In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:

```
# Load Data
insertion = np.genfromtxt('insertion.csv', delimiter=',')
merge = np.genfromtxt('merge.csv', delimiter=',')
shell1 = np.genfromtxt('shell1.csv', delimiter=',')
shell2 = np.genfromtxt('shell2.csv', delimiter=',')
shell3 = np.genfromtxt('shell3.csv', delimiter=',')
shell4 = np.genfromtxt('shell4.csv', delimiter=',')
hybrid1 = np.genfromtxt('hybrid1.csv', delimiter=',')
hybrid2 = np.genfromtxt('hybrid2.csv', delimiter=',')
hybrid3 = np.genfromtxt('hybrid3.csv', delimiter=',')
```

In [3]:

```
# The first tenth lines of the data are for uniform distributed permutations,
# the second then lines are for almost-sorted permutations, and the last ten
# lines are for reverse sorted permuations.Therefore, using these masks to
# obtain these data
uniform_mask = np.arange(0,10)
almost_sorted_mask = np.arange(10, 20)
reverse_sorted_mask = np.arange(20, 30)
```

In [4]:

```
# Print out one set of data for example
print(insertion[uniform_mask])
```

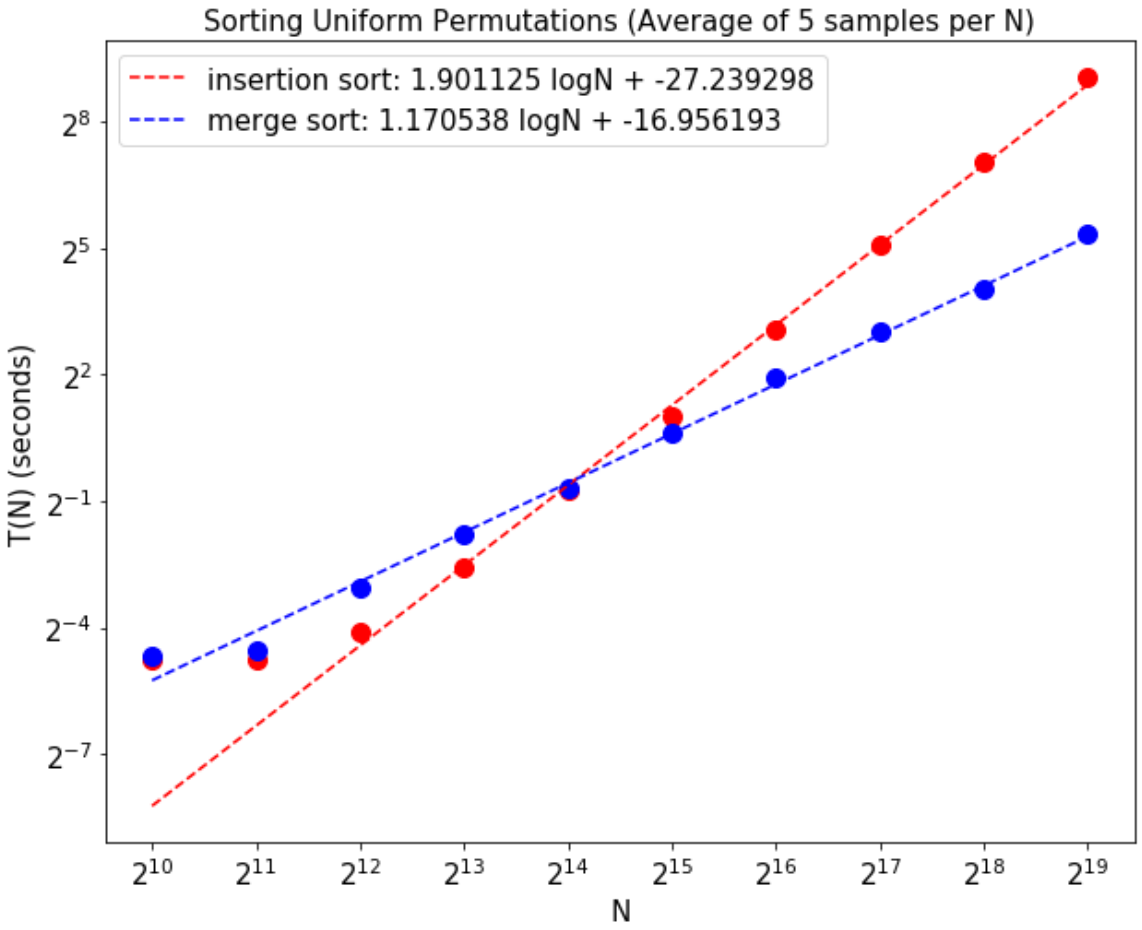
```
[[1.02400e+03 3.70000e-02]
 [2.04800e+03 3.70000e-02]
 [4.09600e+03 5.80000e-02]
 [8.19200e+03 1.69000e-01]
 [1.63840e+04 5.93000e-01]
 [3.27680e+04 2.02700e+00]
 [6.55360e+04 8.41300e+00]
 [1.31072e+05 3.33690e+01]
 [2.62144e+05 1.31413e+02]
 [5.24288e+05 5.34231e+02]]
```

The first colume indicates the sizes of the input data, which are 2^n where n=10....19. The second colume indicates the running times of the sorting algorithm. These runing times are the results of the average of 5 runs of the sorting algortihm with the input size.

In [5]:

```
# Plotting Sorting Uniform Permutations (Insertion Sort, Merge Sort)
f, ax = plt.subplots(1,1, figsize=(10,8))


m1, b1 = np.polyfit(np.log2(insertion[uniform_mask][2:,0]), np.log2(insertion[uniform_ma
sk][2:,1]), 1)
m2, b2 = np.polyfit(np.log2(merge[uniform_mask][:,0]), np.log2(merge[uniform_mask][:,1])
, 1)


plt.loglog(insertion[uniform_mask][:,0], insertion[uniform_mask][:,1], '.', basex=2, bas
ey=2,
           markersize=20, color='red')
plt.loglog(merge[uniform_mask][:,0], merge[uniform_mask][:,1], '.', basex=2, basey=2,
           markersize=20, color='blue')

plt.loglog(insertion[uniform_mask][:,0], 2**(m1*np.log2(insertion[uniform_mask][:,0])+b1
```

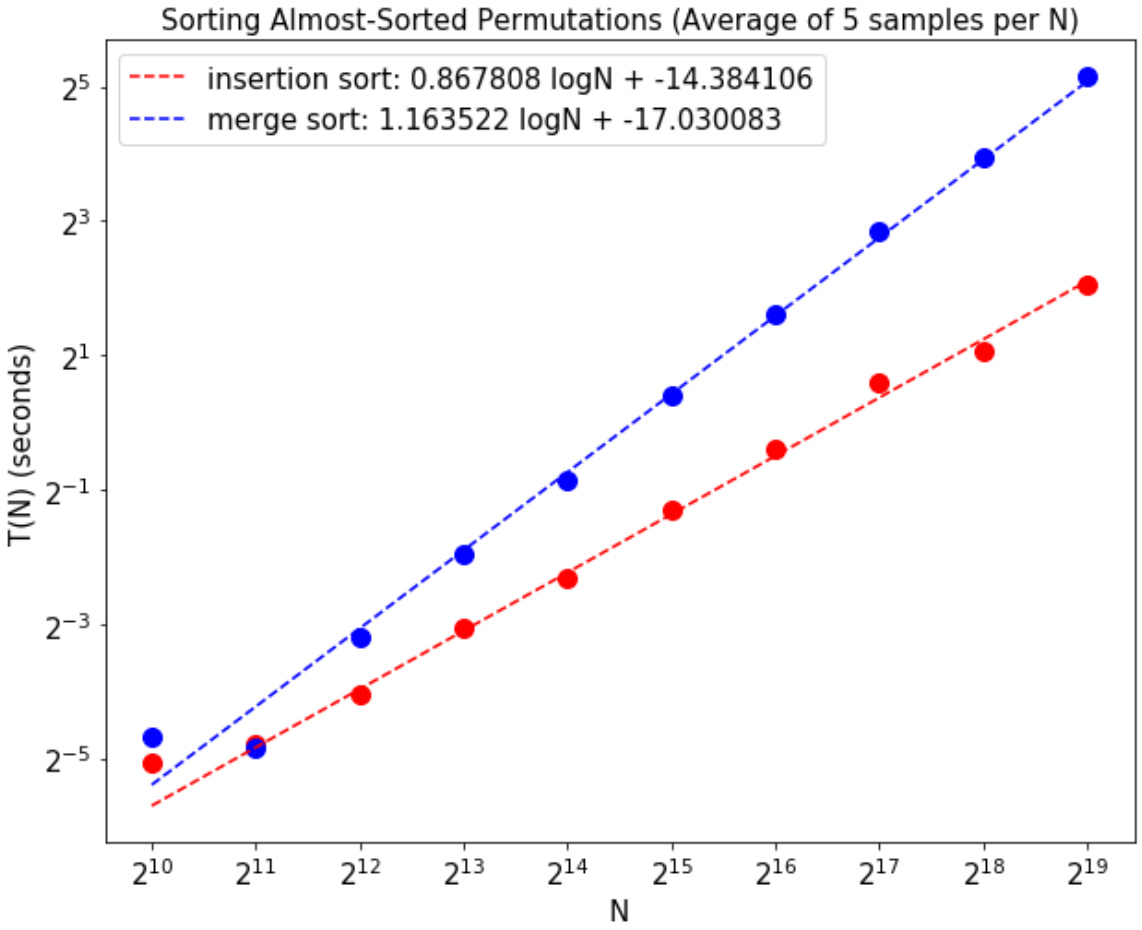
```
), basex=2, basey=2,
          linestyle='--', label=f'insertion sort: {m1:4f} logN + {b1:4f}', color='red')
plt.loglog(merge[uniform_mask][:,0], 2**(m2*np.log2(merge[uniform_mask][:,0])+b2), basex
=2, basey=2,
          linestyle='--', label=f'merge sort: {m2:4f} logN + {b2:4f}', color='blue')

plt.xlabel('N', fontsize=15)
plt.ylabel('T(N) (seconds)', fontsize=15)
plt.xticks(insertion[uniform_mask][:,0], fontsize=15)
plt.yticks(fontsize=15)
plt.title('Sorting Uniform Permutations (Average of 5 samples per N)', fontsize=15)
plt.legend(fontsize=15)
plt.show()
```



In [6]:

```
# Plotting Sorting Almost-sorted Permutations (Insertion Sort, Merge Sort)
f, ax = plt.subplots(1,1, figsize=(10,8))


m1, b1 = np.polyfit(np.log2(insertion[almost_sorted_mask][2:,0]), np.log2(insertion[almo
st_sorted_mask][2:,1]), 1)
m2, b2 = np.polyfit(np.log2(merge[almost_sorted_mask][:,0]), np.log2(merge[almost_sorted
_mask][:,1]), 1)


plt.loglog(insertion[almost_sorted_mask][:,0], insertion[almost_sorted_mask][:,1], '.', b
asex=2, basey=2,
           markersize=20, color='red')
plt.loglog(merge[almost_sorted_mask][:,0], merge[almost_sorted_mask][:,1], '.', basex=2,
basey=2,
           markersize=20, color='blue')

plt.loglog(insertion[almost_sorted_mask][:,0], 2**(m1*np.log2(insertion[almost_sorted_ma
sk][:,0])+b1), basex=2, basey=2,
          linestyle='--', label=f'insertion sort: {m1:4f} logN + {b1:4f}', color='red')
plt.loglog(merge[almost_sorted_mask][:,0], 2**(m2*np.log2(merge[almost_sorted_mask][:,0]
)+b2), basex=2, basey=2,
          linestyle='--', label=f'merge sort: {m2:4f} logN + {b2:4f}', color='blue')
```

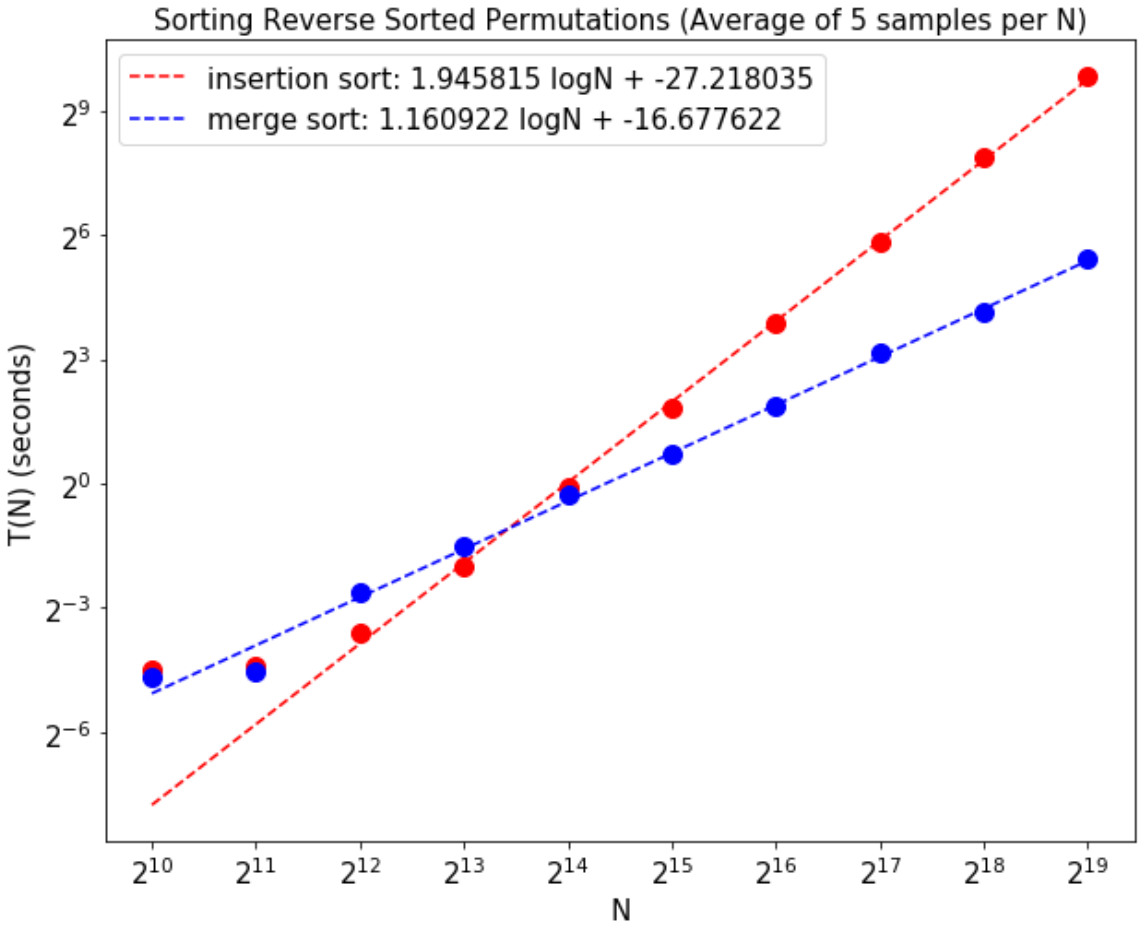
```
plt.xlabel('N', fontsize=15)
plt.ylabel('T(N) (seconds)', fontsize=15)
plt.xticks(insertion[almost_sorted_mask][:,0], fontsize=15)
plt.yticks(fontsize=15)
plt.title('Sorting Almost-Sorted Permutations (Average of 5 samples per N)', fontsize=15)
plt.legend(fontsize=15)
plt.show()
```



In [7]:

```
# Plotting Sorting Almost-sorted Permutations (Insertion Sort, Merge Sort)
f, ax = plt.subplots(1,1, figsize=(10,8))


m1, b1 = np.polyfit(np.log2(insertion[reverse_sorted_mask][2:,0]), np.log2(insertion[rev
erse_sorted_mask][2:,1]), 1)
m2, b2 = np.polyfit(np.log2(merge[reverse_sorted_mask][:,0]), np.log2(merge[reverse_sort
ed_mask][:,1]), 1)


plt.loglog(insertion[reverse_sorted_mask][:,0], insertion[reverse_sorted_mask][:,1], '.',
basex=2, basey=2,
           markersize=20, color='red')
plt.loglog(merge[reverse_sorted_mask][:,0], merge[reverse_sorted_mask][:,1], '.', basex=2
, basey=2,
           markersize=20, color='blue')

plt.loglog(insertion[reverse_sorted_mask][:,0], 2**(m1*np.log2(insertion[reverse_sorted_
mask][:,0])+b1), basex=2, basey=2,
          linestyle='--', label=f'insertion sort: {m1:4f} logN + {b1:4f}', color='red')
plt.loglog(merge[reverse_sorted_mask][:,0], 2**(m2*np.log2(merge[reverse_sorted_mask][:,
0])+b2), basex=2, basey=2,
          linestyle='--', label=f'merge sort: {m2:4f} logN + {b2:4f}', color='blue')

plt.xlabel('N', fontsize=15)
plt.ylabel('T(N) (seconds)', fontsize=15)
plt.xticks(insertion[reverse_sorted_mask][:,0], fontsize=15)
plt.yticks(fontsize=15)
plt.title('Sorting Reverse Sorted Permutations (Average of 5 samples per N)', fontsize=15
)
```

```
plt.legend(fontsize=15)
plt.show()
```



Comments:

First of all, I did not use the data points with the size of 2^10 and 2^11 for the best-fit linear regression of insertion sort because when the size of the data is small, the result cannot be fitted with a line. From the plot, we can see that insertion sort will be faster algorithm comparing to merge sort when the input size is relatively small. As the input size goes bigger, merge sort will overperform insertion sort, which means merge sort will be faster than insertion sort for large input size.

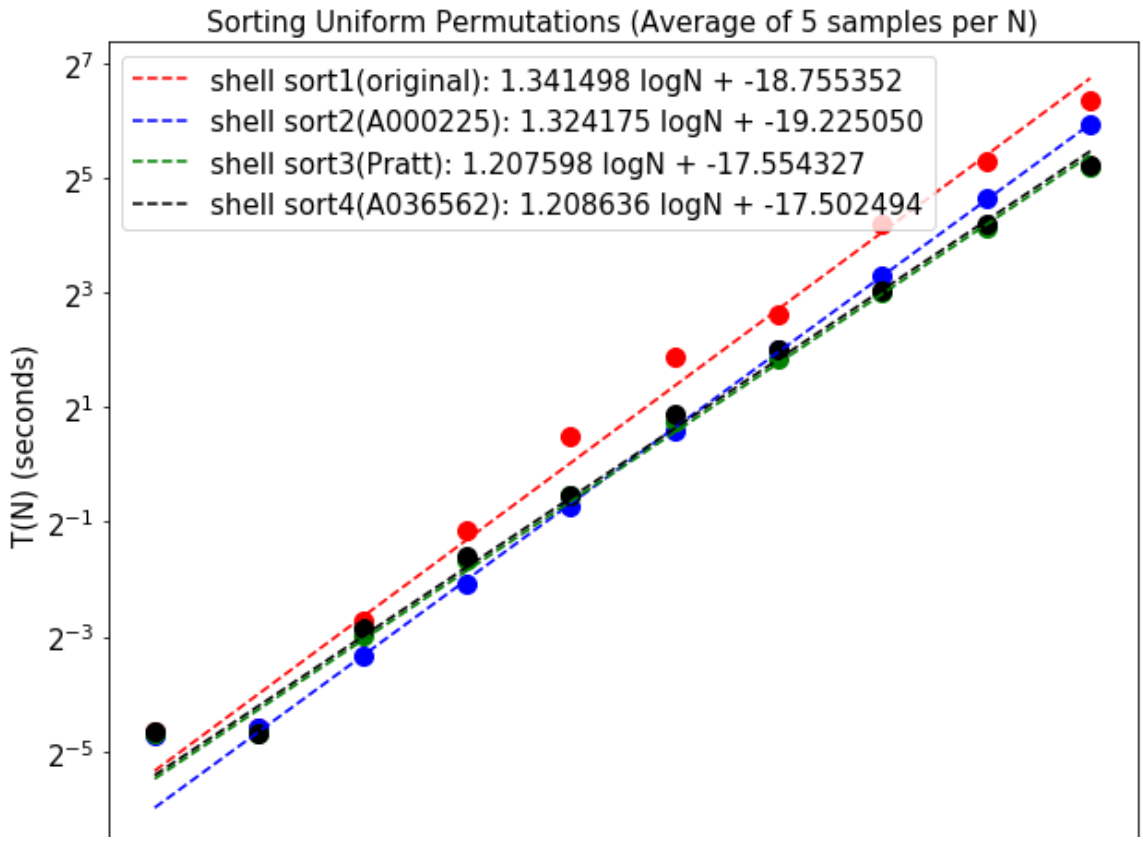
From the slopes of the asymptotic best-fit lines, we can see that with Uniform Permutations(average case) and Reverse Sorted Permutations(worst case), insertion sort has a slope close to 2. This means the running time is growing quadratically with the growing input size, and the average and worst case time complexity of insertion sort are O(N^2). However, with Almost Sorted Permutations, insertion sort has a slope close to 1. This means the running time is growing linearly with the growing input size, and the time complexity of insertion sort in this case is O(N). This is the best case for insertion sort. In this case my insertion sort even overperforms my merge sort.

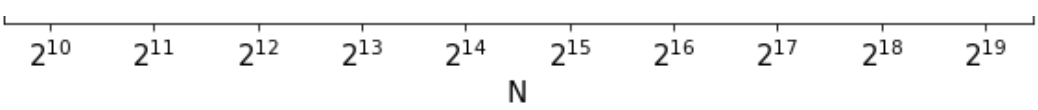
For merge sort, we can see that no matter which permutation we are using, the slopes of the asymptotic best-fit lines are always about 1.16 or 1.17. This means the running time of merge sort is always a little bit slower than linear running time which should have a slope of 1. This is reasonable since we knnow that the time complexity of merge sort is O(NlogN). This also concludes that different permutations of the input number have no impact on the running time of merge sort.

In [8]:

```
# Plotting Sorting Uniform Permutations (Shell Sort)
f, ax = plt.subplots(1,1, figsize=(10,8))
```

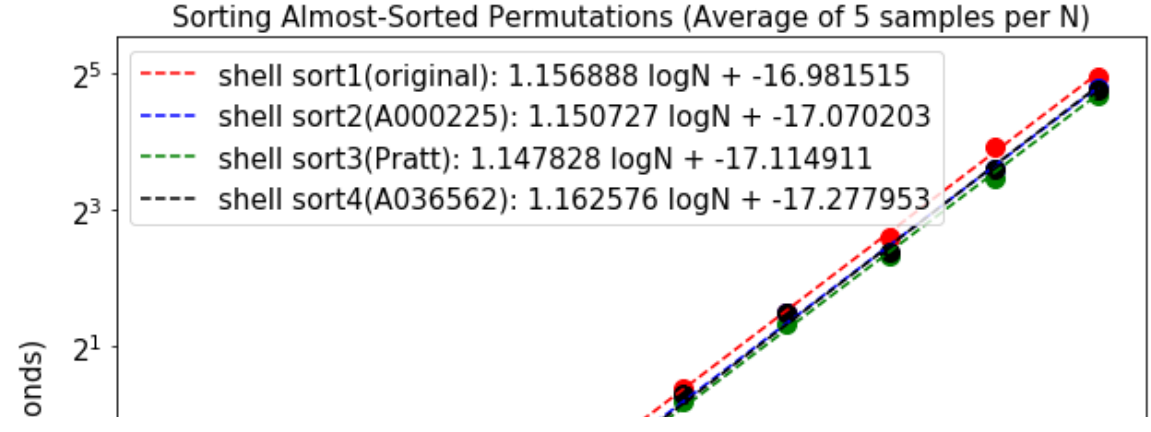
```
m1, b1 = np.polyfit(np.log2(shell1[uniform_mask][1:,0]), np.log2(shell1[uniform_mask][1:
,1]), 1)
m2, b2 = np.polyfit(np.log2(shell2[uniform_mask][1:,0]), np.log2(shell2[uniform_mask][1:
,1]), 1)
m3, b3 = np.polyfit(np.log2(shell3[uniform_mask][1:,0]), np.log2(shell3[uniform_mask][1:
,1]), 1)
m4, b4 = np.polyfit(np.log2(shell4[uniform_mask][1:,0]), np.log2(shell4[uniform_mask][1:
,1]), 1)


plt.loglog(shell1[uniform_mask][:,0], shell1[uniform_mask][:,1], '.', basex=2, basey=2,
           markersize=20, color='red')
plt.loglog(shell2[uniform_mask][:,0], shell2[uniform_mask][:,1], '.', basex=2, basey=2,
           markersize=20, color='blue')
plt.loglog(shell3[uniform_mask][:,0], shell3[uniform_mask][:,1], '.', basex=2, basey=2,
           markersize=20, color='green')
plt.loglog(shell4[uniform_mask][:,0], shell4[uniform_mask][:,1], '.', basex=2, basey=2,
           markersize=20, color='black')

plt.loglog(shell1[uniform_mask][:,0], 2**(m1*np.log2(shell1[uniform_mask][:,0])+b1), bas
ex=2, basey=2,
          linestyle='--', label=f'shell sort1(original): {m1:4f} logN + {b1:4f}', color=
'red')
plt.loglog(shell2[uniform_mask][:,0], 2**(m2*np.log2(shell2[uniform_mask][:,0])+b2), bas
ex=2, basey=2,
          linestyle='--', label=f'shell sort2(A000225): {m2:4f} logN + {b2:4f}', color='
blue')
plt.loglog(shell3[uniform_mask][:,0], 2**(m3*np.log2(shell3[uniform_mask][:,0])+b3), bas
ex=2, basey=2,
          linestyle='--', label=f'shell sort3(Pratt): {m3:4f} logN + {b3:4f}', color='gr
een')
plt.loglog(shell4[uniform_mask][:,0], 2**(m4*np.log2(shell4[uniform_mask][:,0])+b4), bas
ex=2, basey=2,
          linestyle='--', label=f'shell sort4(A036562): {m4:4f} logN + {b4:4f}', color='
black')

plt.xlabel('N', fontsize=15)
plt.ylabel('T(N) (seconds)', fontsize=15)
plt.xticks(shell1[uniform_mask][:,0], fontsize=15)
plt.yticks(fontsize=15)
plt.title('Sorting Uniform Permutations (Average of 5 samples per N)', fontsize=15)
plt.legend(fontsize=15)
plt.show()
```

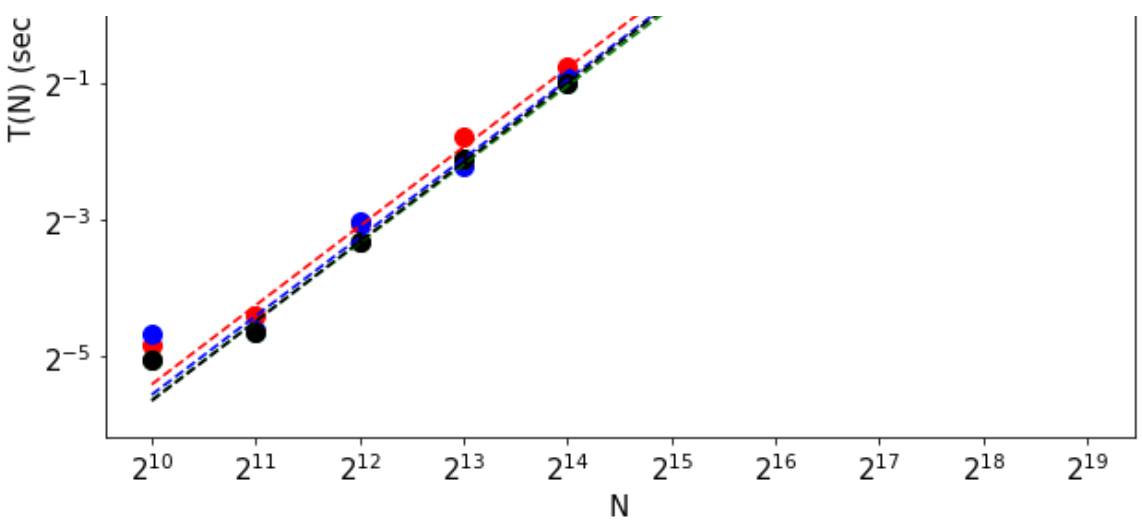




In [9]:

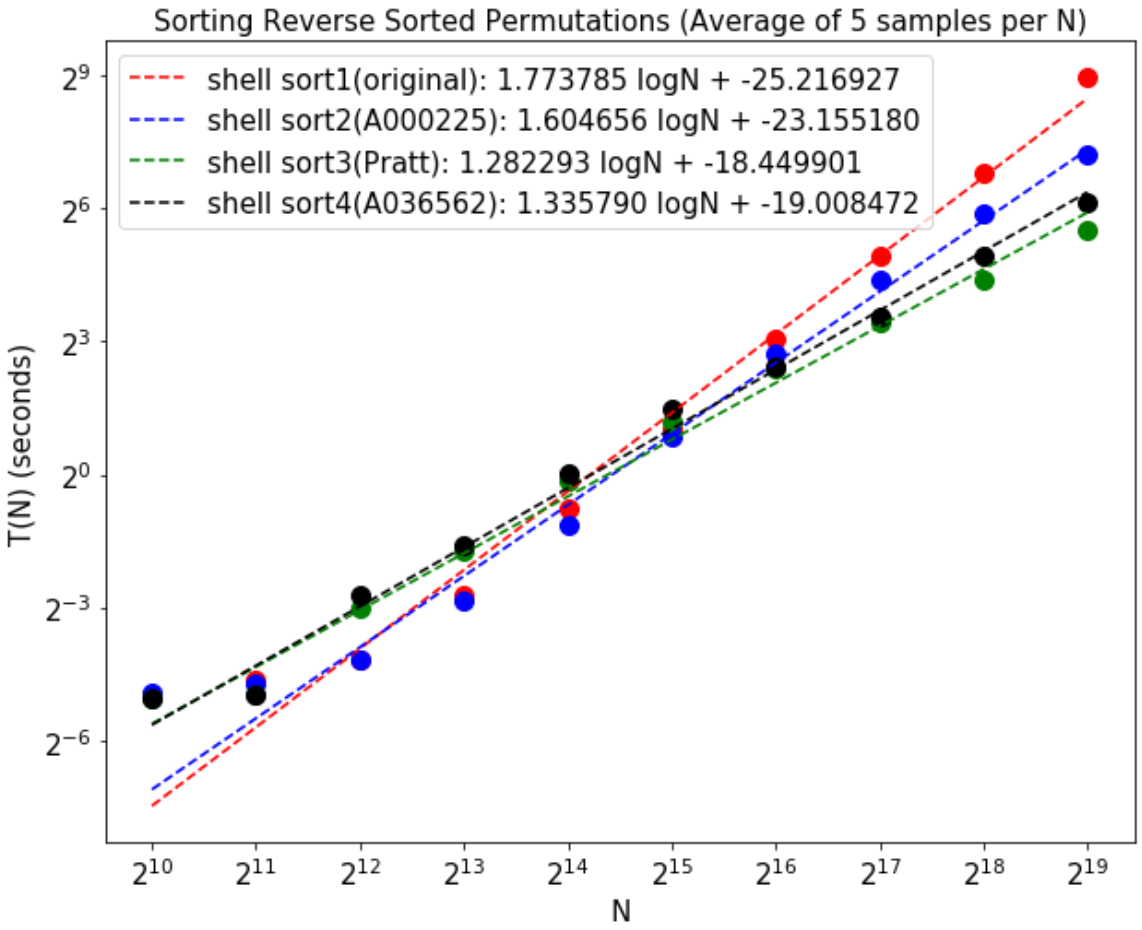
```
# Plotting Sorting Almost-sorted Permutations (Shell Sort)
f, ax = plt.subplots(1,1, figsize=(10,8))


m1, b1 = np.polyfit(np.log2(shell1[almost_sorted_mask][1:,0]), np.log2(shell1[almost_sor
ted_mask][1:,1]), 1)
m2, b2 = np.polyfit(np.log2(shell2[almost_sorted_mask][1:,0]), np.log2(shell2[almost_sor
ted_mask][1:,1]), 1)
m3, b3 = np.polyfit(np.log2(shell3[almost_sorted_mask][1:,0]), np.log2(shell3[almost_sor
ted_mask][1:,1]), 1)
m4, b4 = np.polyfit(np.log2(shell4[almost_sorted_mask][1:,0]), np.log2(shell4[almost_sor
ted_mask][1:,1]), 1)


plt.loglog(shell1[almost_sorted_mask][:,0], shell1[almost_sorted_mask][:,1], '.', basex=2
, basey=2,
           markersize=20, color='red')
plt.loglog(shell2[almost_sorted_mask][:,0], shell2[almost_sorted_mask][:,1], '.', basex=2
, basey=2,
           markersize=20, color='blue')
plt.loglog(shell3[almost_sorted_mask][:,0], shell3[almost_sorted_mask][:,1], '.', basex=2
, basey=2,
           markersize=20, color='green')
plt.loglog(shell4[almost_sorted_mask][:,0], shell4[almost_sorted_mask][:,1], '.', basex=2
, basey=2,
           markersize=20, color='black')

plt.loglog(shell1[almost_sorted_mask][:,0], 2**(m1*np.log2(shell1[almost_sorted_mask][:,
0])+b1),
           basex=2, basey=2, linestyle='--', label=f'shell sort1(original): {m1:4f} logN
+ {b1:4f}', color='red')
plt.loglog(shell2[almost_sorted_mask][:,0], 2**(m2*np.log2(shell2[almost_sorted_mask][:,
0])+b2),
           basex=2, basey=2, linestyle='--', label=f'shell sort2(A000225): {m2:4f} logN
+ {b2:4f}', color='blue')
plt.loglog(shell3[almost_sorted_mask][:,0], 2**(m3*np.log2(shell3[almost_sorted_mask][:,
0])+b3),
           basex=2, basey=2, linestyle='--', label=f'shell sort3(Pratt): {m3:4f} logN +
{b3:4f}', color='green')
plt.loglog(shell4[almost_sorted_mask][:,0], 2**(m4*np.log2(shell4[almost_sorted_mask][:,
0])+b4),
           basex=2, basey=2, linestyle='--', label=f'shell sort4(A036562): {m4:4f} logN
+ {b4:4f}', color='black')

plt.xlabel('N', fontsize=15)
plt.ylabel('T(N) (seconds)', fontsize=15)
plt.xticks(shell1[uniform_mask][:,0], fontsize=15)
plt.yticks(fontsize=15)
plt.title('Sorting Almost-Sorted Permutations (Average of 5 samples per N)', fontsize=15)
plt.legend(fontsize=15)
plt.show()
```





In [10]:

```
# Plotting Sorting Reverse sorted Permutations (Shell Sort)
f, ax = plt.subplots(1,1, figsize=(10,8))


m1, b1 = np.polyfit(np.log2(shell1[reverse_sorted_mask][1:,0]), np.log2(shell1[reverse_s
orted_mask][1:,1]), 1)
m2, b2 = np.polyfit(np.log2(shell2[reverse_sorted_mask][1:,0]), np.log2(shell2[reverse_s
orted_mask][1:,1]), 1)
m3, b3 = np.polyfit(np.log2(shell3[reverse_sorted_mask][1:,0]), np.log2(shell3[reverse_s
orted_mask][1:,1]), 1)
m4, b4 = np.polyfit(np.log2(shell4[reverse_sorted_mask][1:,0]), np.log2(shell4[reverse_s
orted_mask][1:,1]), 1)


plt.loglog(shell1[reverse_sorted_mask][:,0], shell1[reverse_sorted_mask][:,1], '.', basex
=2, basey=2,
           markersize=20, color='red')
plt.loglog(shell2[reverse_sorted_mask][:,0], shell2[reverse_sorted_mask][:,1], '.', basex
=2, basey=2,
           markersize=20, color='blue')
plt.loglog(shell3[reverse_sorted_mask][:,0], shell3[reverse_sorted_mask][:,1], '.', basex
=2, basey=2,
           markersize=20, color='green')
plt.loglog(shell4[reverse_sorted_mask][:,0], shell4[reverse_sorted_mask][:,1], '.', basex
=2, basey=2,
           markersize=20, color='black')

plt.loglog(shell1[reverse_sorted_mask][:,0], 2**(m1*np.log2(shell1[reverse_sorted_mask][
:,0])+b1),
           basex=2, basey=2, linestyle='--', label=f'shell sort1(original): {m1:4f} logN
+ {b1:4f}', color='red')
plt.loglog(shell2[reverse_sorted_mask][:,0], 2**(m2*np.log2(shell2[reverse_sorted_mask][
:,0])+b2),
           basex=2, basey=2, linestyle='--', label=f'shell sort2(A000225): {m2:4f} logN
+ {b2:4f}', color='blue')
plt.loglog(shell3[reverse_sorted_mask][:,0], 2**(m3*np.log2(shell3[reverse_sorted_mask][
:,0])+b3),
           basex=2, basey=2, linestyle='--', label=f'shell sort3(Pratt): {m3:4f} logN +
{b3:4f}', color='green')
plt.loglog(shell4[reverse_sorted_mask][:,0], 2**(m4*np.log2(shell4[reverse_sorted_mask][
:,0])+b4),
           basex=2, basey=2, linestyle='--', label=f'shell sort4(A036562): {m4:4f} logN
+ {b4:4f}', color='black')

plt.xlabel('N', fontsize=15)
plt.ylabel('T(N) (seconds)', fontsize=15)
plt.xticks(shell1[uniform_mask][:,0], fontsize=15)
plt.yticks(fontsize=15)
plt.title('Sorting Reverse Sorted Permutations (Average of 5 samples per N)', fontsize=15
)
plt.legend(fontsize=15)
plt.show()
```



Comments:

Similar to insertion sort, I ignore data points with the size of 2^10 and 2^11 for the best-fit linear regression while ploting the linear regression line for an asymptotic best-fit line.

As results, we can see that for Uniformly Distributed Permutations(average case), different shell sort algorithms have similar asymtotic slopes, while the slopes of shellsort3(Pratt) and shellsort4(A036562) are slightly smaller than the other two(Original, A000225). The slopes are about 1.2 to 1.35. This means the average time complexity of shell sort is about O(N^(1.2 to 1.35)). Pratt and A036562 shell sort algorithms are slightly faster than the other two for larger input size in average case.

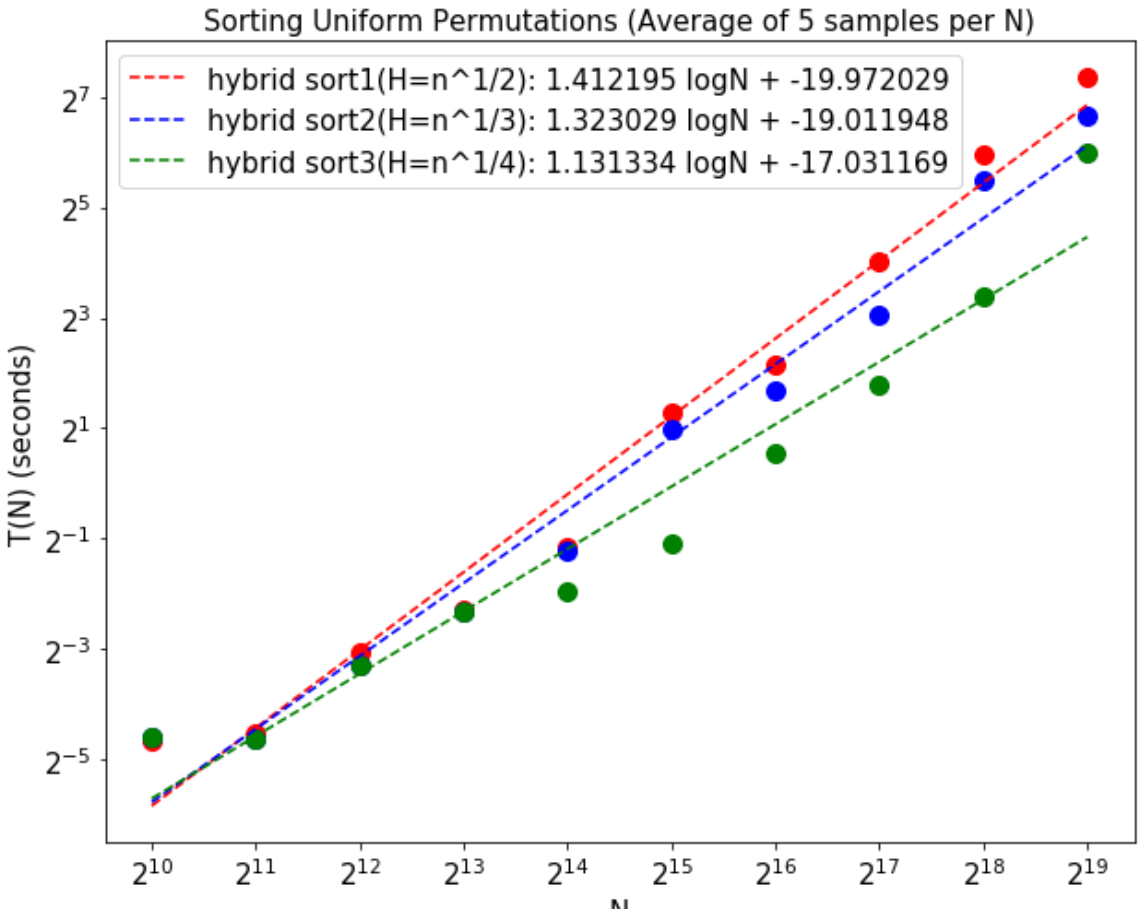
For Almost-sorted Permutations, these shell sort algorithms also have smilar asymtotic slopes. Comparing to Uniformly Distributed Permutations, these algorithms have a little bit smaller asymtotic slopes in this case. This means the running time of these shell sort is faster for Almost-sorted Permutations for large input size. This is reasonable because shell sort is based on insertion sort. The time complexity for all shell sort algorithms are about O(N^1.15) in this case.

For Reverse Sorted Permutations, we can see the different asymtotic slopes for different shell sort algorithms. With a large input size, the order of the running times of these algorithm is: shell sort1 > shell sort2> shell sort4> shell sort3 (from slowest algorithm to fastest algorithm). The shell sort algorithm using Pratt sequence overperforms the other algorithms with time complexity of O(N^1.28). The original shell sort algorithm has the worse time complexity of O(N^1.77) which is close to O(N^2). The shell sort algorithm using A000225 sequence has the time complexity of O(N^1.6) which is close to O(N^3/2), while the one using A036562 sequence has a better time complexity of O(N^1.336) which is close to O(N^4/3)

In [11]:

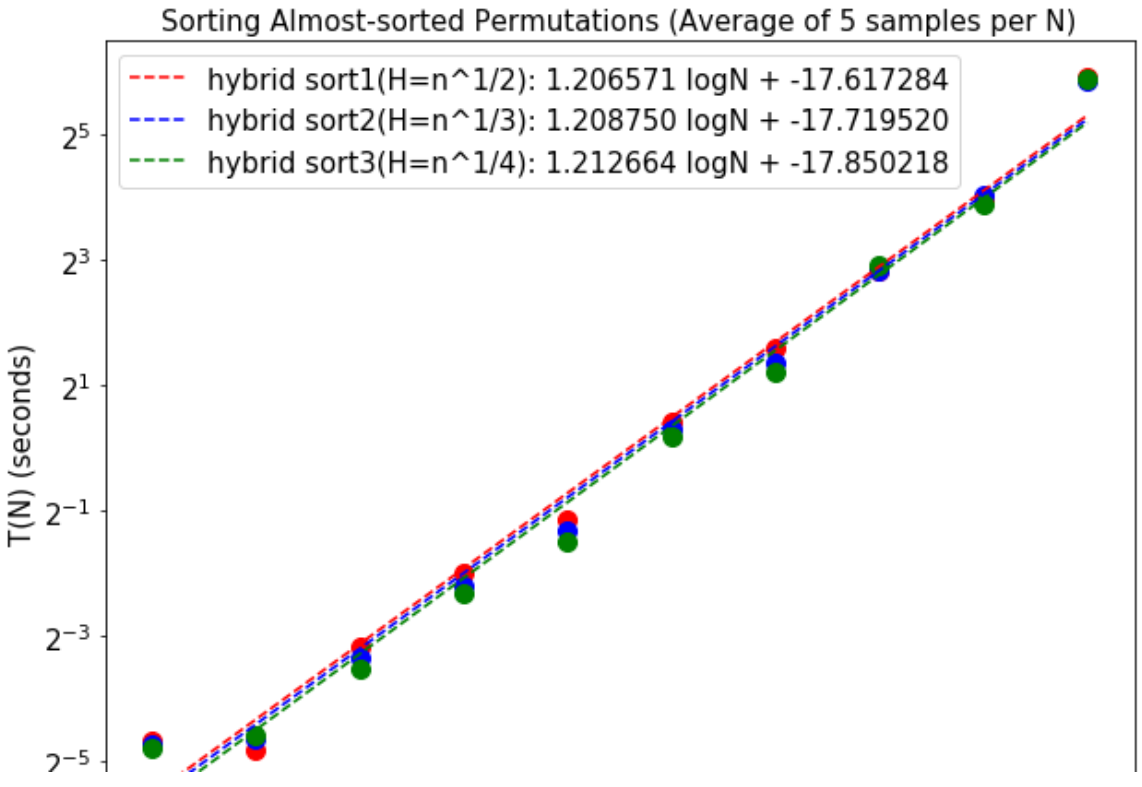
```
# Plotting Sorting Uniform Permutations (Hybrid Sort)
```

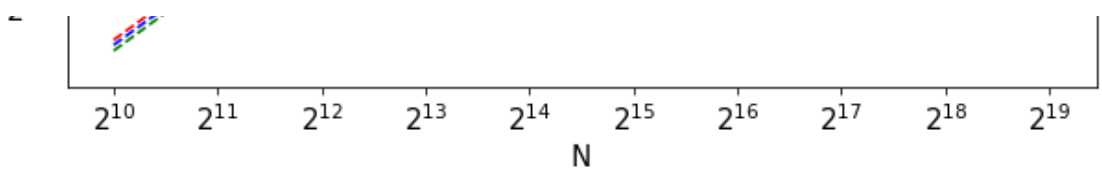
```
# Plotting Sorting Uniform Permutations (Hybrid Sort)
f, ax = plt.subplots(1,1, figsize=(10,8))


m1, b1 = np.polyfit(np.log2(hybrid1[uniform_mask][:,0]), np.log2(hybrid1[uniform_mask][:
,1]), 1)
m2, b2 = np.polyfit(np.log2(hybrid2[uniform_mask][:,0]), np.log2(hybrid2[uniform_mask][:
,1]), 1)
m3, b3 = np.polyfit(np.log2(hybrid3[uniform_mask][:,0]), np.log2(hybrid3[uniform_mask][:
,1]), 1)


plt.loglog(hybrid1[uniform_mask][:,0], hybrid1[uniform_mask][:,1], '.', basex=2, basey=2
,
           markersize=20, color='red')
plt.loglog(hybrid2[uniform_mask][:,0], hybrid2[uniform_mask][:,1], '.', basex=2, basey=2
,
           markersize=20, color='blue')
plt.loglog(hybrid3[uniform_mask][:,0], hybrid3[uniform_mask][:,1], '.', basex=2, basey=2
,
           markersize=20, color='green')

plt.loglog(hybrid1[uniform_mask][:,0], 2**(m1*np.log2(hybrid1[uniform_mask][:,0])+b1), b
asex=2, basey=2,
         linestyle='--', label=f'hybrid sort1(H=n^1/2): {m1:4f} logN + {b1:4f}', color=
'red')
plt.loglog(hybrid2[uniform_mask][:,0], 2**(m2*np.log2(hybrid2[uniform_mask][:,0])+b2), b
asex=2, basey=2,
         linestyle='--', label=f'hybrid sort2(H=n^1/3): {m2:4f} logN + {b2:4f}', color=
'blue')
plt.loglog(hybrid3[uniform_mask][:,0], 2**(m3*np.log2(hybrid3[uniform_mask][:,0])+b3), b
asex=2, basey=2,
         linestyle='--', label=f'hybrid sort3(H=n^1/4): {m3:4f} logN + {b3:4f}', color=
'green')

plt.xlabel('N', fontsize=15)
plt.ylabel('T(N) (seconds)', fontsize=15)
plt.xticks(hybrid1[uniform_mask][:,0], fontsize=15)
plt.yticks(fontsize=15)
plt.title('Sorting Uniform Permutations (Average of 5 samples per N)', fontsize=15)
plt.legend(fontsize=15)
plt.show()
```



In [12]:

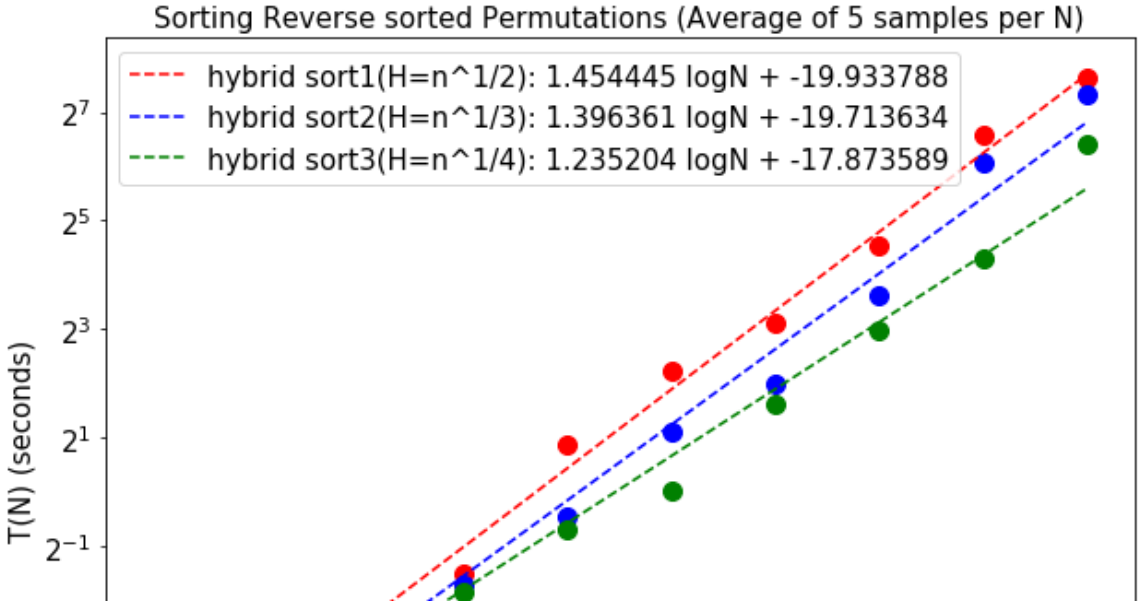
```
# Plotting Sorting Almost-sorted Permutations (Hybrid Sort)
f, ax = plt.subplots(1,1, figsize=(10,8))


m1, b1 = np.polyfit(np.log2(hybrid1[almost_sorted_mask][:,0]), np.log2(hybrid1[almost_so
rted_mask][:,1]), 1)
m2, b2 = np.polyfit(np.log2(hybrid2[almost_sorted_mask][:,0]), np.log2(hybrid2[almost_so
rted_mask][:,1]), 1)
m3, b3 = np.polyfit(np.log2(hybrid3[almost_sorted_mask][:,0]), np.log2(hybrid3[almost_so
rted_mask][:,1]), 1)


plt.loglog(hybrid1[almost_sorted_mask][:,0], hybrid1[almost_sorted_mask][:,1], '.', basex
=2, basey=2,
           markersize=20, color='red')
plt.loglog(hybrid2[almost_sorted_mask][:,0], hybrid2[almost_sorted_mask][:,1], '.', basex
=2, basey=2,
           markersize=20, color='blue')
plt.loglog(hybrid3[almost_sorted_mask][:,0], hybrid3[almost_sorted_mask][:,1], '.', basex
=2, basey=2,
           markersize=20, color='green')

plt.loglog(hybrid1[almost_sorted_mask][:,0], 2**(m1*np.log2(hybrid1[almost_sorted_mask][
:,0])+b1), basex=2, basey=2,
          linestyle='--', label=f'hybrid sort1(H=n^1/2): {m1:4f} logN + {b1:4f}', color=
'red')
plt.loglog(hybrid2[almost_sorted_mask][:,0], 2**(m2*np.log2(hybrid2[almost_sorted_mask][
:,0])+b2), basex=2, basey=2,
          linestyle='--', label=f'hybrid sort2(H=n^1/3): {m2:4f} logN + {b2:4f}', color=
'blue')
plt.loglog(hybrid3[almost_sorted_mask][:,0], 2**(m3*np.log2(hybrid3[almost_sorted_mask][
:,0])+b3), basex=2, basey=2,
          linestyle='--', label=f'hybrid sort3(H=n^1/4): {m3:4f} logN + {b3:4f}', color=
'green')

plt.xlabel('N', fontsize=15)
plt.ylabel('T(N) (seconds)', fontsize=15)
plt.xticks(hybrid1[almost_sorted_mask][:,0], fontsize=15)
plt.yticks(fontsize=15)
plt.title('Sorting Almost-sorted Permutations (Average of 5 samples per N)', fontsize=15)
plt.legend(fontsize=15)
plt.show()
```

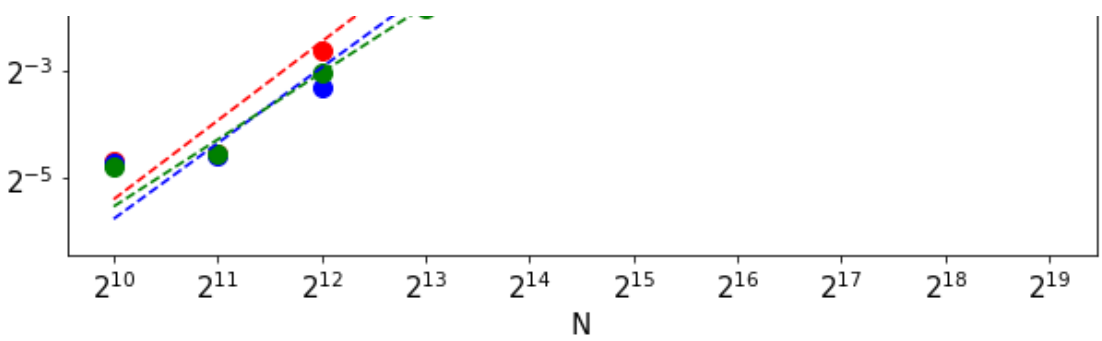




In [13]:

```
# Plotting Sorting Reverse sorted Permutations (Hybrid Sort)
f, ax = plt.subplots(1,1, figsize=(10,8))


m1, b1 = np.polyfit(np.log2(hybrid1[reverse_sorted_mask][:,0]), np.log2(hybrid1[reverse_
sorted_mask][:,1]), 1)
m2, b2 = np.polyfit(np.log2(hybrid2[reverse_sorted_mask][:,0]), np.log2(hybrid2[reverse_
sorted_mask][:,1]), 1)
m3, b3 = np.polyfit(np.log2(hybrid3[reverse_sorted_mask][:,0]), np.log2(hybrid3[reverse_
sorted_mask][:,1]), 1)


plt.loglog(hybrid1[reverse_sorted_mask][:,0], hybrid1[reverse_sorted_mask][:,1], '.', bas
ex=2, basey=2,
           markersize=20, color='red')
plt.loglog(hybrid2[reverse_sorted_mask][:,0], hybrid2[reverse_sorted_mask][:,1], '.', bas
ex=2, basey=2,
           markersize=20, color='blue')
plt.loglog(hybrid3[reverse_sorted_mask][:,0], hybrid3[reverse_sorted_mask][:,1], '.', bas
ex=2, basey=2,
           markersize=20, color='green')

plt.loglog(hybrid1[reverse_sorted_mask][:,0], 2**(m1*np.log2(hybrid1[reverse_sorted_mask
][:,0])+b1), basex=2, basey=2,
          linestyle='--', label=f'hybrid sort1(H=n^1/2): {m1:4f} logN + {b1:4f}', color=
'red')
plt.loglog(hybrid2[reverse_sorted_mask][:,0], 2**(m2*np.log2(hybrid2[reverse_sorted_mask
][:,0])+b2), basex=2, basey=2,
          linestyle='--', label=f'hybrid sort2(H=n^1/3): {m2:4f} logN + {b2:4f}', color=
'blue')
plt.loglog(hybrid3[reverse_sorted_mask][:,0], 2**(m3*np.log2(hybrid3[reverse_sorted_mask
][:,0])+b3), basex=2, basey=2,
          linestyle='--', label=f'hybrid sort3(H=n^1/4): {m3:4f} logN + {b3:4f}', color=
'green')

plt.xlabel('N', fontsize=15)
plt.ylabel('T(N) (seconds)', fontsize=15)
plt.xticks(hybrid1[reverse_sorted_mask][:,0], fontsize=15)
plt.yticks(fontsize=15)
plt.title('Sorting Reverse sorted Permutations (Average of 5 samples per N)', fontsize=15
)
plt.legend(fontsize=15)
plt.show()
```





Comments:

From the plot of different hybrid sort algorithms, we can see that different hybrid sort algorithms have similar running time for Almost-sorted Permutations. For Uniformly Distributed Permutations(average case) and Reverse sorted Permutations(worst case), the ones with n to the smaller fraction overperforms the ones with n to the larger fraction. This is because insertion sort is only faster than merge sort when the input size is small. However, when H is large, this means we will switch from merge sort to insertion sort when the size of the array is still relatively big. This will slow down the process when the input size is large because we are doing lots of insertion sort on large array. We can see that for hybrid sort1, the data points with input size from 2^15 to 2^19 have large running time. This is caused by the reason above. This does not affect the running time for Almost-sorted Permutations because insertion sort has a great performance on Almost-sorted Permutations. The performance of insertions sort on Almost-sorted Permutations is even better than merge sort sometimes. Therefore, we can see that hybrid sort3 is actually overperformed by the other two hybrid sort. Based on the asymptotic slope of these algorithms, the time complexity of hybrid sort is about O(N^1.2 to 1.4). The time complexity would be better if we pick the appropriate H.

Conclusion:

Overall, among all these sorting algorithms, I would say hybrid sort3 is the best. This is because hybrid sort algorithm takes the advantages of both merge sort and insertion sort. If we carefully pick the value for H, the running time of the sorting algorithm will be fast. Among these algorithms, hybrid sort3 picks the best H for our input size. Therefore, I would pick hybrid sort3 as the best sorting algorithm in our experiment. However, if we know that the input array is almost sorted, I would say insertion sort would be the best algorithm. Insertion sort overperforms all of our experimental algorithms when the input array is almost sorted. Shell sort is a better version of insertion sort when the array is not almost sorted. It will never be worse than O(N^2). It has a time complexity of O(N^1.2 to 1.7) depending on the sequence we choose. The best shell sort in our experiment would be the one using Pratt sequence.