Bin Packing Algorithms.
---

1. A classical problem, with long and interesting history.
   One of the early problems shown to be intractable. Lends to simple
   algorithms that require clever analysis.

2. You are given N items, of sizes s1, s2, ..., sN. All sizes are such
   that 0 < si <= 1. You have an infinite supply of unit size bins.
   Goal is to pack the items in as few bins as possible.

   EXAMPLE: 0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8

3. Many many applications:
 placing data on multiple disks; job scheduling;
 packing advertisements in fixed length radio/TV station breaks;
 or storing a large collection of music onto tapes/CD's, etc.

4. Two versions.
 Online---items arrive one at a time (in unknown order), each must
  be put in a bin, before considering the next item.
 Offline--all items given upfront.

   The online problem would seem more difficult. In fact, it's easy to
   convince ourselves that a ONLINE algorithm cannot always get the
   optimal solution. Consider the following input:

 M "small" items of size 1/2 - e, followed by M "large" items of
 size 1/2 + e, for any 0 < e < 0.001.

    The optimal solution is to pack them in pairs (one small, one large);
    this requires M bins.

    Now, the ONLINE algorithm doesn't know what's coming down the pipe,
    or even how long the pipe is. So, for instance, what should it do
    with the first M small items. If it packs 2 of them in each bin,
    then it will be stuck when the second half arrives, with M large
    items. On the other hand, if it puts one small items in each bin
    in the first half, then we can just stop the input right there, in
    which case the algorithm would have used twice as many bins as needed.

5. This ad hoc argument is not a proof. But we can turn this into a
    formal proof, and show the following LOWER BOUND.

    There exist inputs that can force ANY online bin-packing algorithm
    to use at least 4/3 times the optimal number of bins.

    PROOF.
    An important observation is that because we (the adversary) can
    truncate the input whenever we like, the algorithm must maintain
    its guaranteed ratio AT ALL points during its course.

    Consider the input sequence:
 I1, sequence of M small items of size (1/2 - e), followed by
 I2, sequence of M large items of size (1/2 + e).
    Let's consider the state of the online algorithm after it has processed
    I1. Suppose it has used b number of bins. At this point, the optimal
    solution uses M/2 bins, so if the online algorithm beats 4/3 ratio,
    it must satisfy:
  b/(M/2) < 4/3 ==>   b/M < 2/3. (*)

    Now consider the state of the online algorithm after all items have
    been processed. Since all new items have size > 1/2, every NEW bin
    created after the first b bins will have exactly one item put in it.
    (Some items may go into the first b bins.)
    Since only the first b bins can have 2 items, and the remaining bins
    have 1 item each, we see that packing 2M items will require at least
    (2M - b) bins. Again, since the optimal at this stage is M bins, the
    online algorithm must guarantee that (2M - b) < 4M/3, which simplifies to
  b/M > 2/3.   (**)

    But now we have a contracdiction, (*) and (**). Thus, NO online
    algorithm can beat the 4/3 ratio.

   We now show 3 very simple online algorithms that each uses at most

twice the optimal bins.

6. Next Fit.
When processing the next item, see if it fits in the same bin
as the last item. Start a new bin only if it does not. Incredibly simple
to implement (linear time.)

Example:

```
empty empty empty empty empty
0.5  0.1
0.2 0.4 0.7 0.3 0.8
```

Next Fit also has a simple worst-case analysis.

Theorem: If M is the number of bins in the optimal solution, then
Next Fit never uses more than 2M bins.
There exist sequences that force Next Fit to use 2M-2 bins.

Proof.
Consider any two adjancent bins. The sum of items in these
two bins must be > 1; otherwise, NextFit would have put all
the items of second bin into the first. Thus, total occupied
space in (B1 + B2) is > 1. The same holds for B3+B4 etc....
Thus, at most half the space is wasted, and so Next Fit uses
at most 2M bins.

For the lower bound, consider the sequence in which si = 0.5 for
i odd, and si = 2/N for i even. (Suppose N is divisible by 4.)
Then, the optimal puts all 0.5 items in pairs, using N/4 bins.
All small items fit in a single bin, so the opt is N/4 + 1.
Next Fit will put 1 large, 1 small in each bin, requiring N/2 bins.

Lower Bound:

```
0.5 0.5 ... 0.5  2/N
0.5 0.5 ... 0.5  2/N
     ...

B1 B2  B_{N/4}  B_{N/4 + 1}


empty empty ... empty  empty
2/N 2/N  2/N  2/N
0.5 0.5  0.5  0.5

B1 B2    B_{N/2}
```

7. First Fit.
Next Fit can be easily improved: rather than checking just the
last bin, we check all previous bins to see if the next item will
fit. Start a new bin, only when it does not.

Example:

```
empty empty empty empty
0.1
0.5 0.3
0.2 0.4 0.7 0.8
```

First Fit easy to implement in O(N^2) time. With proper data structures,
it can be implemented in O(N log N) time.

Theorem: First Fit never uses more than 2M bins, if M is the optimal.

Proof. At most one bin can be more than half empty: otherwise the
contents of the second half-full bin would be placed in the first.

Theorem: If M is the optimal number of bins, then First Fit never
uses more than 1.7M bins. On the other hand, there are
sequences that force it to use at least 17/10 (M-1) bins.

The upper bound proof is quite complicated.
We show an example that forces First Fit to use 10/6 times optimal.

Consider the sequence: 6M items of size 1/7 + e; followed by 6M
items of size 1/3 + e; followed by 6M items of size 1/2 + e.

Optimal strategy is to pack each bin with one from each group,
requiring 6M bins.

When First Fit is run, it packs all small items first, in 1 bin.
It then packs all medium items, but requires 6M/2 = 3M bins.
(Only 2 per bin fit.) It then requires 6M bins for the large items.
Thus, in total First Fit uses 10M bins.

```
   empty  empty  empty
1/7
1/7
...
1/7  1/3
1/7
1/7
...
1/7  1/3 + e
1/7  1/3 + e  1/2 + e
```

9. Best Fit.
The third strategy places the next item in the *tightest* spot.
That is, put it in the bin so that smallest empty space is left.

Example. 0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8

```
empty empty  empty
0.1
0.5  0.3
0.2 0.4 0.7 0.8
```

Also easy to implement in O(N log N) time.
Unfortunately, the generic bad cases for First Fit etc. apply
to Best Fit also. Best Fit never uses more than 1.7 times optimal.
Complicated analysis, omitted.

10. Offline Algorithms.
If we can view the entire sequence upfront, we should expect to
do better. With exhaustive enumeration, of course, we can find the
optimum. But even offline bin packing is not easy *if* we have only
a polynomial amount of time. (NP-Complete.)

A trouble with online algorithms is that packing large items is
difficult, especially if they occur late in the sequence. We can
circumvent this by *sorting* the input sequence, and placing the
large items first. With sorting, we get First Fit Decreasing and
Best Fit Decreasing, as offline analogs of online FF and BF.

With sorting, the input sequence becomes:
0.8, 0.7, 0.5, 0.4, 0.3, 0.2, 0.1

Applying First Fit Decreasing, we get an optimal.

```
  0.1
0.2 0.3 0.4
0.8 0.7 0.5
```

Note that the bad cases that require 10M bins as opposed to 6M
also do not apply here. In fact, we show the following theorem.

THEOREM: First Fit Decreasing uses at most (4M + 1)/3 bins
if the optimal is M.

11. First Fit Decreasing.
The proof of FFD's performance depends on two technical observations.

1. Suppose the N items have been sorted in descending order of size;
s1 > s2 > ... > sN. If the optimal packing uses M bins, then
all bins in the FFD after M have items of size <= 1/3.

2. The number of items FFD puts in bins after M is at most M-1.

Proof of 1. By contradtiction. Suppose si is the first item to
be put in bin M+1, and si > 1/3. Therefore, we also have that
s1, s2, ..., si-1 > 1/3. From this, it follows that each of the
first M bins has at most 2 items each.

Claim. The state of FFD just before si was placed is the following:

the first few bins have exactly 1 item, remaining have 2 items.
If not, then there must be two bins Bx, By, with x < y, such
that Bx has two items x1, x2, and By has 1 item y1.
Since x1 was put in earlier bin, x1 >= y1.
Since x2 was put in before si, x2 >= si.
Thus, x1 + x2 >= y1 + si. But this implies that si could have
fit in By, which contradicts our assumption. Thus, if si > 1/3,
then the first M bins must be arranged so that first j have
1 item; the next M-j have two items.

To finish the proof, we now argue that there is no way put all the
items in M bins, contradicting the assumption of optimality.

No two items from s1, s2, ..., sj can be put in a single bin;
if so, FFD would have done it. Because FFD faild to put any
of the items s_{j+1}, ..., s_{i-1} into first j bins, in any
solution (including optimal), there must be j bins that do
not contain any item from s_{j+1}, ..., s_{i-1}. Thus, all these
items must be contained in the remaining M-j bins. Further,
there are 2(M-j) such items (because in FFD each of these
M-j bins had 2 items).

Now, if si > 1/3, then there is no way for si to be placed
in any of these M bins: it can't fit in the first j because
otherwise FFD would have done it; it can't go in the remaining
M-j because each of them already has two items of sizes > 1/3.
Thus, the optimal would require at least M+1 bins, which is
a contradiciton!

So, it must be that si <= 1/3.

Proof of 2.

Suppose that there are at least M objects put in the extra bins.
Since all items fit in M bins, we have sum_{i=1}^N si <= M.
Suppose bin j is filled with total weight Wj. Suppose the
first M extra objects have sizes x1, x2, ..., xM.
Because the items packed by FFD in first M bins plus the
first M extra are subset of total, we have

\sum_{i=1}^N si >= \sum_{j=1}^M Wj + \sum+{j=1}^M xj
         >= \sum_{j=1}^M (Wj + xj)

Now, Wj + xj > 1, for each j; otherwise FFD would put xj in Bj.
Thus, \sum_{i=1}^N si > \sum_{j=1}^M 1 > M

But that's impossible because all si fit in M bins.
So, there must be only M-1 items in the extra bins.

Proof of Theorem.
There are M-1 extra items, each of size <= 1/3. Thus, there can
be at most (M-1)/3 extra bins. Thus, the total number of bins
needed by FFD is (4M+1)/3.

12. More complicated Theorem.
If M is the optimal number of bins, then FFD never uses
more than 11M/9 + 4 bins. There are sequences for which
FFD uses 11M/9 bins.
when