## Algorithms:

First-Fit: Pack items in sequence to the first bin that still has capacity to fit it.

First-Fit Decreasing: First sort the items in descending order (by area?) and then apply First-Fit.

Best-Fit: Pack items in sequence to the bin with the smallest capacity avail capable to fit it.

Guillotine packing: place part, guillotine the space around the part such that there are 2 new spaces, not overlapping, bordering the part.

Consider all the combinations of stock sheets, where the total area is bigger than the total area of the items to be packed, and sorted ascendingly. Using the first combination would yield optimum solution, but may not be feasible.

## Naming

Spaces: areas available for objects to be placed

Part: an object to be placed

Board: the complete stock prior to subdividing it and prior to placing any parts

Packing: the act of placing a part in a space and any other actions required to continue packing another part

## Left/top first with combined area…

The idea with this algorithm is to combine the spaces in order to have more options for packing other parts.

We start packing a single board with origin “o” and termination point “x”

S0

Sx

We will keep a collection op vectors S where parts may be packed in future. Each vector Si will contain an origin point and a value x’. The value of x’ will be the x value to which the previously packed part reaches. No part can be placed above the origin left of x’. x’ may be the same value of x of the origin.

We have a list of parts P:

We define our initial collection of points as 2 points S0, and Sx.

S = [S0, Sx]

After packing a part at S0, we add two new points to the collection, top-right (S1-1), and bottom left (S1-2) of the packed part and remove S0.

At this point, we need to investigate if we could move S1-1 more up. If S1-1.x is right of S0.x’, change   
S1-1.y = S0.y

S1\_1

S0

S1\_2

Sx

S1\_2.x’

S0.x’

S = [S1-2, S1-1, Sx] – ordered asc by x-coordinate, then y

We loop through the points Si in S, and seek for a part that will fit at the point Si.

If no part is found, remove the point.???....this might influence positioning on other points, so don’t think so.

For this, we will need to calculate the maximum length and width for a part to fit at Si.

For the maximum length, we simply subtract Sx.x – Si.x

For the maximum width however, is more complex: From Si travers down, until a point Sj is found with Sj.y = Si.y, and Sj.x > Si.x.

S1\_1.x’

S1\_1

S0

S0.x’

S1\_2.x’

S1\_2

S1\_1.w

S1\_2.w

Sx

Before placing parts, we need to sort them…

If we sort parts desc by length, then width:

If we sort parts desc by width, then length:

New idea: represent the open space on a board by a collection of rectangles stretching from the last part placed to the edge of the board

Placed part

Space2

Space1

Placing the next part, we need a way to evaluate the open space to be the combination of these rectangles and a part can be placed, spanning them. Parts should be ordered desc according to width, but locations further to the left should be favoured….so spaces should be ordered by x and y displacements ascending…maybe even distance from the origin(?) so ascending on the sum of x2 and y2???? (no need to root it)….

Loop through the spaces, and place the biggest(first) part that will fit on the space(or spanning it and spaces next to it).

When a part is placed, we need to modify the spaces it overlaps:

If it covers the width of the space, move the origin of the space to the right(and shorten)

If it covers the width partially, create a new space horizontal below the part and modify the  
 one on the right to the right and less high, to only cover the part.

This algorithm may create dead spaces behind a part, but sorting by width descending should minimize this.

Placed part3

Space3

Space4

Placed part4

Placed part3

Placed part2

Space2

Space1

Placed part1

Placed part1

Space1

Space2

Space3

Placed part2

Space3

Placed part1

Space1

Space2

Space3

Placed part2

Placed part1

Space1

Space2

Define collection X, a list of possible x values to test at

If l>L or w>W

Can not place part

(x;y)=(X0;0)

Loop through placed parts

If overlap

If (w > W - overlapped part.bottom )

If l>L-X(++)

Can not place

(x;y)=(X(++); 0)

Restart looping placed parts at beginning

y=overlapped part.bottom

place part at (x;y)

add part.right to X

Given

Rectangle B, (0;0, L;W),

Rectangle P(x;y, l;w) a rectangle to be placed inside B

C[], a list of rectanlges already placed inside B

Q=[0, Ci.right, B.L] //sorted list of x values to try and place P

If P.l>B.L or P.w>B.W then exit(-1)

Let j = 0 // the index of Q to use when placing P

Let P = (Q0;0, l;w)

Let i = 0 // the index of the next placed rectangle to test for overlapping

while i < C.length

if Ci . overlap(P)

if P.w > B.W-Ci.bottom

if P.l > B.L – Q(j+1)

exit(-1)

P.x= Q(++j)

P.y=0

i = 0

else

P.y = Ci.bottom

i=0

else

i++

C.add(P)

If ! Q.Contains(P.right)

Q.add(P.right)

Note: maybe maintain 2 collections: Q[0, Ri.right,B.L] and S[0, Ri.bottom]

C=[P1, P2, P3]

Q=[0,P1.l,P2.l,P3.l,L]

j=0

i=2

B=(0;0, L,W)