Stephen Belden

Meghan Haukaas

Chris Ruiz

**Bin Packing Test Cases**

**Test Case Description**

We will be building and testing two different bin packing algorithms. Both algorithms respect the rules of bins, both perform in the same time complexity, and both algorithms are expected to fill bins efficiently. As such, most test cases can be shared between both algorithms. Only with randomly generated input data do we expect to see differences in how each algorithm fills bins.

**General Test Case Construction**

1. Input: Nothing  
   Expected Output: 1 bin, 0% full
2. Input: 1 object equal in size to 1 bin  
   Expected Output: 1 bin, 100% full
3. Input: 1 object larger than 1 bin  
   Expected Output: 1 bin, 0% full, and an error
4. Input: n objects of size 1/n bin  
   Examples: 2 objects of size 1/2 bin; 3 objects of size 1/3 bin  
   Expected Output: 1 bin, 100% full
5. Input: n objects of size greater than 1/n bin  
   Example: 3 objects of 2/3 bin  
   Expected output: n bins, each more than 50% full
6. Input: A number of unequal objects adding up to size 1 bin  
   Example: 0.5 \* size, 0.3 \* size, 0.2 \* size  
   Expected output: 1 bin, 100% full
7. Input: A large set of objects that can be fit perfectly into n bins, inserted randomly  
   Expected output for first fit: no more than 2 \* n bins  
   Expected output for best fit: no more than 1.7 \* n bins
8. Input: n objects of size k (all less than the size of a bin), inserted randomly  
   Example: any valid random input  
   Expected output: any valid output (no more than n bins, no bins over 100%)  
   **This test case will be used to verify time complexity. As n changes, completion time should change with O(N log N)**

**Test Cases for Specific Edge Cases**

For these test cases, the capacity of each bin is 1

1. Insert: 0.6, 0.6, 0.4  
   Expected Output: One bin holds 0.6 and 0.4, another bin holds 0.6  
   Reasoning: This test makes sure that both algorithms look back through existing bins before creating a new bin.
2. Insert: 0.5, 0.6, 0.3, 0.4  
   Expected Output for first fit: binA{0.5, 0.3} binB{0.6,0.4}  
   Expected Output for best fit: binA{0.5, 0.4} binB{0.6, 0.3}  
   Reasoning: This test shows a clear difference between how first fit and best fit insert elements.
3. Insert: 0.5, 0.6, 0.3, 0.1, 0.5  
   Expected Output for first fit: binA{0.5, 0.3, 0.1} binB{0.6} binC{0.5}  
   Expected Output for best fit: binA{0.5, 0.5} binb{0.6, 0.3, 0.1}  
   Reasoning: This test shows a case in which best fit is more space efficient than first fit.