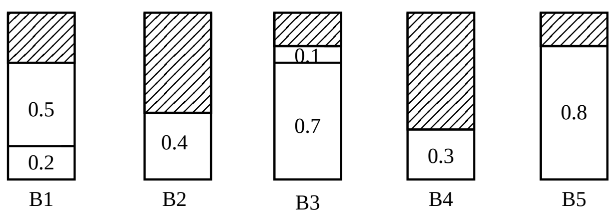
Bin Packing

- Given n items with sizes s_1, s_2, ..., s_n such that $0 \leq s_i \leq 1$ for $i \leq i \leq n$, pack them into the fewest number of unit capacity bins.
- Problem is NP-hard (NP-Complete). There is no known polynomial time algorithm for its solution, and it is conjectured none exists.
- On-line algorithms: Each item must be placed in a bin (and never moved again) before the next item can be viewed (processed).
- Off-line algorithm: You may view all items before placing any item into a bin.
- We present algorithms to generate suboptimal solutions (approximations).

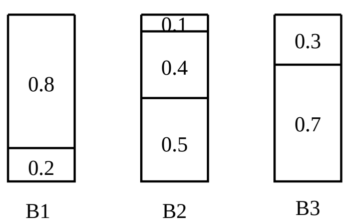
Next Fit (aprox. alg.)

(on-line algorithm): Check to see if the current item fits in the current bin. If so, then place it there, otherwise start a new bin.

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



Next Fit



Optimal Packing

Theorem 10.2

Let M be the number of bins required to pack a list I of items optimally. Next Fit will use at most $2M$ bins.

Proof:

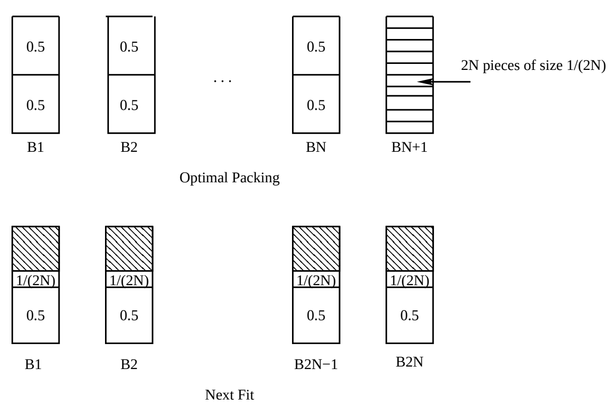
- Let $s(B_i)$ be the sum of the sizes of all the items assigned to bin B_i in the solution generated by Next Fit.
- For any two adjacent bins (B_j and B_{j+1}), we know that $s(B_j) + s(B_{j+1}) > 1$.
- Let k be the number of bins used by Next Fit for list I.
- We prove the case when k is even, the proof for the other case is omitted (as it is similar and establishes that $k < 2M + 1$).

- As stated above, $s(B_1) + s(B_2) > 1$, $s(B_3) + s(B_4) > 1$, ..., $s(B_{k-1}) + s(B_k) > 1$.
- Adding these inequalities we know that $\sum s(B_i) > k/2$.
- By definition $OPT = M > k/2$.
- The solution $SOL = k < 2M$.
- Therefore, $SOL \leq 2M$ (because of odd case).

Theorem 10.2 (2nd Part)

There exist sequences such that Next Fit uses $2M - 2$ bins, where M is the number of bins in an optimal solution.

- Define an instance with $4N$ items
- The odd numbered ones have s_i value $1/2$, and the even number ones have s_i value $1/(2N)$.

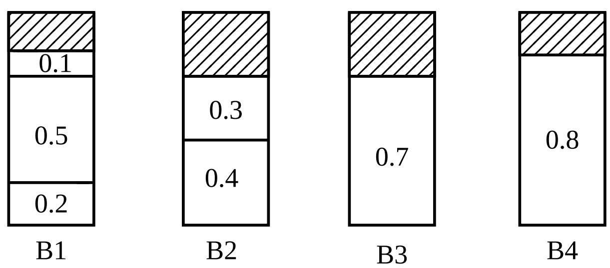


- $OPT = N + 1 = M$
- Therefore, $N = M - 1$
- Solution $SOL = 2N = 2M - 2$.

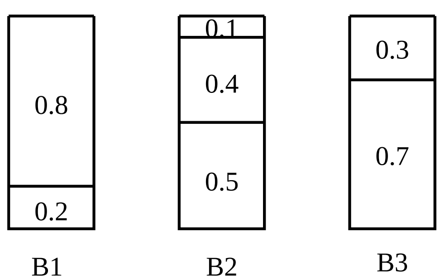
First Fit (aprox. alg.)

(on-line algorithm): Scan the bins in order and place the new item in the first bin that is large enough to hold it. A new bin is created only when an item does not fit in the previous bins.

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



First Fit



Optimal Packing

Can be easily implemented to take $O(n^2)$ time.

Better Implementation

- Can be implemented to take $O(n \log n)$ time.
- Idea: Use a red-black tree (or other balanced tree) with height $O(\log n)$.
- Each node has three values: index of bin, remaining capacity of bin, and best (largest) in all the bins represented by the subtree rooted at the node.
- The ordering of the tree is by the bin index.

Example

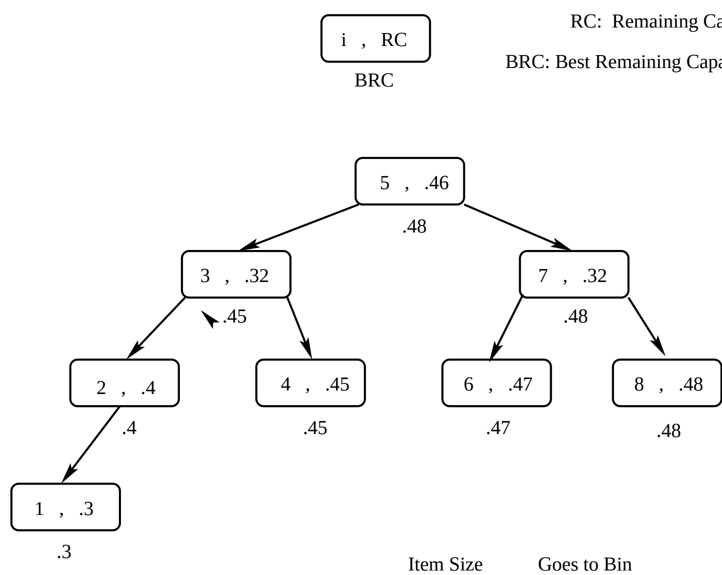
Table 1: Remaining Capacity in Bins

Bin	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
R. Cap.	.3	.4	.32	.45	.46	.47	.32	48

i: bin index

RC: Remaining Capacity

BRC: Best Remaining Capacity in subtree



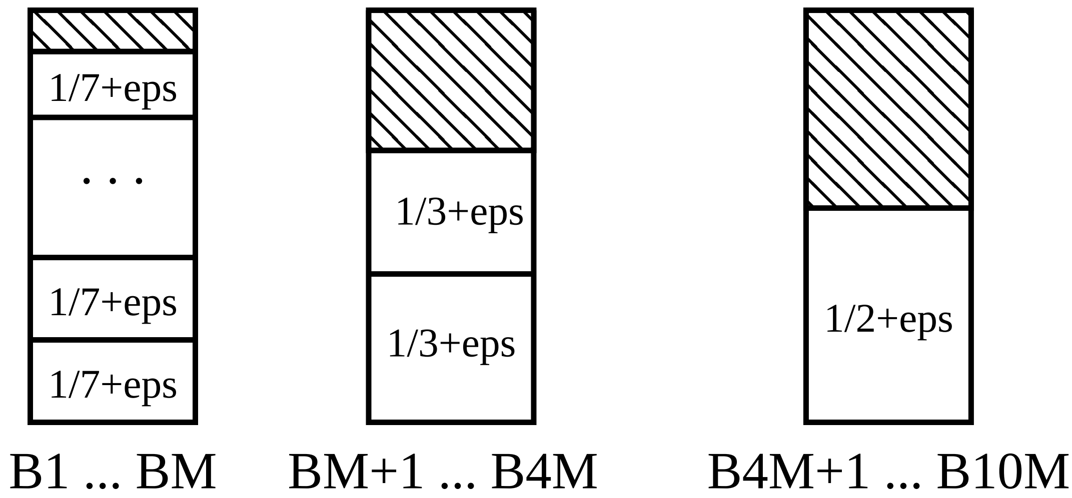
Item Size	Goes to Bin
s <= .3	B1
.3 < s <= .4	B2
.4 < s <= .45	B4
.45 < s <= .46	B5
.46 < s <= .47	B6
.47 < s <= .48	B8

Upper Bounds

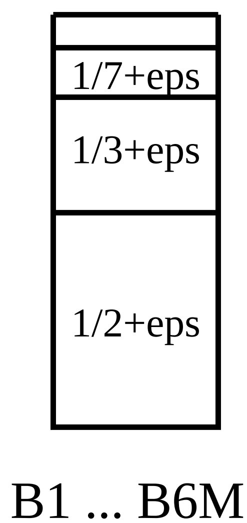
- Claim: Let M be the optimal number of bins required to pack a list I of items. Then First Fit never uses more than $\lceil 1.7M \rceil$.
- Proof was not covered.
- Theorem: There exist sequences such that First Fit uses $1.7(M-1)$ bins.

Example for 1.66...

- $6M$ items of size $\frac{1}{7} + \epsilon$.
- $6M$ items of size $\frac{1}{3} + \epsilon$.
- $6M$ items of size $\frac{1}{2} + \epsilon$.



First Fit

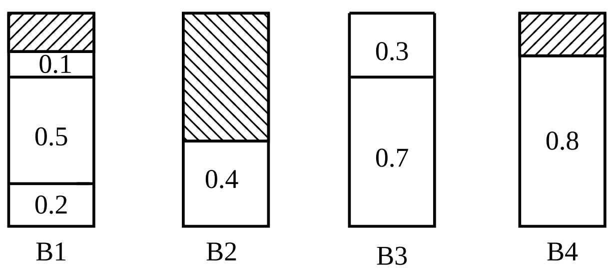


Optimal Packing

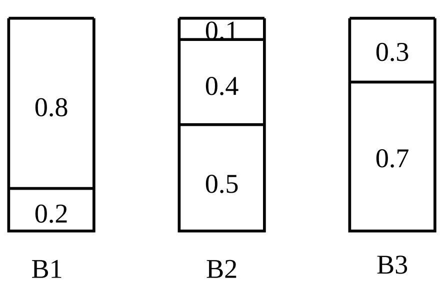
Best Fit (aprox. alg.)

- (on-line algorithm): New item is placed in a bin where it fits the tightest. If it does not fit in any bin, then start a new bin.
- Claim: If OPT uses M bins, then Best Fit uses at most $\sim 1.7M$.
- Can be implemented to take $O(n \log n)$.

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



Best Fit



Optimal Packing

There are problems instances for which First Fit uses fewer bins than Best Fit and vice-versa.

Better Aprox. Alg.

- (off-line algorithm): First Fit Decreasing
- (off-line algorithm): Best Fit Decreasing