that ALL of the x coordinates in the two skyline are distinct; that only one building will start or end at any given x coordinate. Also, you may return a skyline that has the same height at consecutive segments. Here is the function prototype:

```
public static int[] mergeSky(int[] skyA, int[] skyB);
```

For example, if skyA = [2, 10, 5, 20, 8, 6, 20] and
skyB = [3, 8, 10, 3, 25, 50, 30, 15, 40]

then the method should return

[2, 10, 3, 10, 5, 20, 8, 8, 10, 6, 20, 3, 25, 50, 30, 15, 40]

For clarity, the heights are highlighted in yellow.

```
public static int[] mergeSky(int[] skyA, int[] skyB) {

      int[] res = new int[skyA.length+skyB.length+1];

      int i = 0, j = 0, k = 0;
      int curA = 0, curB = 0;

      while (i<skyA.length || j<skyB.length) {

            if (j>=skyB.length || (i<skyA.length && skyA[i]<skyB[j])){
                  res[k++] = skyA[i++];
                  curA =  i<skyA.length ? skyA[i++] : 0;
            }

            else {
                  res[k++] = skyB[j++];
                  curB = j<skyB.length ? skyB[j++] : 0;
            }
            if (k<res.length) res[k++] = Math.max(curA, curB);
      }
      return res;
}
```

2) Karatsuba's Integer Multiplication Algorithm

Illustrate Karatsuba's Algorithm to multiply the following integers:

I = 3298
J = 4167

Clearly show the 3 recursive multiplications that will occur. (Don't break these down further recursively, just show their results.)

Then, show HOW to combine those results appropriately to create the correct product.

$I_h = 32$, $I_l = 98$
$J_h = 41$, $J_l = 67$

$I_hJ_h = 32 \times 41 = 1312$
$I_lJ_l = 98 \times 67 = 6566$
$(I_h + I_l)(J_h + J_l) = (32 + 98)(41 + 67) = 130 \times 108 = 14040$
So, middle term = $14040 - 1312 - 6566 = 6162$

Final answer = $1312 \times 10^4 + 6162 \times 10^2 + 6566 = 13120000$

```
      616200
+       6566
-------------
  13742766
```

Indeed, it turns out that $3298 \times 4167 = 13742766$.

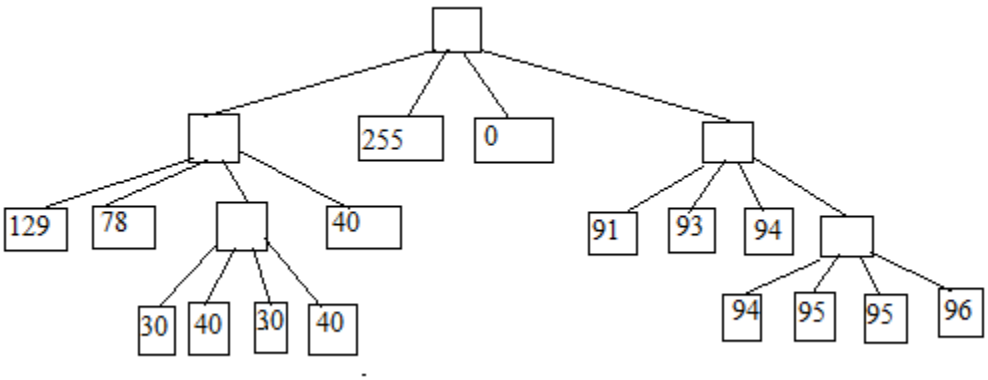
3) Divide and Conquer Code – QuadTrees

A quadtree is a data structure to store an image. For simplicity, we assume the dimensions of the image to be 2^k x 2^k, for some non-negative integer k. If each pixel in the whole image has the same value, we store this entire picture as a single node with that value. On the other hand, if this image has at least two distinct values in its pixels, we store no color value in the root node of the tree, but instead store four pointers to the upper left, upper right, lower left and lower right quadrants of the image. These pointers are simply pointing to quadtrees that represent that portion of the image. This alternate storage scheme for a picture can be seen well visually. Imagine we have the following 8 x 8 grayscale image:

129	129	78	78	255	255	255	255
129	129	78	78	255	255	255	255
30	40	40	40	255	255	255	255
40	30	40	40	255	255	255	255
0	0	0	0	91	91	93	93
0	0	0	0	91	91	93	93
0	0	0	0	94	94	94	95
0	0	0	0	94	94	95	96

Note: A grayscale pixel stores an intensity of shade as in integer, ranging from 0 (white) to 255 (black).

We would store this image as a quadtree as follows:



One useful computation for a grayscale image is the average intensity of its pixels. (This might correspond to how much ink would be used to print it, for example.) Complete the method intensity in the quadtree class on the next page so that it returns a double, corresponding to the average value in the pixels of the designated quadtree image.

```
class quadtree {

    final public static int INTERNAL = -1;
    private int shade;
    private quadtree[] children;

    // Assume pic = 2^k by 2^k
    public quadtree(int[][] pic, int x, int y, int n) {

        if (same(pic, x, y, n)) {
            shade = pic[x][y];
            children = null;
        }

        else {
            shade = INTERNAL;
            children = new quadtree[4];
            children[0] = new quadtree(pic,x,y,n/2);
            children[1] = new quadtree(pic,x,y+n/2,n/2);
            children[2] = new quadtree(pic,x+n/2,y,n/2);
            children[3] = new quadtree(pic,x+n/2,y+n/2,n/2);
        }
    }

    // Requires that x >= 0, y >= 0, x+n <= pic.length, y+n <= pic.length
    public static boolean same(int[][] pic, int x, int y, int n) {
        int val = pic[x][y];
        for (int i=x; i<x+n; i++)
            for (int j=y; j<y+n; j++)
                if (pic[i][j] != val)
                    return false;
        return true;
    }

    public double intensity() {

        if ( shade != INTERNAL )
            return shade;

        double ans = 0;
        for (int i=0; i<4; i++)

            ans += children[i].intensity();


        return ans/children.length;



    }
}
```

4) Consider running the closest pair of points algorithm on the 16 points shown below. Rather than run the recursion on each group of 8 points, just calculate the answer for both sets of 8 points via brute force. Then, illustrate which points would lie within the "middle strip" where we look for an answer that's better between the two sets of 8 points.

Here are the points, conveniently sorted by x-coordinate:

(3, 7), (4, 1), (6, 12), (7, 7), (10, 11), (10, 18), (13, 2), (15, 9),

(16, 8), (20, 17), (22, 12), (23, 1), (27, 11), (30, 5), (33, 20), (34, 2)

The left half of the data is the top set of points and the right half is the bottom set of points. By inspection, we see that (3, 7) and (7, 7) are 4 apart. Now, we can just brute force check only points where the x coordinates differ by less than 4. None have a small enough difference in y to beat 4.

The right half of the data seems more spaced out. In fact, (30, 5) and (34, 2) are 5 apart (3-4-5 triangle), and nothing seems better than this checking the x values that are less than 5 apart.

So answer from left is 4, and right is 5. This makes delta 4.

Our dividing line is x = 15.5. So our strip in the middle goes from x = 11.5 to x = 19.5. The points to include in the strip **are (13, 2), (15, 9), and (16, 8).** It's these latter two, with distance $\sqrt{2}$ between them that is the closest pair of points in this data.