```
    1. PROGRAM OUTPUT
    − Tests are on the square root of 1000 random numbers 0..1 and the square of 100
    0 random numbers 0..1.
    * Results:

5       square,      >,---       *         |------------       ,< ,0.01 ,0.10 ,0.26
     ,0.51 ,0.84, |, 0.26, 0.52 ,1.00
        squareRoot,  >,          --------|      *     ----    ,< ,0.30 ,0.53 ,0.69
     ,0.83 ,0.95, |, 0.69, 0.36 ,1.00

    − read test data for the sample files sk[abcde] and run Scott−Knott on that samp
    le data.
    * Results:

10
    ----| data/ska.txt |----------------------
    one :mu 0.5 :rank 1
    two :mu 0.8 :rank 2

15  ----| data/skb.txt |----------------------
    rx4 :mu 0.3125 :rank 1
    rx1 :mu 0.535 :rank 2
    rx3 :mu 0.75 :rank 3
    rx2 :mu 0.8 :rank 3
20
    ----| data/skp.txt |----------------------
    rx1 :mu 0.2 :rank 1
    rx2 :mu 0.2 :rank 1
    rx3 :mu 0.75 :rank 2
25
    ----| data/skc.txt |----------------------
    rx1 :mu 100.0 :rank 1
    rx3 :mu 100.0 :rank 1
    rx5 :mu 100.0 :rank 1
30  rx2 :mu 100.2 :rank 1
    rx4 :mu 100.2 :rank 1
    rx6 :mu 100.2 :rank 1

    ----| data/skd.txt |----------------------
35  rx1 :mu 100.0 :rank 1
    rx2 :mu 120.2 :rank 1
    rx3 :mu 140.0 :rank 2
    rx6 :mu 402.2 :rank 3
    rx4 :mu 420.2 :rank 3
40  rx5 :mu 440.0 :rank 3

    ----| data/ske.txt |----------------------
    rx1 :mu 1.0 :rank 1
    rx2 :mu 2.0 :rank 1
45  rx3 :mu 3.0 :rank 1
    rx4 :mu 1000.0 :rank 2

    2. SOURCE CODES
    ************************************************************
50  <ptile.py>
    def ptile(lst, chops, width, form, lo, hi):
        lo = 0 if lo == '' else lo
        hi = 100 if hi == '' else hi
        form = '%3.0f' if form == '' else form
55      width = 50 if width == '' else width
        bar = '|'
        out = []
        for i in range(width):
            out += [' ']
60      nlst = sorted(lst) if type(lst) == list else sorted(lst.values())
        n = len(lst)
        who = {}
        where0 = {}
        for p in chops.keys():
65          who[p] = float(nlst[int(float(p)*n)])
            where = int(width*(float(who[p]) − lo)/(hi − lo))
            tmp = {}
            tmp['x'] = where
            tmp['*'] = chops[p]
```



```
70          where0[p] = tmp
        w = len(where0.keys())
        wheres = {}
        for i in range(w): wheres[i] = where0[sorted(where0.keys())[i]]
        for i in range(w):
75          start = wheres[i]['x']
            stop = width if i == w−1 else wheres[i+1]['x']
            for j in range(start, stop):out[j] = wheres[i]['*']
        out[int(width/2)] = bar
        median = float(nlst[int(0.5*n)])
80      spread = float(nlst[int(0.75*n)]) − float(nlst[int(0.25*n)])
        maxv = float(nlst[n−1])
        where = int(width*(median − lo)/(hi − lo))
        out[where] = '*'
        out = ' >,'+''.join(out)[:]+',< ,'+' ,'.join([form%who[s] for s in sorted(wh
    o.keys())])\
85      +', |, '+form%(float(median)) +', '+form%(spread)+' ,'+ form%(maxv)
        return out
    ************************************************************
    <rank.py>
    import math
90  import lib
    class Num:
        def __init__(i):
            i.mu = {}
            i.sum = {}
95          i.m2 = {}
            i.var = {}
            i.n = {}
            i.x = {}
            i.label = {}
100         i.name = []

    class Div:
        def __init__(i):
            i.total = []
105         i.cohen = []
            i.mittas = []
            i.a12 = []
            i.order = {}
            i.level = 0
110
    def ranks(filename, a):
        print "\n----|", filename,"|----------------------"
        f = open(filename)
        _Nums = Num()
115     _Div = Div()
        obs(f,0,_Nums,_Div)
        rank(0,_Nums,_Div,a)
        maxv = len(_Div.order.keys())
        for i in range(maxv):
120         i = i+ 1
            k = _Div.order[i]['=']
            print k, ':mu', _Nums.mu[k], ':rank', _Nums.label[k]

    def obs(f, all, _Nums, _Div):
125     now = all
        line = f.readline()
        while line:
            line = line.split()
            for i in line:
130             if i[0].isdigit():
                    v = float(i)
                    inc(v, now, _Nums)
                    inc(v, all, _Nums)
                else: now = i
135         line = f.readline()
        f.close()

        for i in _Nums.name:
            if i != all:
140             temp={}
                temp['='] = i
```

```
                temp['x'] = _Nums.mu[i]
                _Div.order[i] = temp
        s = 0
        norder = {}
        while s < len(_Nums.name)-1:
            tmp = 10**17
            ind = 0
            s = s+1
            norder[s] = {}
            for i in _Div.order.keys():
                if tmp > _Div.order[i]['x']:
                    tmp = _Div.order[i]['x']
                    ind = i
            norder[s]['='] = _Div.order[ind]['=']
            norder[s]['x'] = _Div.order[ind]['x']
            del _Div.order[ind]
        _Div.order = norder
    def inc(v, k, nums):
        nums.label[k] = 0
        if k not in nums.name:
            nums.name += [k]
            nums.n[k] = 0
            all = nums.n[k] = nums.n[k] + 1
            nums.x[k] = []
            nums.sum[k] = v
            delta = float(v)
            nums.mu[k] = float(delta/all)
            nums.m2[k] = 0
            nums.var[k] = 0
        else:
            all = nums.n[k] = nums.n[k] + 1
            nums.sum[k] = nums.sum[k] + v
            delta = v - nums.mu[k]
            nums.mu[k] = nums.mu[k] + delta/all
            nums.m2[k] = nums.m2[k] + float(delta*(v-nums.mu[k]))
            nums.var[k] = float(nums.m2[k])/float(all - 1 + lib.PINCH)
        nums.x[k] += [v]

    def rank(all,nums,div,a):
         div.cohen  = float(a["-cohen"])*math.sqrt(nums.var[all])
         div.mittas = a["--mittas"]
         div.a12    = a["-a12"]
         div.level  = 0
         div.total  = nums.n[all]
         rdiv(1,len(div.order.keys()),1,nums, div)


    def rdiv(lo, hi, c, nums, div):
        cut = divm(lo, hi, nums, div)
        if cut:
            div.level = div.level + 1
            c = rdiv(lo, cut-1, c, nums, div) + 1
            c = rdiv(cut, hi, c, nums, div)
        else:
            for i in range(lo, hi+1): nums.label[div.order[i]['=']] = c
        return c

    def divInits(lo,hi,nums,div,num0,num1):
        b= div.order[lo]["="];
        num0.n[lo]= nums.n[b];
        num0.sum[lo]= nums.sum[b]
        b= div.order[hi]["="];
        num1.n[hi]= nums.n[b];
        num1.sum[hi]= nums.sum[b]
        for i in range(hi-1, lo-1, -1):
            b = div.order[i]['=']
            num1.n[i] = num1.n[i+1] + nums.n[b]
            num1.sum[i] = num1.sum[i+1] + nums.sum[b]
        return num1.sum[lo]/num1.n[lo]

    def divm(lo, hi, nums, div):
       num0 = Num()
       num1 = Num()
```



```
        muAll = divInits(lo,hi,nums, div, num0, num1)
        maxv   = -1
        cut = None
        for i in range(lo+1, hi+1):
            b = div.order[i]['=']
            num0.n[i] = num0.n[i-1] + nums.n[b]
            num0.sum[i] = num0.sum[i-1] + nums.sum[b]
            left = num0.n[i]
            muLeft = num0.sum[i]/left
            right = num1.n[i]
            muRight = num1.sum[i]/right
            e = errDiff(muAll, left, muLeft, right, muRight)
            if div.cohen:
                if abs(muLeft - muRight) <= float(div.cohen): continue
            if div.mittas:
                if e < maxv:continue
            if div.a12:
                if bigger(lo, i, hi, nums, div) < float(div.a12):continue
            maxv = e
            cut = i
        return cut

    def errDiff(mu, n0, mu0, n1, mu1):
        return n0*(mu - mu0)**2.0 + n1*(mu - mu1)**2.0

    def bigger(lo, mid, hi, nums, div):
        below = values(lo, mid-1, nums, div)
        above = values(mid, hi, nums, div)
        return a12statistic(below, above)

    def values(i, j, nums, div):
        out = []
        for k in range(i, j+1):
            b = div.order[k]['=']
            out += nums.x[b]
        return out

    def a12statistic(below, above):
        comparisons = more = same = 0
        for j in range(len(above)):
            for i in range(len(below)):
                comparisons = comparisons + 1
                more = more + 1 if above[j] > below[i] else more
                same = same + 1 if above[j] == below[i] else more
        return (more + 0.5*same)/comparisons
    ************************************************************
    <main.py>
    import rank
    import lib
    if __name__ == "__main__":
        a = lib.pairs("-cohen,0.3,--mittas,1,-a12,0.6")
        rank.ranks('data/ska.txt', a)
        rank.ranks('data/skb.txt', a)
        rank.ranks('data/skp.txt', a)
        rank.ranks('data/skc.txt', a)
        rank.ranks('data/skd.txt', a)
        rank.ranks('data/ske.txt', a)
    """
    import lib
    import ptile
    import random

    if __name__ == "__main__":
        lst = []
        lst4 = []
        f = '%3.2f'
        for i in range(1000): lst += [random.random()**2]
        for i in range(1000): lst4 += [random.random()**0.5]
        chops1 = lib.pairs('0.1,-,0.3, ,0.5, ,0.7,-,0.9, ')
        chops2 = lib.pairs('0.25,-,0.5,-,0.75, ')
        out1 = ptile.ptile(lst, chops1, 40, f, 0, 1)
        out4 = ptile.ptile(lst4, chops1, 40, f, 0, 1)
        print "square,    ", out1
```

```
        print "squareRoot,", out4
        lst2 = lib.pairs('1,0.51,2,0.49,3,0.48,4,0.52,5,0.25,6,0.48,7,0.49,8,0.51,9,
    0.52,10,0.48')
290     lst3 = lib.pairs('1,0.81,2,0.82,3,0.80,4,0.79,5,0.78,6,0.8,7,0.81,8,0.82,9,0
    .79,10,0.78')
        out2 = ptile.ptile(lst2, chops2, 40, f, 0.45,0.85)
        out3 = ptile.ptile(lst3, chops2, 40, f, 0.45,0.85)
        print "rx4, ", out2
        print "rx5, ", out3
295 """
```