Loc Duc Minh Khong - 67798234

CS165

Project 2 - report

I - Bin Packing Algorithms Implemented

All bin pack algorithms are implemented in C++11 and have 3 lists: list of items, list of assignment bins, list of free space of bins

Next Fit: iterate the list of items, comparing the size of the item with the free space of the current bin. If the item size is smaller, place the item to that bin and recompute the free space of the bin. If the item size is larger, assign it to a new bin, also calculate the free space of the bin after placing the item. Runtime O(n);

First Fit: go through the list of items and find the first bin that can store the item. If there is no fit bin, assign the item to a new bin. To improve the runtime of finding the first bin that can hold the item, implement a Zip Tree, in which each node has 3 values: bin index (key), remaining capacity (value) and best remaining capacity in subtree (value), ordered by bin index. Update the BRC and the Zip tree after each iteration. Runtime for first fit is O(nlog n).

Best Fit: go through the list of items and place the item in a bin where it fits the tightest . If there is no fit bin, place the item to a new bin. To improve the runtime of finding the best bin, implement a Zip Tree, in which each node has 2 values: remaining capacity (key), bin index (value), ordered by remaining capacity. Using remove and insert node functions (log n) to update Zip tree after each iteration. Therefore, the runtime is O(nlog n).

First Fit Decreasing (FFD): using merge sort to sort the items by size, from largest to smallest then using first fit algorithm.

Best Fit Decreasing (BFD) : using merge sort to sort the items by size, from largest to smallest then using best fit algorithm.

Zip Tree:

- Struct Node: key, value and rank, left child, right child.
- Node rank is generated randomly and chosen independently from a geometric distribution mean 1;
- Insert and Remove node functions are implemented by following the iterative pseudo code version in the lecture slide.

II - Input data and its distributions

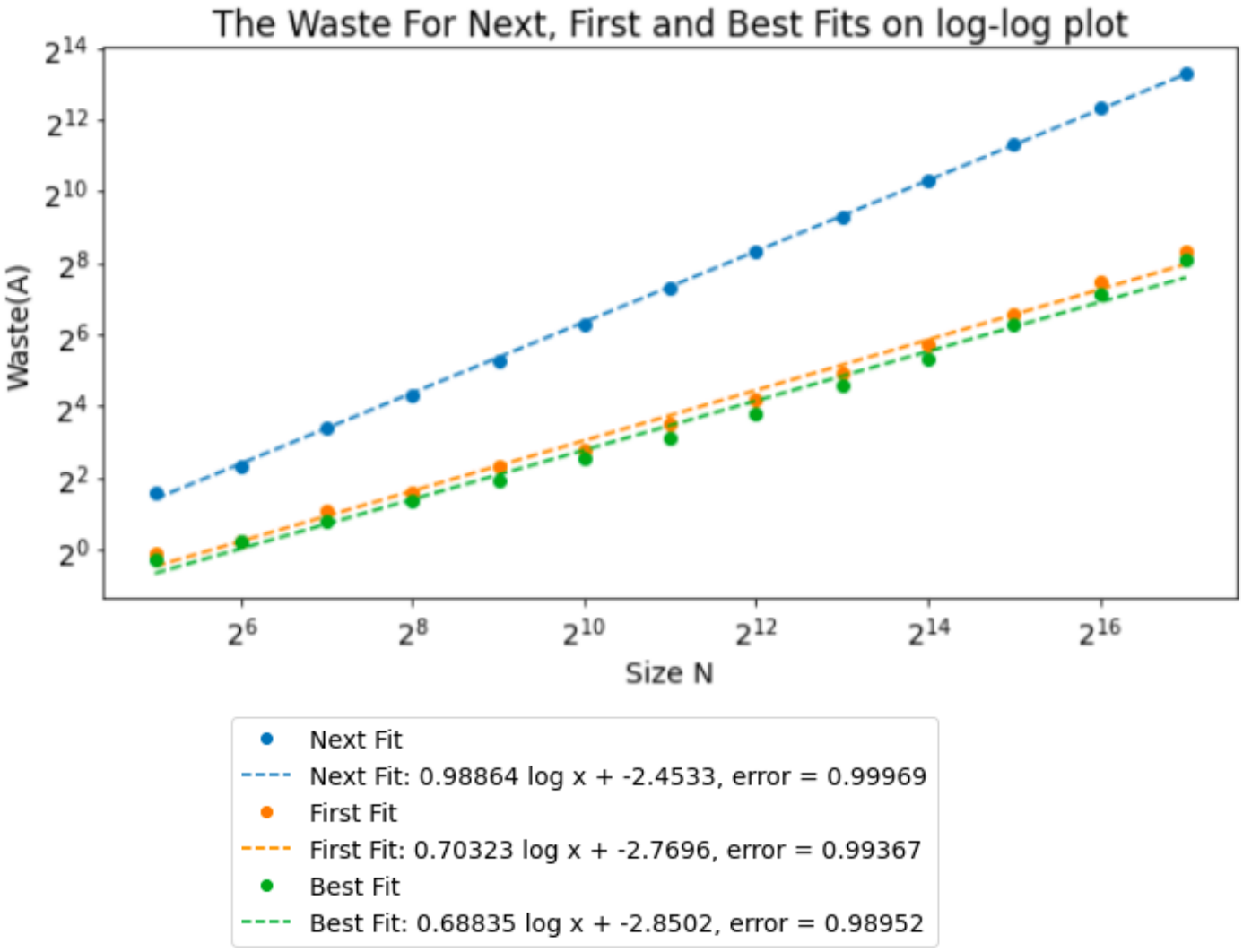
- Generate list of items of length n = {32, 64,128, 256, 512,1024, 2048, 4096, 8192, 16384, 32768, 65536. 131072}. All elements in the list are generated uniformly at random in range [0.0, 0.6).
- For each length n, run 10 tests with a different list of items of size n. Compute Waste(A) by getting the total sizes of the free_space list. The final Waste(A) of each size n is the average Waste(A) after 10 runs.
- Generate random list of items function:

```cpp
std::vector<double> generate_items(int size) {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<double> dis (0.0,0.6);
    std::vector<double> items;

    for(int i = 0; i< size; i++)
        items.push_back(dis(gen));

    return items;
}
```

III - Next Fit, First Fit and Best Fit



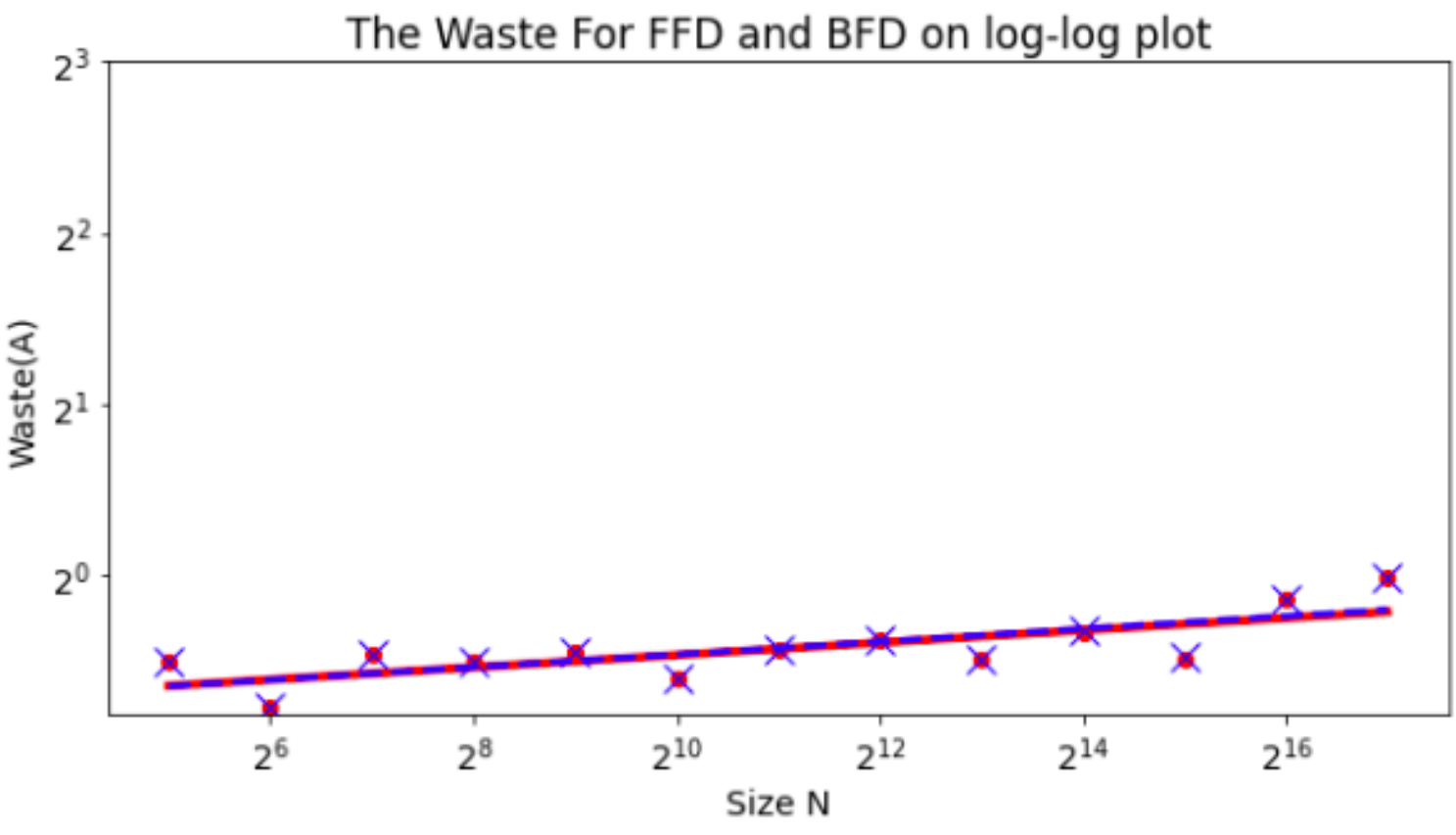
The graph shows that when the size list of items increases, the waste for using the Next, First and Best fit algorithms increase. First fit and Best fit look similar, but Waste(Next Fit) increases much higher than Waste(First Fit) and Waste(Best Fit). It is because Next fit does not scan for previous bins to place items. It immediately creates a new bin if the item does not fit in the current bin. Therefore, Next fit will use more bins than First and Best Fit. And as a result Waste(Next Fit) is higher than Waste(First Fit) and Waste(Best Fit).

The asymptotic slope of Waste(Next Fit) is 0.98864 (O (n)) while First fit is 0.70323 and best fit is 0.68835.

Best fit and First fit are significantly better and efficient than Next Fit for solving the bin-packing problem.

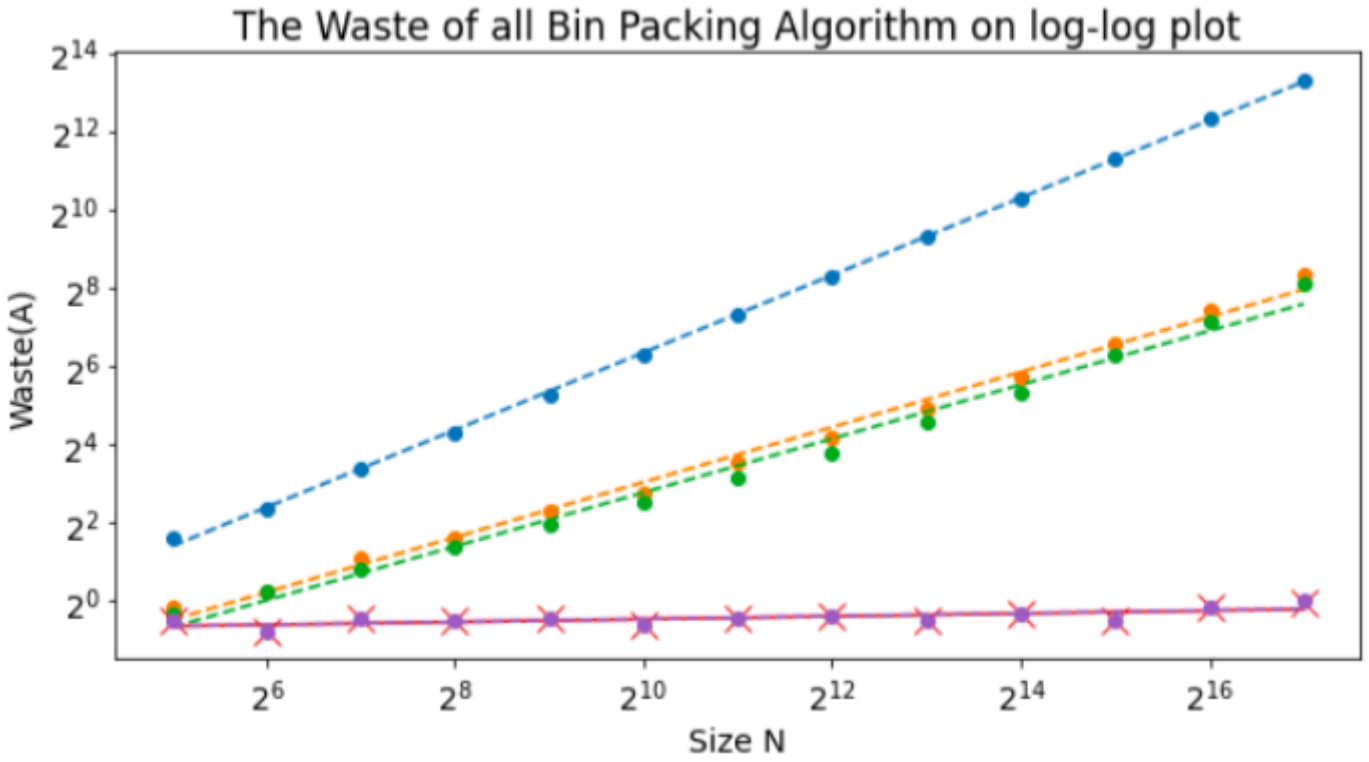
Best fit and First fit look the same, so find the best fit bin or first fit bin have the same performance.

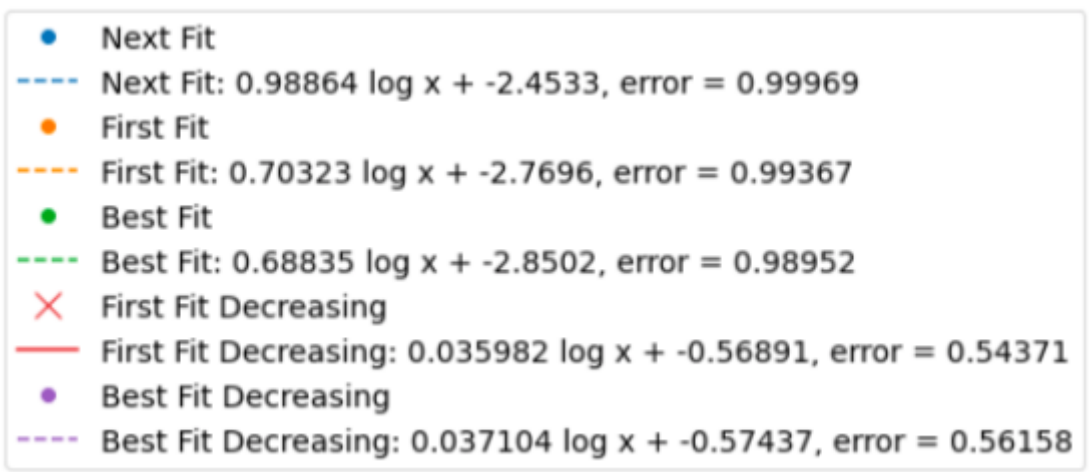
IV - First Fit Decreasing and Best Fit Decreasing



The chart illustrates that Waste(FFD) and Waste(BFD) are very similar and do not change much, although the size of the list of items increases. And their asymptotic slopes are just a bit different and very small (0,035982 and 0.037104 respectively) which make Waste(A) approximately O(1). It seems like packing the largest items first is more likely to produce the optimal solution.

V - Compare Bin Packing Algorithms





As the chart above shows that FFD and BFD are much more efficient than Next fit, First fit and Best fit to solve the bin-packing problem when the item sizes are uniformly distributed. The Waste(FFD) and Waste(BFD) look unchanged while Waste(Next Fit), Waste(First fit) and Waste(best fit) increase considerably.

This proves that sorting and placing items from largest to smallest give the best and stable Waste(A).

VI - Input Sensitivity

The input size may be not large enough to differentiate First Fit Decreasing and Best Fit Decreasing.

The input is only uniformly distributed and in range [0.0,0.6).

VII - Winner/suggested Bin Packing Algorithm

First Fit Decreasing is the winner to solve the bin packing problem because its Waste is much smaller than Waste of Next fit, First fit and Best fit algorithms. Although Best Fit Decreasing has the same performance as First Fit Decreasing, Best Fit Decreasing implementation is more difficult and less efficient than First Fit Decreasing. Best Fit Decreasing has to reallocate the Node after each change in the Zip Tree, but First Fit Decreasing just needs to recalculate the BRC and only changes when a new node is added.