Approx_Mini_1

February 8, 2023

1 Approximation Algorithms Mini Project 1

1.1 Introduction

This is our mini-project on bin packing for the algorithms first fit, best fit, and their sorted versions. These are motivated by problems that exist in the real world, such as optimally placing items in storage trucks to require the least number of trucks.

We empirically validated our results on datasets with 10000 datapoints each. We created three synthetic datasets with probability distributions. These are:

- Uniform distribution: [0, 1]
- Left-tailed: 0.75 probability [0, 0.25], 0.25 probability [0.75, 1]
- Right-tailed: 0.25 probability [0, 0.75], 0.25 probability [0.75, 1]

1.2 Results

[ ]:
```
!rm -r /content/approx-mini1
!cd /content && git clone https://github.com/derexwangmang/approx-mini1
```

```
Cloning into 'approx-mini1'…
remote: Enumerating objects: 131, done.
remote: Counting objects: 100% (131/131), done.
remote: Compressing objects: 100% (103/103), done.
remote: Total 131 (delta 34), reused 120 (delta 27), pack-reused 0
Receiving objects: 100% (131/131), 712.19 KiB | 20.35 MiB/s, done.
Resolving deltas: 100% (34/34), done.
```

[ ]:
```
# this "magic" command makes graphics produced by matplotlib appear in the␣
 ↪notebook.
%matplotlib inline

# make sure we're in the right directory
%cd /content/

import csv
import matplotlib.pyplot as plt
from tabulate import tabulate
```

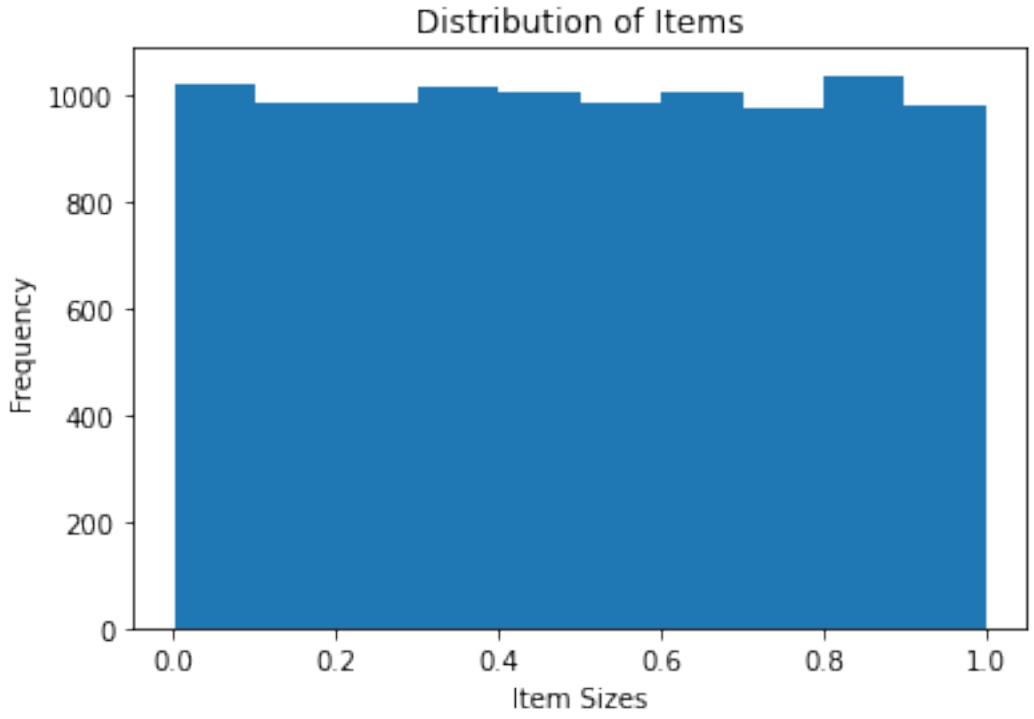
```
/content
```

1.3 Graphs

1.3.1 Uniform Item Distribution

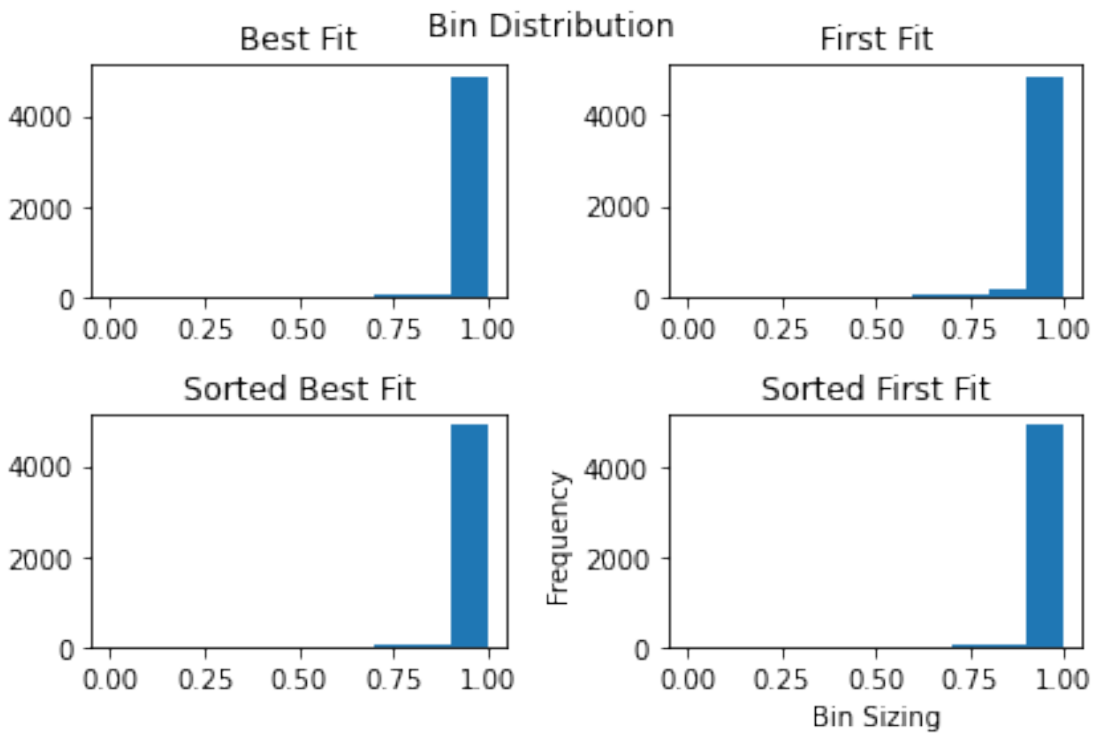
[ ]:
```
%cd /content/approx-mini1/random_10000

with open('data.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    data = []
    for row in reader:
        data.append(row[:])

plt.hist(data)
plt.xlabel("Item Sizes")
plt.ylabel("Frequency")
plt.title("Distribution of Items")
plt.show()
```

```
/content/approx-mini1/random_10000
```



[ ]:
```
with open('best_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    best_fit = []
    for row in reader:
      if row[:]:
```

```
        best_fit.append(row[:])

with open('first_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    first_fit = []
    for row in reader:
      if row[:]:
        first_fit.append(row[:])

with open('sorted_best_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    sorted_best_fit = []
    for row in reader:
      if row[:]:
        sorted_best_fit.append(row[:])

with open('sorted_first_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    sorted_first_fit = []
    for row in reader:
      if row[:]:
        sorted_first_fit.append(row[:])

fig, ax = plt.subplots(nrows=2, ncols=2)
fig.suptitle("Bin Distribution")

ax[0,0].hist(best_fit[0], range=(0, 1))
ax[0,0].set_title('Best Fit')
ax[0,1].hist(first_fit[0], range=(0, 1))
ax[0,1].set_title('First Fit')
ax[1,0].hist(sorted_best_fit[0], range=(0, 1))
ax[1,0].set_title('Sorted Best Fit')
ax[1,1].hist(sorted_first_fit[0], range=(0, 1))
ax[1,1].set_title('Sorted First Fit')

plt.xlabel("Bin Sizing")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```



[ ]:
```
def find_median(arr):
  arr.sort()
  x = len(arr)/2
  if len(arr)%2==0:
    return (arr[len(arr)//2] + arr[len(arr)//2+1]) / 2
  else:
    return arr[len(arr)//2]

def find_mean(arr):
  sumo = 0
  for i in arr:
    sumo += i
  return sumo/len(arr)

table = [["Algorithm", "# of Bins", "Time (ms)", "Median Bin Weight", "Mean Bin␣
 ↪Weight"],
         ["Best Fit", str(int(best_fit[1][0])), str(float(best_fit[2][0])),␣
 ↪str(find_median(best_fit[0])), str(find_mean(best_fit[0]))],
         ["First Fit", str(int(first_fit[1][0])), str(float(first_fit[2][0])),␣
 ↪str(find_median(first_fit[0])), str(find_mean(first_fit[0]))],
         ["Sorted Best Fit", str(int(sorted_best_fit[1][0])),␣
 ↪str(float(sorted_best_fit[2][0])), str(find_median(sorted_best_fit[0])),␣
 ↪str(find_mean(sorted_best_fit[0]))],
```

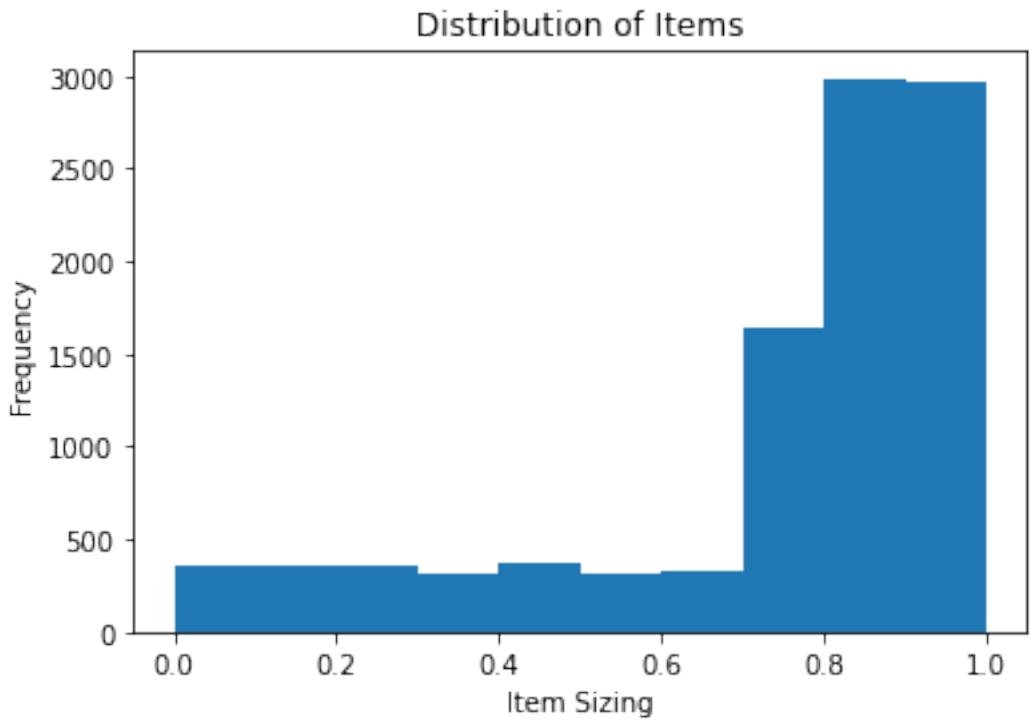
```
        ["Sorted First Fit", str(int(sorted_first_fit[1][0])),␣
↪str(float(sorted_first_fit[2][0])), str(find_median(sorted_first_fit[0])),␣
↪str(find_mean(sorted_first_fit[0]))]

print(tabulate(table, headers='firstrow'))
```

```
Algorithm           # of Bins    Time (ms)    Median Bin Weight    Mean Bin
Weight
----------------  -----------  -----------  -------------------
-----------------
Best Fit                 5087      1206.29              0.99695
0.981885
First Fit                5139      1095.82              0.990475
0.97195
Sorted Best Fit          5025      1780.4               0.999893          0.994
Sorted First Fit         5025      1176.07              0.999885          0.994
```

1.3.2 Left Tailed Distribution

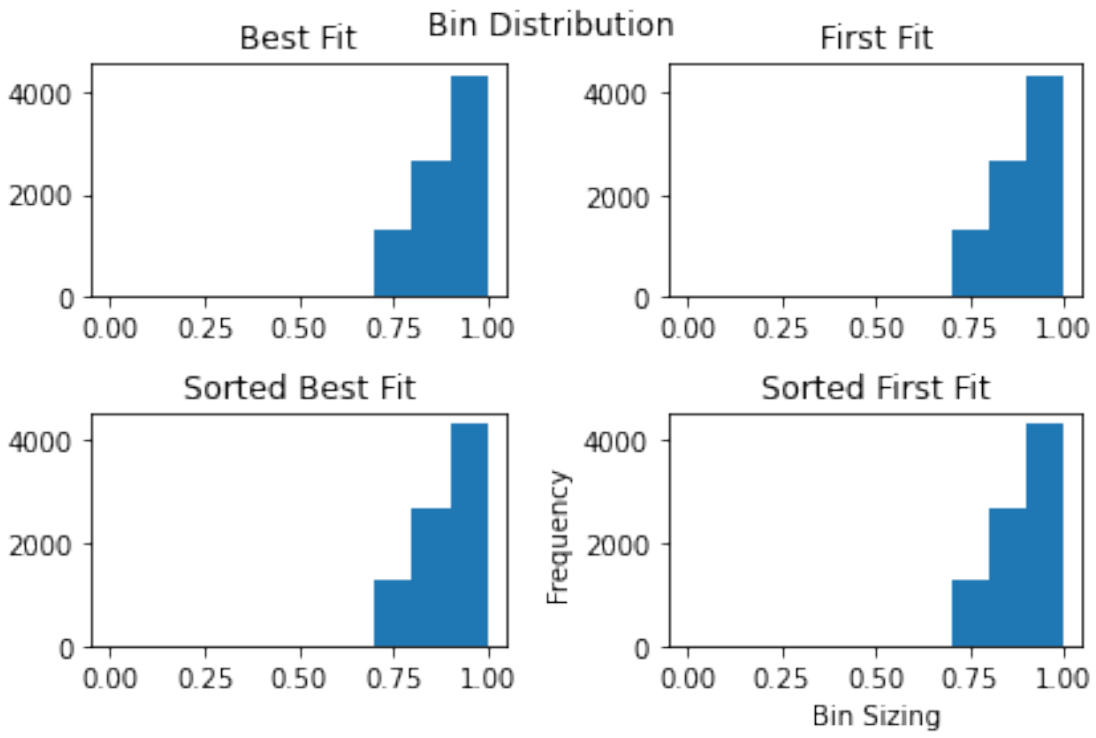
[ ]:
```
%cd /content/approx-mini1/left_tailed_10000

with open('data.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    data = []
    for row in reader:
        data.append(row[:])

plt.hist(data)
plt.xlabel("Item Sizing")
plt.ylabel("Frequency")
plt.title("Distribution of Items")
plt.show()
```

```
/content/approx-mini1/left_tailed_10000
```



[ ]:
```
with open('best_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    best_fit = []
    for row in reader:
      if row[:]:
        best_fit.append(row[:])

with open('first_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    first_fit = []
    for row in reader:
      if row[:]:
        first_fit.append(row[:])

with open('sorted_best_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    sorted_best_fit = []
    for row in reader:
      if row[:]:
        sorted_best_fit.append(row[:])

with open('sorted_first_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
```

```
    sorted_first_fit = []
    for row in reader:
      if row[:]:
        sorted_first_fit.append(row[:])

fig, ax = plt.subplots(nrows=2, ncols=2)
fig.suptitle("Bin Distribution")

ax[0,0].hist(best_fit[0], range=(0, 1))
ax[0,0].set_title('Best Fit')
ax[0,1].hist(first_fit[0], range=(0, 1))
ax[0,1].set_title('First Fit')
ax[1,0].hist(sorted_best_fit[0], range=(0, 1))
ax[1,0].set_title('Sorted Best Fit')
ax[1,1].hist(sorted_first_fit[0], range=(0, 1))
ax[1,1].set_title('Sorted First Fit')

plt.xlabel("Bin Sizing")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```



[ ]:
```
def find_median(arr):
  arr.sort()
  x = len(arr)/2
  if len(arr)%2==0:
    return (arr[len(arr)//2] + arr[len(arr)//2+1]) / 2
  else:
    return arr[len(arr)//2]

def find_mean(arr):
  sumo = 0
  for i in arr:
    sumo += i
  return sumo/len(arr)

table = [["Algorithm", "# of Bins", "Time (ms)", "Median Bin Weight", "Mean Bin␣
 ↪Weight"],
         ["Best Fit", str(int(best_fit[1][0])), str(float(best_fit[2][0])),␣
 ↪str(find_median(best_fit[0])), str(find_mean(best_fit[0]))],
         ["First Fit", str(int(first_fit[1][0])), str(float(first_fit[2][0])),␣
 ↪str(find_median(first_fit[0])), str(find_mean(first_fit[0]))],
         ["Sorted Best Fit", str(int(sorted_best_fit[1][0])),␣
 ↪str(float(sorted_best_fit[2][0])), str(find_median(sorted_best_fit[0])),␣
 ↪str(find_mean(sorted_best_fit[0]))],
         ["Sorted First Fit", str(int(sorted_first_fit[1][0])),␣
 ↪str(float(sorted_first_fit[2][0])), str(find_median(sorted_first_fit[0])),␣
 ↪str(find_mean(sorted_first_fit[0]))]]

print(tabulate(table, headers='firstrow'))
```

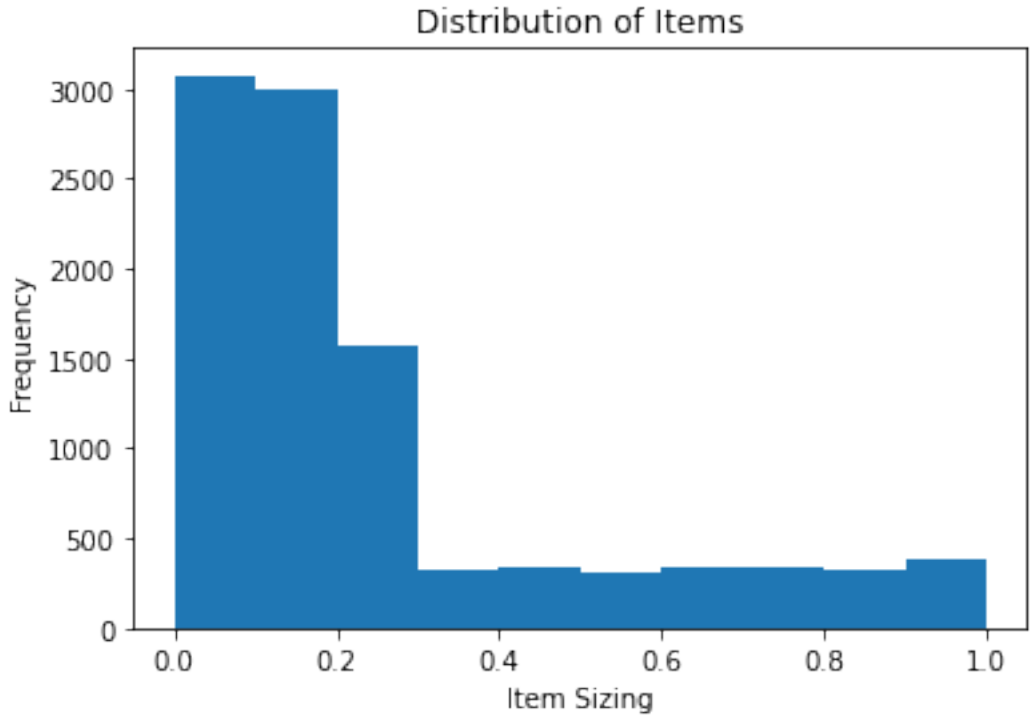
```
Algorithm           # of Bins    Time (ms)    Median Bin Weight    Mean Bin
Weight
----------------  -----------  -----------  -------------------
-----------------
Best Fit                 8284      2035.91             0.907251
0.899225
First Fit                8299      1810                0.907431          0.8976
Sorted Best Fit          8273      2403.25             0.906967
0.900421
Sorted First Fit         8273      2066.61             0.906967
0.900421
```

1.3.3 Right Tailed Distribution

[ ]:
```
%cd /content/approx-mini1/right_tailed_10000

with open('data.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
```

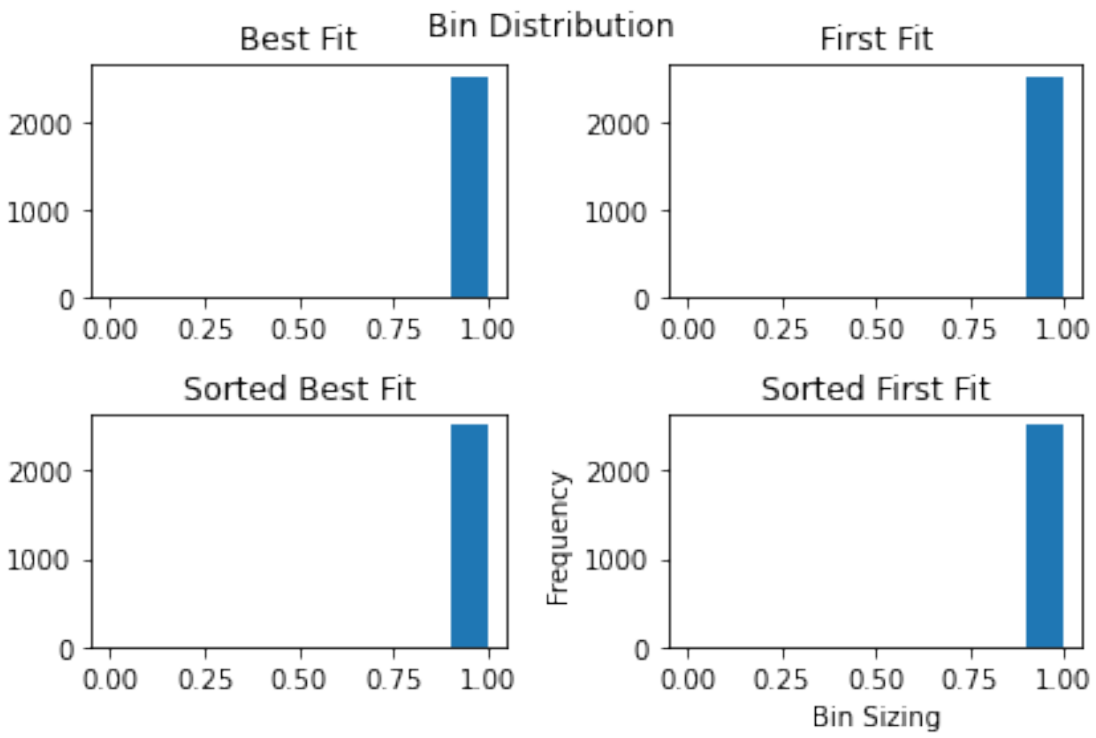
```
    data = []
    for row in reader:
        data.append(row[:])

plt.hist(data)
plt.xlabel("Item Sizing")
plt.ylabel("Frequency")
plt.title("Distribution of Items")
plt.show()
```

```
/content/approx-mini1/right_tailed_10000
```



[ ]:
```
with open('best_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    best_fit = []
    for row in reader:
      if row[:]:
        best_fit.append(row[:])

with open('first_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    first_fit = []
    for row in reader:
```

```
      if row[:]:
        first_fit.append(row[:])

with open('sorted_best_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    sorted_best_fit = []
    for row in reader:
      if row[:]:
        sorted_best_fit.append(row[:])

with open('sorted_first_fit.csv', newline = '') as file:
    reader = csv.reader(file, quoting = csv.QUOTE_NONNUMERIC)
    sorted_first_fit = []
    for row in reader:
      if row[:]:
        sorted_first_fit.append(row[:])

fig, ax = plt.subplots(nrows=2, ncols=2)
fig.suptitle("Bin Distribution")

ax[0,0].hist(best_fit[0], range=(0, 1))
ax[0,0].set_title('Best Fit')
ax[0,1].hist(first_fit[0], range=(0, 1))
ax[0,1].set_title('First Fit')
ax[1,0].hist(sorted_best_fit[0], range=(0, 1))
ax[1,0].set_title('Sorted Best Fit')
ax[1,1].hist(sorted_first_fit[0], range=(0, 1))
ax[1,1].set_title('Sorted First Fit')

plt.xlabel("Bin Sizing")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```



[ ]:
```
def find_median(arr):
  arr.sort()
  x = len(arr)/2
  if len(arr)%2==0:
    return (arr[len(arr)//2] + arr[len(arr)//2+1]) / 2
  else:
    return arr[len(arr)//2]

def find_mean(arr):
  sumo = 0
  for i in arr:
    sumo += i
  return sumo/len(arr)

table = [["Algorithm", "# of Bins", "Time (ms)", "Median Bin Weight", "Mean Bin␣
 ↪Weight"],
         ["Best Fit", str(int(best_fit[1][0])), str(float(best_fit[2][0])),␣
 ↪str(find_median(best_fit[0])), str(find_mean(best_fit[0]))],
         ["First Fit", str(int(first_fit[1][0])), str(float(first_fit[2][0])),␣
 ↪str(find_median(first_fit[0])), str(find_mean(first_fit[0]))],
         ["Sorted Best Fit", str(int(sorted_best_fit[1][0])),␣
 ↪str(float(sorted_best_fit[2][0])), str(find_median(sorted_best_fit[0])),␣
 ↪str(find_mean(sorted_best_fit[0]))],
```

```
        ["Sorted First Fit", str(int(sorted_first_fit[1][0])),␣
↪str(float(sorted_first_fit[2][0])), str(find_median(sorted_first_fit[0])),␣
↪str(find_mean(sorted_first_fit[0]))]]

print(tabulate(table, headers='firstrow'))
```

```
Algorithm           # of Bins    Time (ms)    Median Bin Weight    Mean Bin
Weight
----------------  -----------  -----------  -------------------
-----------------
Best Fit                 2527      595.695             0.999251
0.995672
First Fit                2532      562.403             0.99762
0.993706
Sorted Best Fit          2517      930.645             0.999965
0.999628
Sorted First Fit         2517      812.402             0.99996
0.999628
```

1.4 Analysis

First, we'll analyze the performance of each algorithm in terms of the number of bins it used, the amount of time it took to run, and the distribution of the bin sizes. Our results appear to hold regardless of the distribution of the item sizes.

1.4.1 Number of Bins

In each of the cases, we see that the sorting algorithms will use roughly the same number of bins and will outperform the algorithms that don't sort. Between the two non-sorted algorithms, best fit performs better, returning a smaller number of bins required.

1.4.2 Speed

We see that first fit takes the least amount of time. Either sorted first or best fit is after that. Sorted best fit, across all cases, takes the most amount of time.

This makes sense: while first fit stops searching after finding one bin that fits, best fit will seek to optimize across all the bins that fit the current item. This early stopping allows first fit to be faster.

1.4.3 Distribution of Bin Sizes

We also checked the distribution of each algorithm and noted that the sorting versions of either algorithm waste less space overall. This is reflected in both the means and medians being higher for those two algorithms.

We believe that the reason for the distribution comes down to the order that the bins are filled. In an example that doesn't sort, it's possible that a big item is placed into a bin with a bunch of smaller items, but then two medium sized items come along later that force us to use another bin

each. However, one could've been put in the same bin as the big item with the other in the same bin as the smaller items. By sorting in decreasing order, this problem is solved.