function Sorting4()

clc;

% Generates a matrix of numbered students with their house aptitudes

% House positioning is officially [G S R H]

total = 23; % n is the number of incoming students

remain = mod(total,4);

n = total-remain;

SA = zeros(total,6);

% House Matrices

% Row size to accommodate irregular data; ideally n/4

GR = [];

SL = [];

RA = [];

HU = [];

% House Counts to expand house matrices for any number of fitting students

g\_c = 1;

s\_c = 1;

r\_c = 1;

h\_c = 1;

for i = 1:1:total

% Generates a 4 column row vector filled with a random number

% between 1-10 for house aptitude

RAN = rand(1,4).\*10;

% Stores the newest generated student aptitude vector in a "student list"

SA(i,1) = i;

for run = 1:4

SA(i,run+1) = RAN(run);

end

end

% Matches a student's aptitude scores with their student number

Student\_List = SA;

% Student\_List = [...

% 1.0000 9 10 8 7 71.0000;...

% 2.0000 10 8 7 9 83.0000;...

% 3.0000 1 2 3 4 82.0000;...

% 4.0000 4 3 1 7 82.0000;...

% 5.0000 1 5 6 2 83.0000;...

% 6.0000 3 5 7 8 71.0000;...

% 7.0000 5 4 9 1 82.0000;...

% 8.0000 4 5 7 1 82.0000;...

% 9.0000 8 1 4 2 71.0000;...

% 10.0000 3 6 1 2 83.0000;...

% 11.0000 1 2 3 6 72.0000;...

% 12.0000 8 2 1 7 72.0000;...

% 13.0000 9 8 6 3 83.0000;...

% 14.0000 8 7 4 10 71.0000;...

% 15.0000 8 4 3 5 71.0000;...

% 16.0000 9 7 3 1 71.0000;...

% 17.0000 9 5 2 3 71.0000;...

% 18.0000 10 1 9 4 72.0000;...

% 19.0000 10 9 6 5 83.0000;...

% 20.0000 10 2 4 1 82.0000;...

% 21.0000 2 5 8 3 82.0000;...

% 22.0000 7 2 8 9 72.0000;...

% 23.0000 4 6 1 2 83.0000;...

% ]; %For Testing

% Maximum Possible House Happiness

% This is no longer as scientific because there is not a definite number of

% students in each house. House size will vary by one.

max\_hap=[];

for y = 1:4

temp\_house = sortrows(Student\_List,1+y); % sorts matrix by highest for

% house

numerator = 0;

for w = ((total-round(total/4)+1):total

numerator = numerator+temp\_house(w,y+1);

end

val = numerator/round(total/4); % average of 5 highest ranked students

max\_hap(y) = val;

end

**%% Running Stuff**

x=6; % stupid variable to keep the while loop

% Student\_List

[GR,SL,RA,HU,Student\_List] = file(Student\_List);

while x<=8

[rG,cG] = size(GR);

[rS,cS] = size(SL);

[rR,cS] = size(RA);

[rH,cH] = size(HU);

if rH<=((n/4)+1)&& rG<=((n/4)+1)&& rS<=((n/4)+1)&& rR<=((n/4)+1)

% Final Answer: House Happiness

[GR;0,0,0,0,0,0;SL;0,0,0,0,0,0;RA;0,0,0,0,0,0;HU]

GR\_happy = sum(GR(:,2))/rG;

SL\_happy = sum(SL(:,3))/rS;

RA\_happy = sum(RA(:,4))/rR;

HU\_happy = sum(HU(:,5))/rH;

Happy = [max\_hap;GR\_happy,SL\_happy,RA\_happy,HU\_happy]

break

else

[GRmid,SLmid,RAmid,HUmid,rejects] = remove2(GR,SL,RA,HU)

[GR,SL,RA,HU] = REsort(GRmid,SLmid,RAmid,HUmid,rejects)

end

end

**%% Functions**

% Initial Sort

function [GR,SL,RA,HU,MAT] = file(MAT)

% Takes the maximum house value for each student

for m = 1:1:total

[smax(m),pos(m)] = max(MAT(m,2:5));

end

for h = 1:total

if pos(h) == 1

MAT(h,6) = 'G';

GR(g\_c,:) = MAT(h,:);

g\_c = g\_c + 1;

elseif pos(h) == 2

MAT(h,6) = 'S';

SL(s\_c,:) = MAT(h,:);

s\_c = s\_c + 1;

elseif pos(h) == 3

MAT(h,6) = 'R';

RA(r\_c,:) = MAT(h,:);

r\_c = r\_c + 1;

elseif pos(h) == 4

MAT(h,6) = 'H';

HU(h\_c,:) = MAT(h,:);

h\_c = h\_c + 1;

end

end

end

% New Removal Function

function [GR,SL,RA,HU,rejects] = remove2(GR,SL,RA,HU)

rejects = [];

limit = (n/4)+2;

reject\_pos = 1;

for q = 1:total

% Recalculating because otherwise this thing will error

[rg,cg] = size(GR);

[rs,cs] = size(SL);

[rr,cr] = size(RA);

[rh,ch] = size(HU);

if rg >= limit

for g = 1:rg

[max1(g),pos1(g)] = max(GR(g,2:5));

tempGR = GR; % So we can ask the secondary maximum

tempGR(g,2) = 0; % Negates the actual maximum

[max2(g),pos2(g)] = max(tempGR(g,2:5)); % Secondary Max

dif(g) = max1(g)-max2(g);

snum(g) = GR(g,1);

end

Gdif = [transpose(snum),transpose(max1),transpose(pos1),...

transpose(max2),transpose(pos2),transpose(dif)];

Gdif = sortrows(Gdif,6);

rejectval = Gdif(1,1);

ID = GR(:,1);

for g2 = 1:length(ID)

if rejectval == ID(g2)

rejects(reject\_pos,:) = GR(g2,:);

GR(g2,:) = [];

else

continue

end

end

rejects(reject\_pos,2) = 0; % To eliminate the highest value

reject\_pos = reject\_pos +1;

elseif rs >= limit

for s = 1:rs

[max1(s),pos1(s)] = max(SL(s,2:5));

tempSL = SL;

tempSL(s,3) = 0;

[max2(s),pos2(s)] = max(tempSL(s,2:5));

dif(s) = max1(s)-max2(s);

snum(s) = SL(s,1);

end

Sdif = [transpose(snum),transpose(max1),transpose(pos1),...

transpose(max2),transpose(pos2),transpose(dif)];

Sdif = sortrows(Sdif,6);

rejectval = Sdif(1,1);

ID = SL(:,1);

for s2 = 1:length(ID)

if rejectval == ID(s2)

rejects(reject\_pos,:) = SL(s2,:);

SL(s2,:) = [];

else

continue

end

end

rejects(reject\_pos,3) = 0;

reject\_pos = reject\_pos +1;

elseif rr >= limit

for r = 1:rr

[max1(r),pos1(r)] = max(RA(r,2:5));

tempRA = RA;

tempRA(r,4) = 0;

[max2(r),pos2(r)] = max(tempRA(r,2:5));

dif(r) = max1(r)-max2(r);

snum(r) = RA(r,1);

end

Rdif = [transpose(snum),transpose(max1),transpose(pos1),...

transpose(max2),transpose(pos2),transpose(dif)];

Rdif = sortrows(Rdif,6);

rejectval = Rdif(1,1);

ID = RA(:,1);

for r2 = 1:length(ID)

if rejectval == ID(r2)

rejects(reject\_pos,:) = RA(r2,:);

RA(r2,:) = [];

else

continue

end

end

rejects(reject\_pos,4) = 0;

reject\_pos = reject\_pos +1;

elseif rh >= limit

for h = 1:rh

[max1(h),pos1(h)] = max(HU(h,2:5));

tempHU = HU;

tempHU(h,5) = 0;

[max2(h),pos2(h)] = max(tempHU(h,2:5));

dif(h) = max1(h)-max2(h);

snum(h) = HU(h,1);

end

Hdif = [transpose(snum),transpose(max1),transpose(pos1),...

transpose(max2),transpose(pos2),transpose(dif)];

Hdif = sortrows(Hdif,6);

rejectval = Hdif(1,1);

ID = HU(:,1);

for h2 = 1:length(ID)

if rejectval == ID(h2)

rejects(reject\_pos,:) = HU(h2,:);

HU(h2,:) = [];

else

continue

end

end

rejects(reject\_pos,5) = 0;

reject\_pos = reject\_pos +1;

end

end

end

% REcursive Sorting?

function [GR,SL,RA,HU] = REsort(GR,SL,RA,HU,REmat)

[row,col] = size(REmat); % Gets size of reject matrix

for m = 1:1:row

[smax(m),pos(m)] = max(REmat(m,2:5));

end

% +1 because we don't want to overwrite the last value

[Gr,Gc] = size(GR);

[Sr,Sc] = size(SL);

[Rr,Rc] = size(RA);

[Hr,Hc] = size(HU);

Gr = Gr+1;

Sr = Sr+1;

Rr = Rr+1;

Hr = Hr+1;

for h = 1:length(pos)

if pos(h) == 1

GR(Gr,:) = REmat(h,:);

Gr = Gr + 1;

elseif pos(h) == 2

SL(Sr,:) = REmat(h,:);

Sr = Sr + 1;

elseif pos(h) == 3

RA(Rr,:) = REmat(h,:);

Rr = Rr + 1;

elseif pos(h) == 4

HU(Hr,:) = REmat(h,:);

Hr = Hr + 1;

end

end

end

end