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 ½ 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: insert 0.2, 0.3, 0.5  
   Expected output: 1 bin, 100% full
7. Input: A large set of objects that can be fit perfectly into n bins, inserted randomly  
   Example: insert 0.2, 0.5, 0.5, 0.3, 0.2, 0.5, 0.2, 0.3, 0.3  
   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**

bin = 10

1. nothing

expected: no bins

2. add 9

expected: bin1: 9

3. add 10

expected: bin1: 10

4. add 11

expected: ERROR!

5. add 6

add 6

add 4

expected: bin1: 6,4

bin2: 6

6. add 5

add 6

add 3

add 4

expected: bin1: 5,3

bin2: 6,4

// depending on how best fit actually works

7. add 5

add 6

add 3

add 1

add 5

expected:

first fit: bin1: 5,3,1

bin2: 6

bin3: 5

best fit: bin1: 5,5

bin2: 6,3,1

8. add 5

add 6

add 2

add 1

expected:

first fit: bin1: 5,2,1

bin2: 6

best fit: bin1: 5

bin2: 6,2,1

// is this how best fit would handle this? Getting 1 bin as full as possible. Or would it be more like bin1: 5,1,2 and bin2: 6 where it just makes the first one full like first fit? Or like bin1: 5,2 and bin2: 6,1 where it makes the empty space in each the same? In any case the total amount of empty space is the same, it’s just where the empty space is.

1. add 5

add 11

add 5

ERROR!