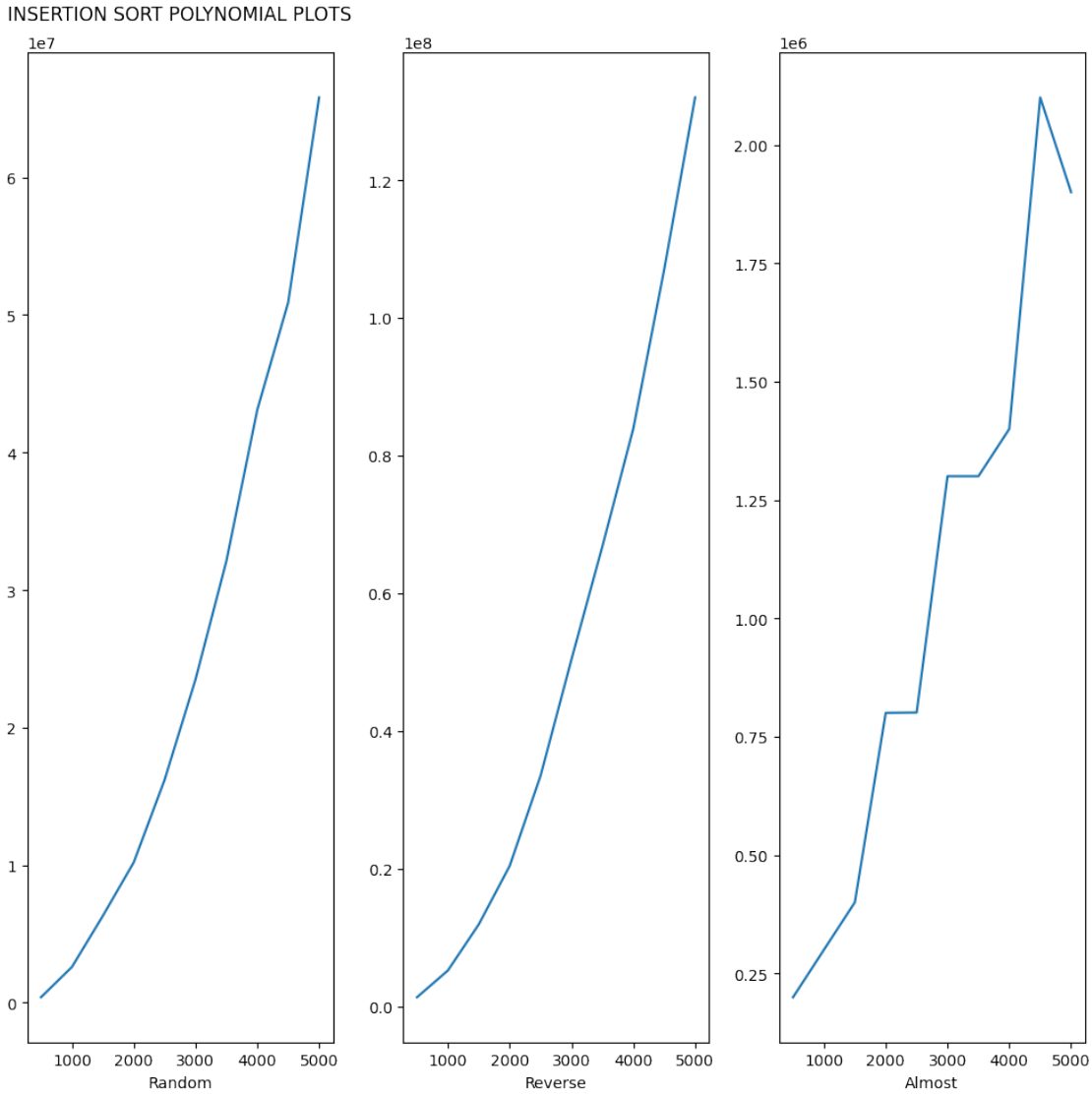
Project 1 Report - Adarsh B Shankar

April 20, 2021

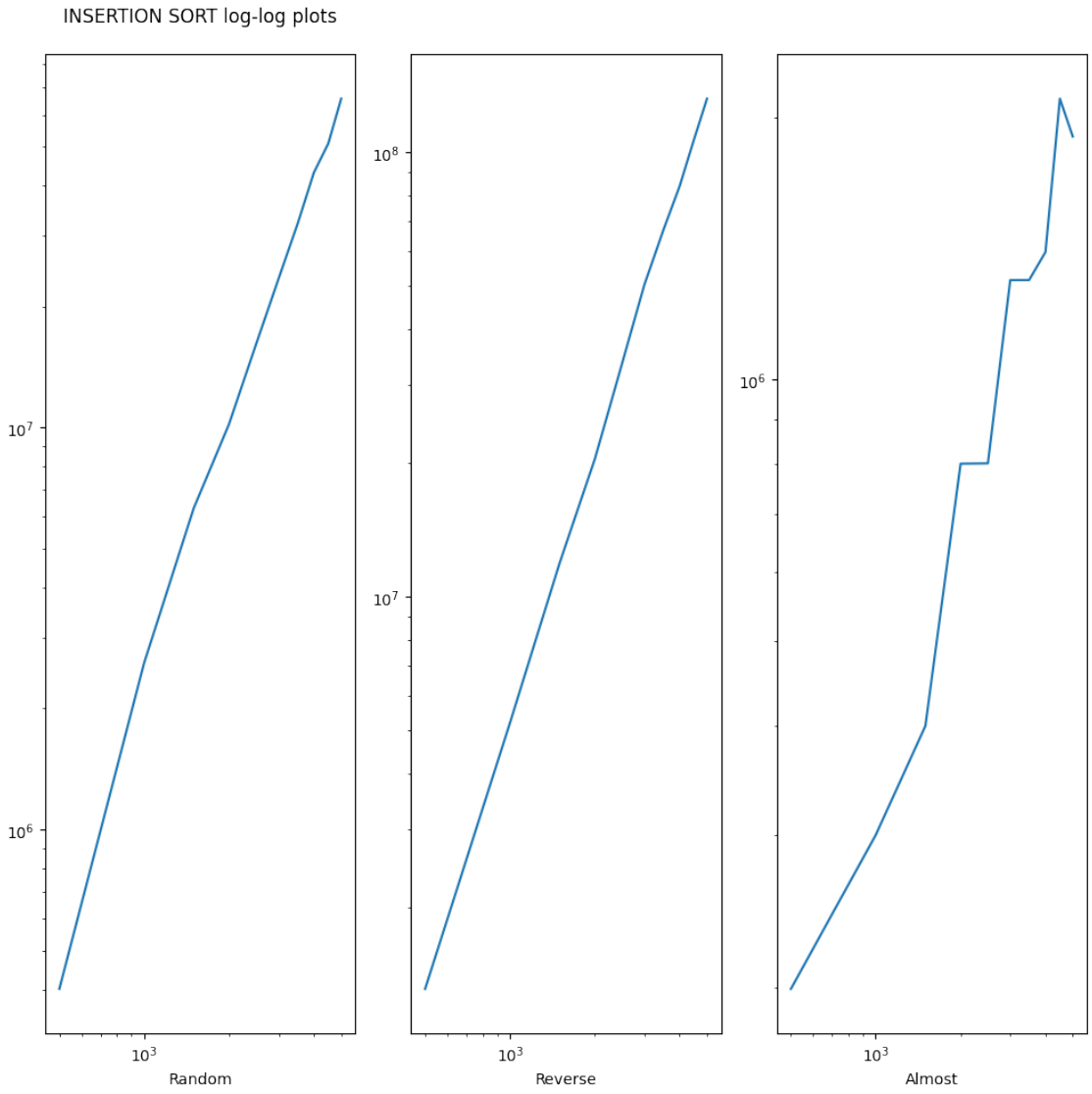
[299]:
```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import numpy as np
from sklearn.metrics import r2_score
from math import log2, e
```

1 Insertion Sort

[300]:
```
df1 = pd.read_csv("InsertionSort_Random.csv")
df2 = pd.read_csv("InsertionSort_Reverse.csv")
df3 = pd.read_csv("InsertionSort_Almost.csv")
# Xi = INPUT SIZE, Yi = Time (nanoseconds)
#RANDOM DATA
x = df1['Size']
y = df1['Time']
#REVERSE DATA
x2 = df2['Size']
y2 = df2['Time']
#ALMOST SORTED DATA
x3 = df3['Size']
y3 = df3['Time']
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("INSERTION SORT POLYNOMIAL PLOTS\n")
plt.plot(x, y)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.plot(x2, y2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.plot(x3, y3)
plt.xlabel("Almost")
plt.tight_layout()
```



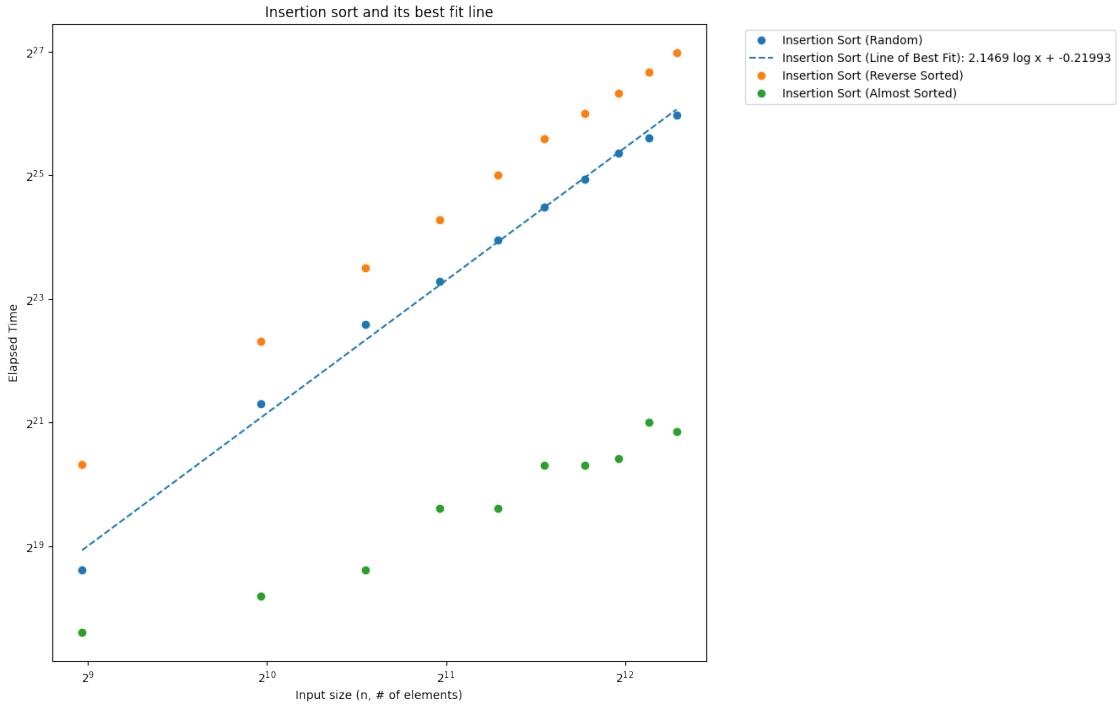
[301]:
```
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("INSERTION SORT log-log plots\n")
plt.loglog(x, y)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.loglog(x2, y2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.loglog(x3, y3)
plt.xlabel("Almost")
plt.tight_layout()
```



1.1 Determining line of best fit

[302]:
```
logx, logy = np.log(x), np.log(y)
m, b = np.polyfit(logx, logy, 1)
fit = np.poly1d((m, b))
expected_logy = fit(logx)
sort_name = "Insertion Sort"
perm_name = "Line of Best Fit"
perm_name2 = "Reverse Input"
perm_name3 = "Almost Sorted Input"
figure(figsize=(10,10),dpi=100)
p = plt.loglog(x, y, '.', base = 2, markersize = 12,label= 'Insertion Sort␣
 ↪(Random)')
```

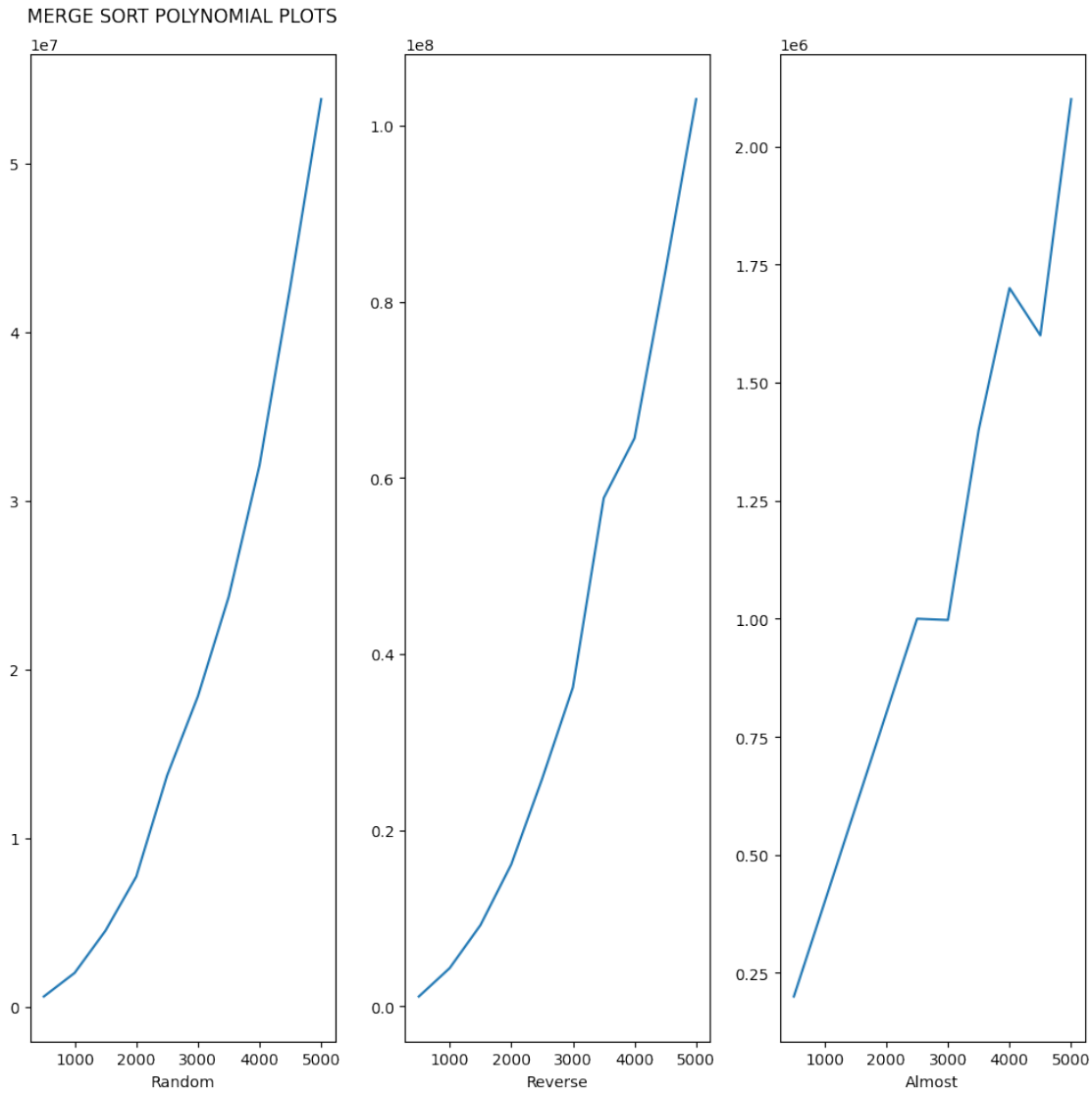
```
fit_p = plt.loglog(x[::len(x)-1], (e ** expected_logy)[::len(y)-1],
                   '--', base = 2,
                   label = f'{sort_name} ({perm_name}): {m:0.5} log x + {b:.5}',
                   markersize = 6, color = p[-1].get_color())

p_rev = plt.loglog(x2, y2, '.', base = 2, markersize = 12,label= 'Insertion␣
 ↪Sort (Reverse Sorted)')
p_almost = plt.loglog(x3, y3, '.', base = 2, markersize = 12,label= 'Insertion␣
 ↪Sort (Almost Sorted)')

plt.title("Insertion sort and its best fit line")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



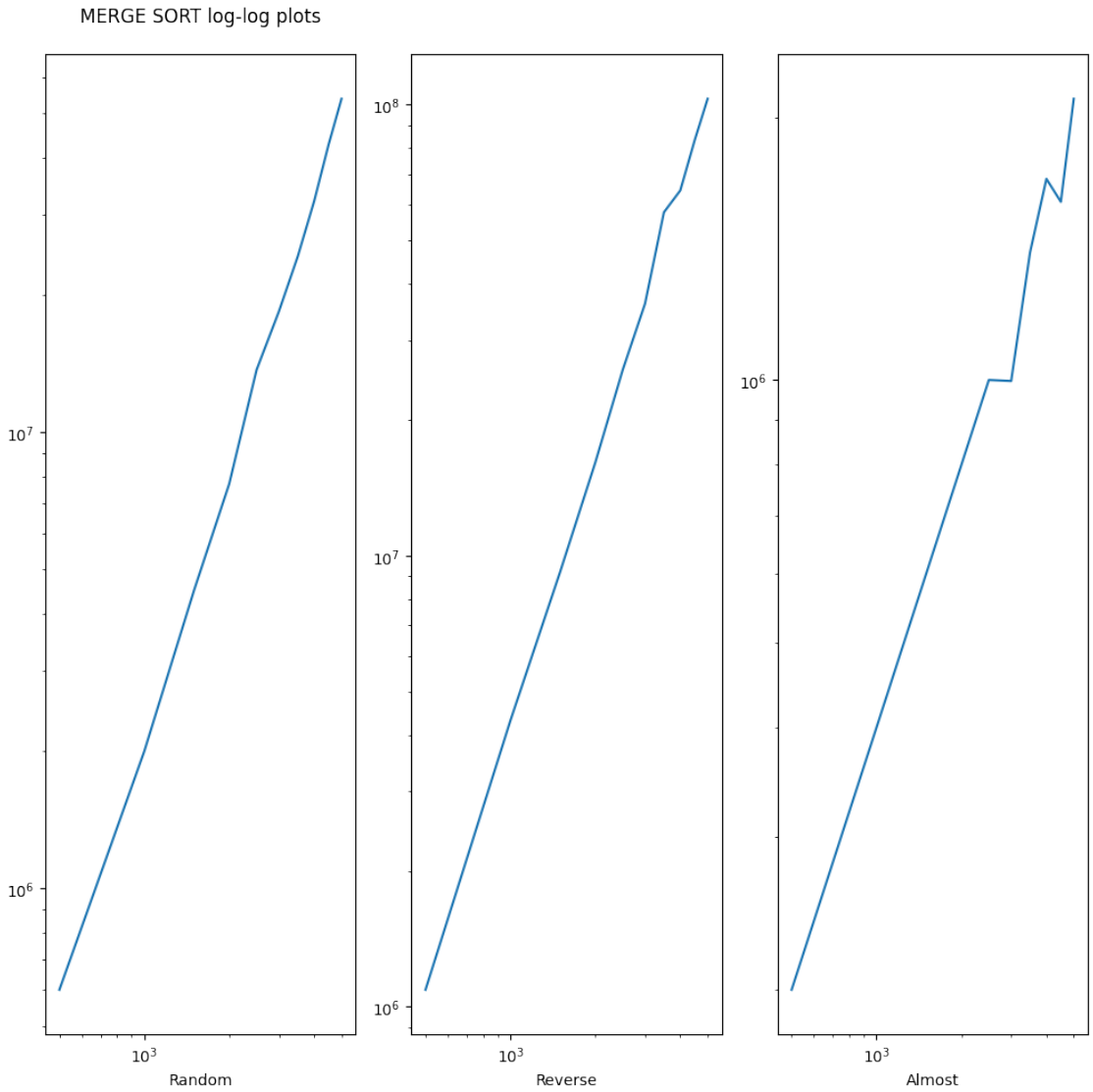
2 Merge Sort

[303]:
```
mf1 = pd.read_csv("MergeSort_Random.csv")
mf2 = pd.read_csv("MergeSort_Reverse.csv")
mf3 = pd.read_csv("MergeSort_Almost.csv")
# Xi = INPUT SIZE, Yi = Time (nanoseconds)
#RANDOM DATA
```

```
xm = mf1['Size']
ym = mf1['Time']
#REVERSE DATA
xm2 = mf2['Size']
ym2 = mf2['Time']
#ALMOST SORTED DATA
xm3 = mf3['Size']
ym3 = mf3['Time']
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("MERGE SORT POLYNOMIAL PLOTS\n")
plt.plot(xm, ym)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.plot(xm2, ym2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.plot(xm3, ym3)
plt.xlabel("Almost")
plt.tight_layout()
```



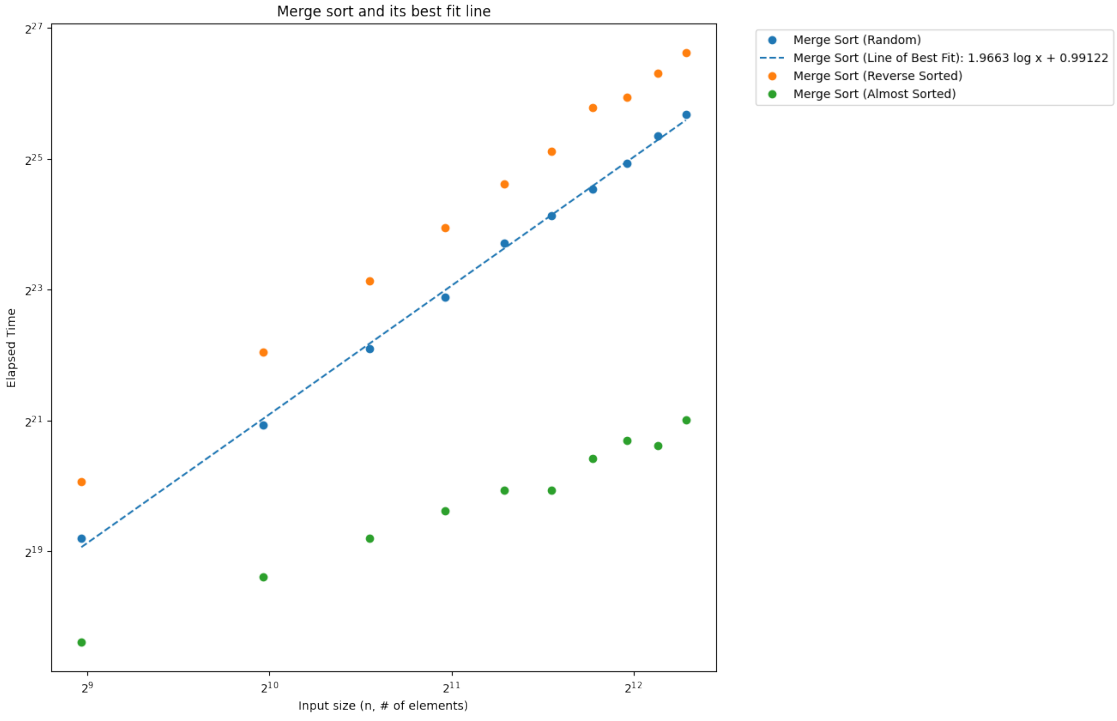
[304]:
```
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("MERGE SORT log-log plots\n")
plt.loglog(xm, ym)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.loglog(xm2, ym2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.loglog(xm3, ym3)
plt.xlabel("Almost")
plt.tight_layout()
```



2.1 Determining line of best fit

[305]:
```
logx, logy = np.log(xm), np.log(ym)
m, b = np.polyfit(logx, logy, 1)
fit = np.poly1d((m, b))
expected_logy = fit(logx)

sort_name = "Merge Sort"
perm_name = "Line of Best Fit"
perm_name2 = "Reverse Input"
perm_name3 = "Almost Sorted Input"
figure(figsize=(10,10),dpi=100)
```

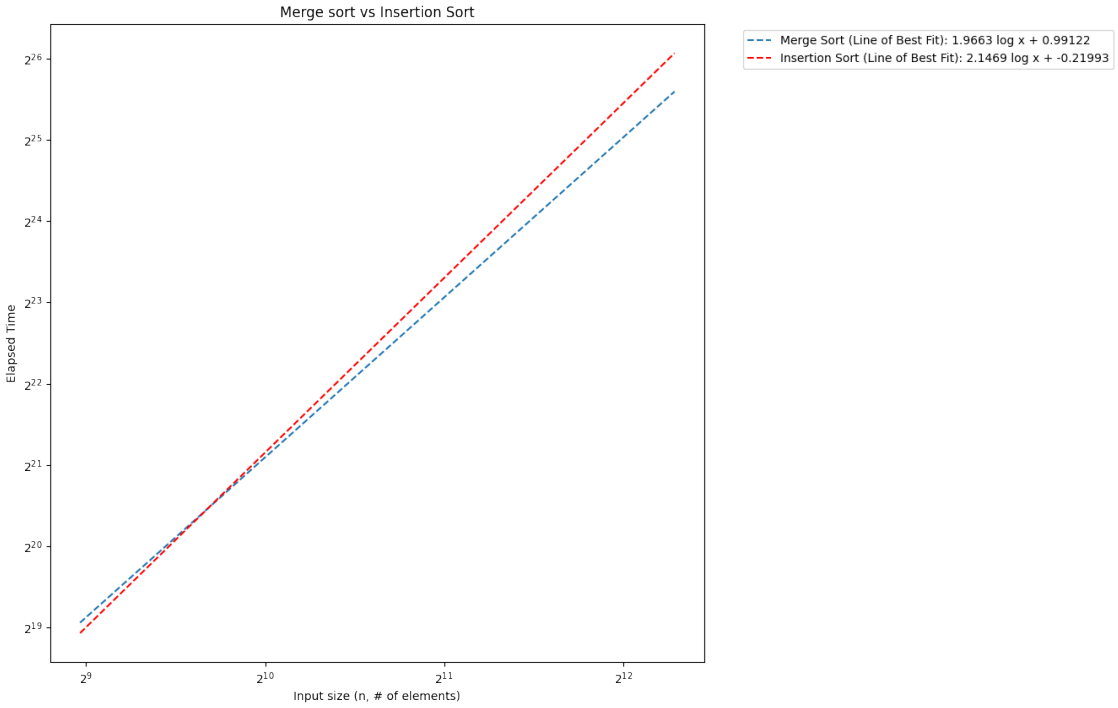
```
p = plt.loglog(xm, ym, '.', base= 2, markersize = 12,label= 'Merge Sort␣
 ↪(Random)')
fit_p = plt.loglog(xm[::len(xm)-1], (e ** expected_logy)[::len(ym)-1],
                   '--', base = 2,
                   label = f'{sort_name} ({perm_name}): {m:0.5} log x + {b:.5}',
                   markersize = 6, color = p[-1].get_color())

p_rev = plt.loglog(xm2, ym2, '.', base = 2, markersize = 12,label= 'Merge Sort␣
 ↪(Reverse Sorted)')
p_almost = plt.loglog(xm3, ym3, '.', base = 2, markersize = 12,label= 'Merge␣
 ↪Sort (Almost Sorted)')

plt.title("Merge sort and its best fit line")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



2.2 Insertion VS Merge Sort

[306]:
```
figure(figsize=(10,10),dpi=100)
logxm, logym = np.log(xm), np.log(ym)
mm, bm = np.polyfit(logxm, logym, 1)
```

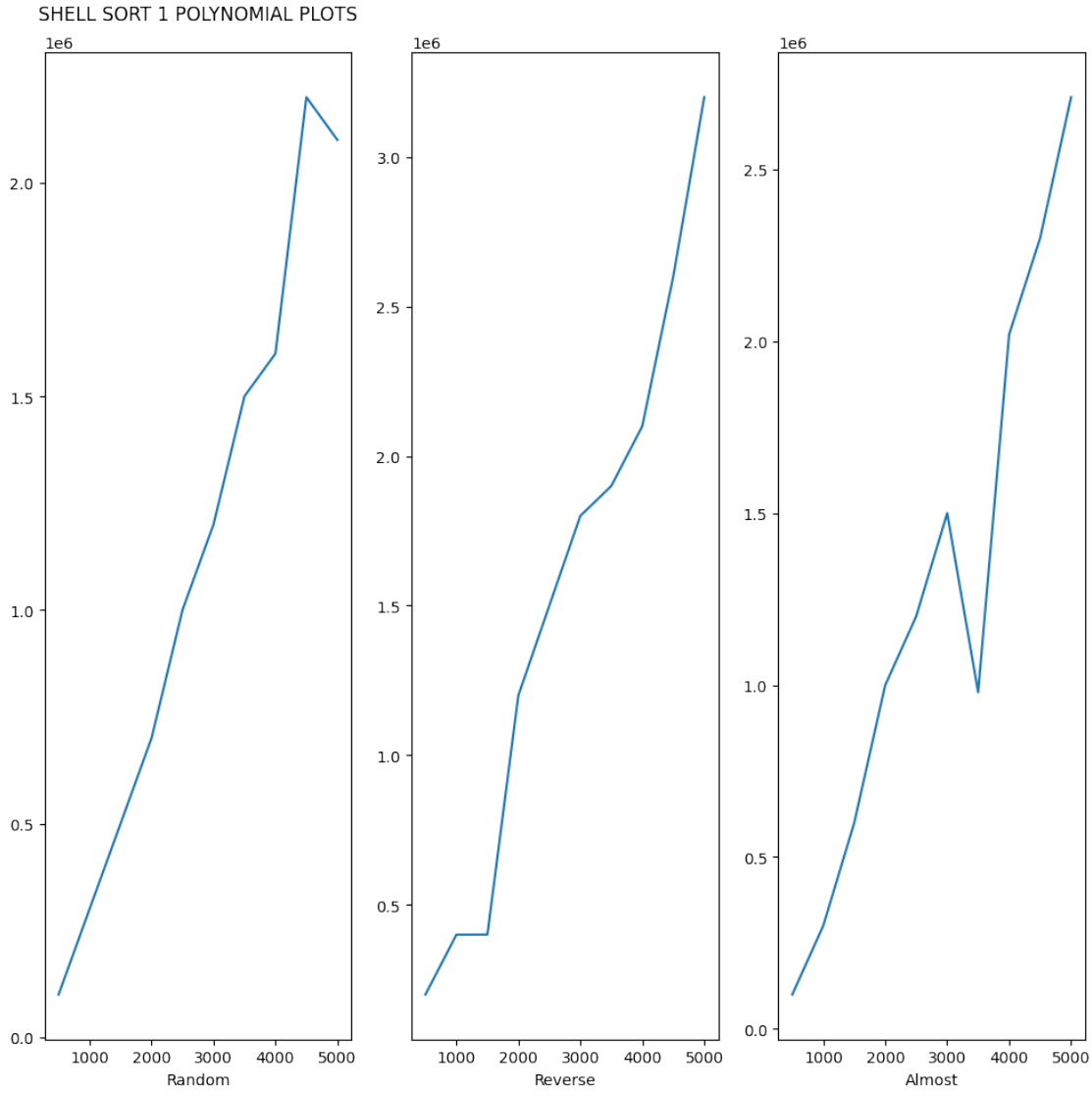
```
fitm = np.poly1d((mm, bm))
expected_logym = fitm(logxm)

logxi, logyi = np.log(x), np.log(y)
mi, bi = np.polyfit(logxi, logyi, 1)
fiti = np.poly1d((mi, bi))
expected_logyi = fiti(logxi)

n1 = "Merge Sort"
plt.loglog(xm[::len(xm)-1], (e ** expected_logym)[::len(ym)-1],
                    '--', base = 2,
                    label = f'{n1} ({perm_name}): {mm:0.5} log x + {bm:.5}',
                    markersize = 6)
n2 = "Insertion Sort"
plt.loglog(x[::len(x)-1], (e ** expected_logyi)[::len(y)-1],
                    '--', base = 2,
                    label = f'{n2} ({perm_name}): {mi:0.5} log x + {bi:.5}',
                    markersize = 6, color = "red")
plt.title("Merge sort vs Insertion Sort")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



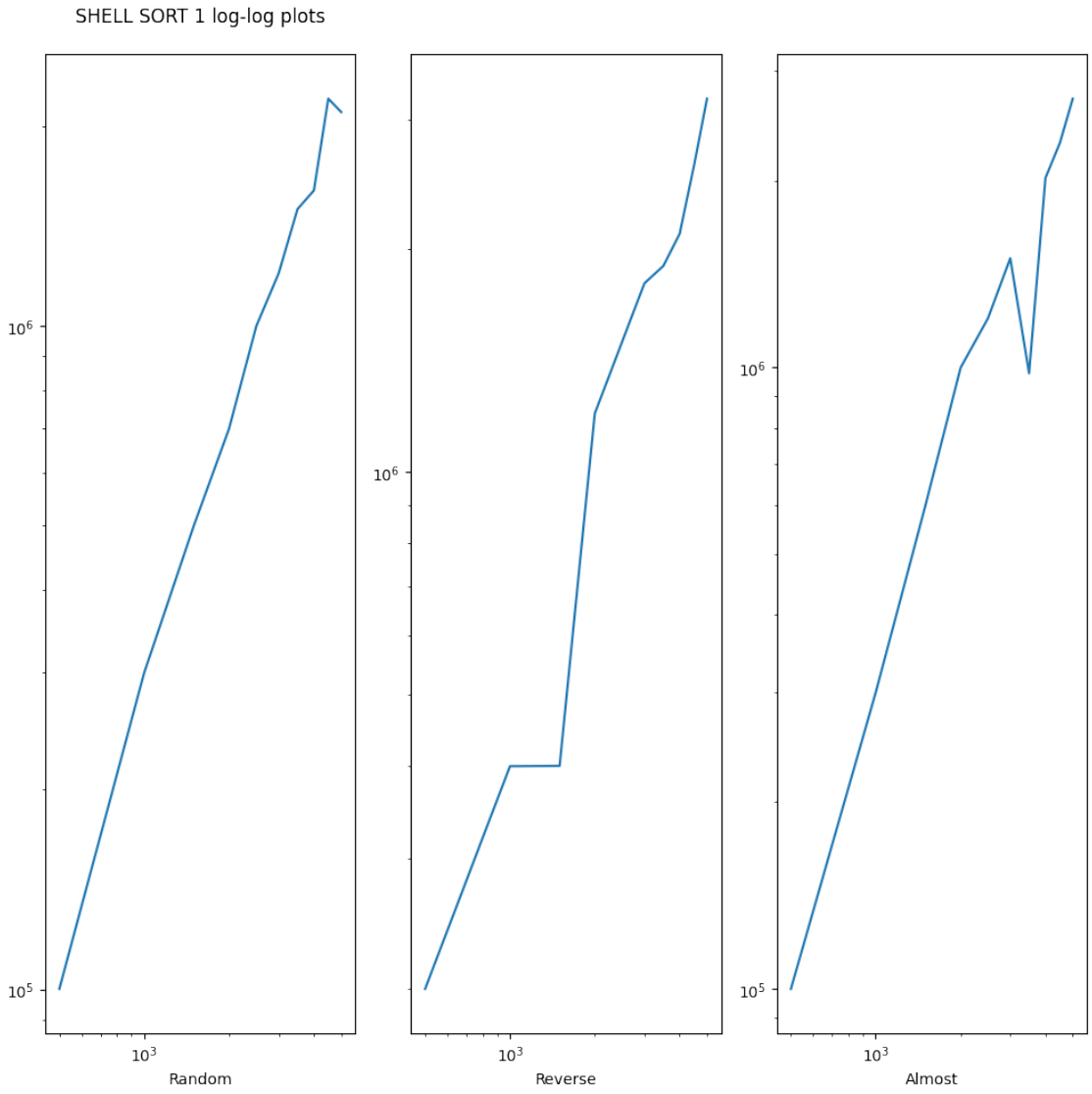
3 Shell Sort

3.1 Shell Sort 1 (Original Gap Sequence)

[307]:
```
s1f1 = pd.read_csv("ShellSort1_Random.csv")
s1f2 = pd.read_csv("ShellSort1_Reverse.csv")
s1f3 = pd.read_csv("ShellSort1_Almost.csv")
# Xi = INPUT SIZE, Yi = Time (nanoseconds)
#RANDOM DATA
xs1 = s1f1['Size']
ys1 = s1f1['Time']
#REVERSE DATA
xs1_2 = s1f2['Size']
ys1_2 = s1f2['Time']
#ALMOST SORTED DATA
xs1_3 = s1f3['Size']
ys1_3 = s1f3['Time']
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("SHELL SORT 1 POLYNOMIAL PLOTS\n")
plt.plot(xs1, ys1)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.plot(xs1_2, ys1_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.plot(xs1_3, ys1_3)
plt.xlabel("Almost")
plt.tight_layout()
```



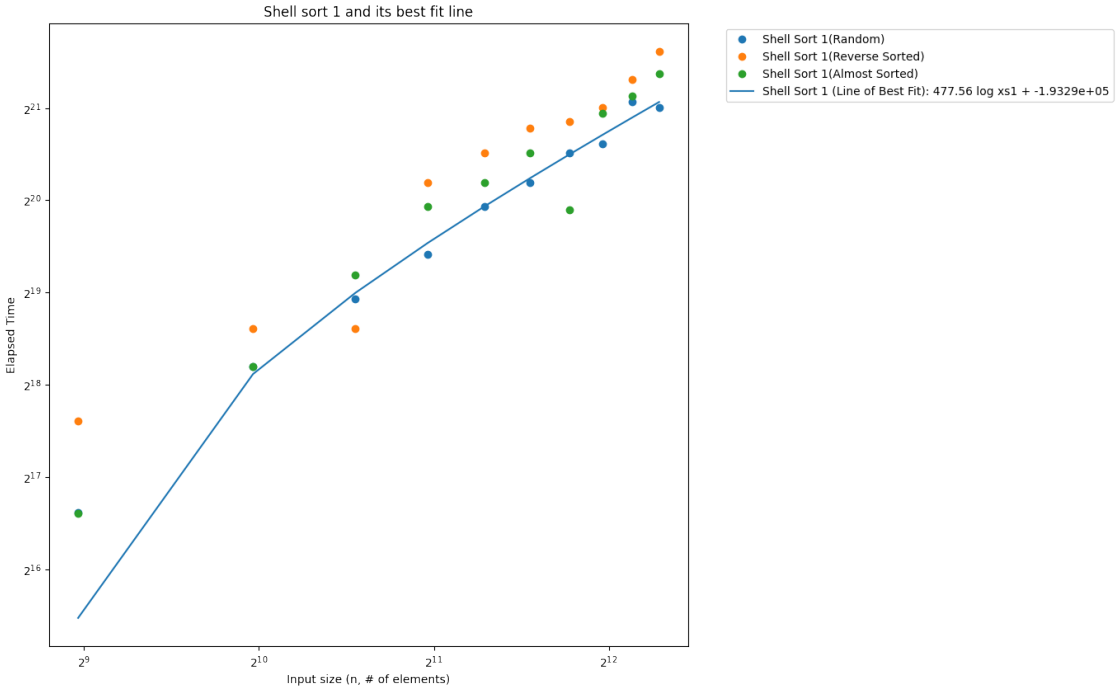
[308]:
```
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("SHELL SORT 1 log-log plots\n")
plt.loglog(xs1, ys1)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.loglog(xs1_2, ys1_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.loglog(xs1_3, ys1_3)
plt.xlabel("Almost")
plt.tight_layout()
```



3.1.1 Determining Line of Best fit

[309]:
```
logxs1, logys1 = np.log(xs1), np.log(ys1)
ms1, bs1 = np.polyfit(xs1, ys1, 1)
fit = np.poly1d((ms1, bs1))
expected_logys1 = fit(logxs1)
sort_name = "Shell Sort 1"
perm_name = "Line of Best Fit"
perm_name2 = "Reverse Input"
perm_name3 = "Almost Sorted Input"
figure(figsize=(10,10),dpi=100)
p = plt.loglog(xs1, ys1, '.', base = 2, markersize = 12,label= 'Shell Sort␣
↪1(Random)')
```

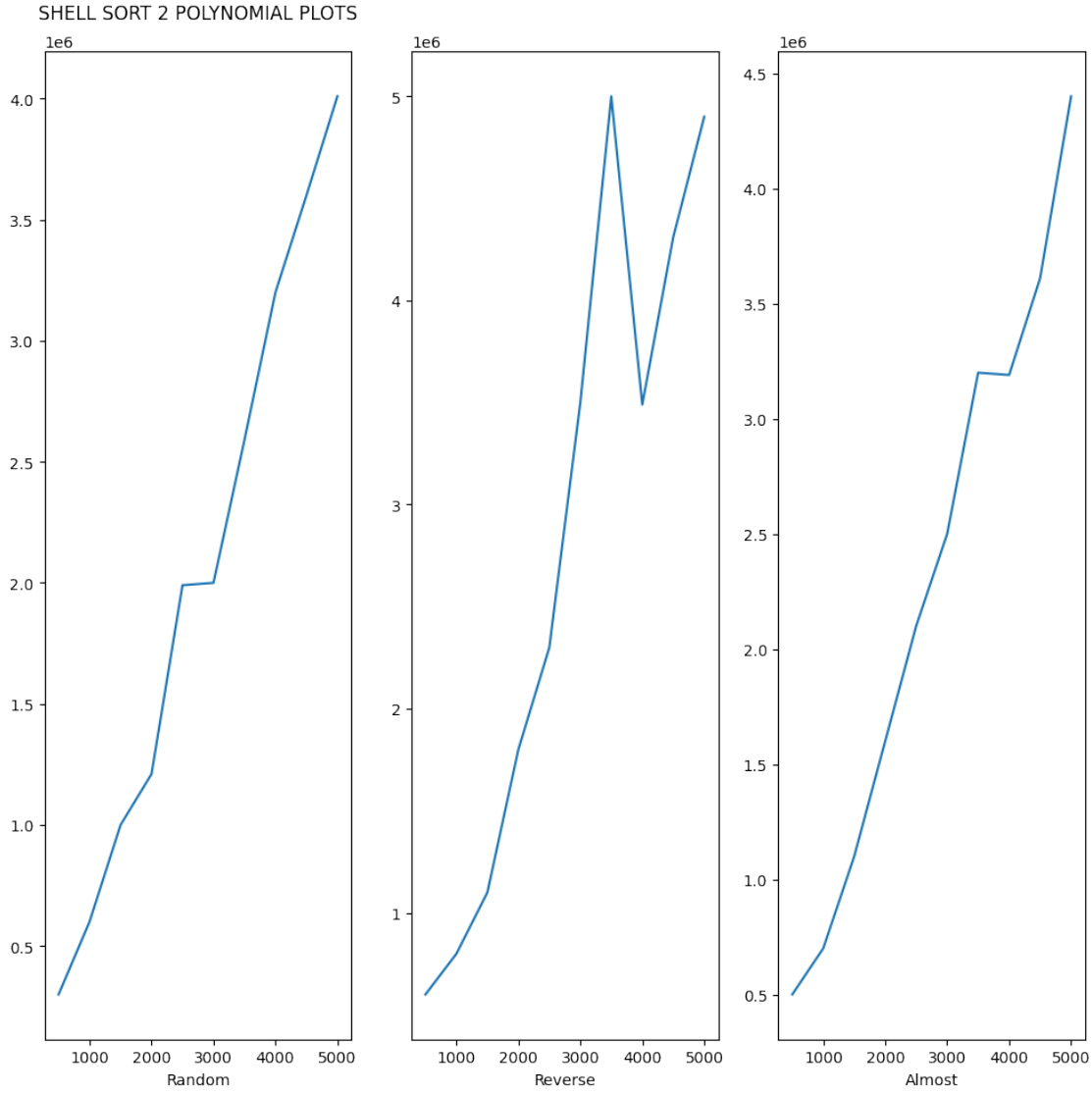
```
p_rev = plt.loglog(xs1_2, ys1_2, '.', base= 2, markersize = 12,label= 'Shell␣
↪Sort 1(Reverse Sorted)')
p_almost = plt.loglog(xs1_3, ys1_3, '.', base = 2, markersize = 12,label=␣
↪'Shell Sort 1(Almost Sorted)')
plt.plot(xs1, ms1*xs1 + (bs1), label = f'{sort_name} ({perm_name}): {ms1:0.5}␣
↪log xs1 + {bs1:.5}',markersize = 6, color = p[-1].get_color())
plt.title("Shell sort 1 and its best fit line")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



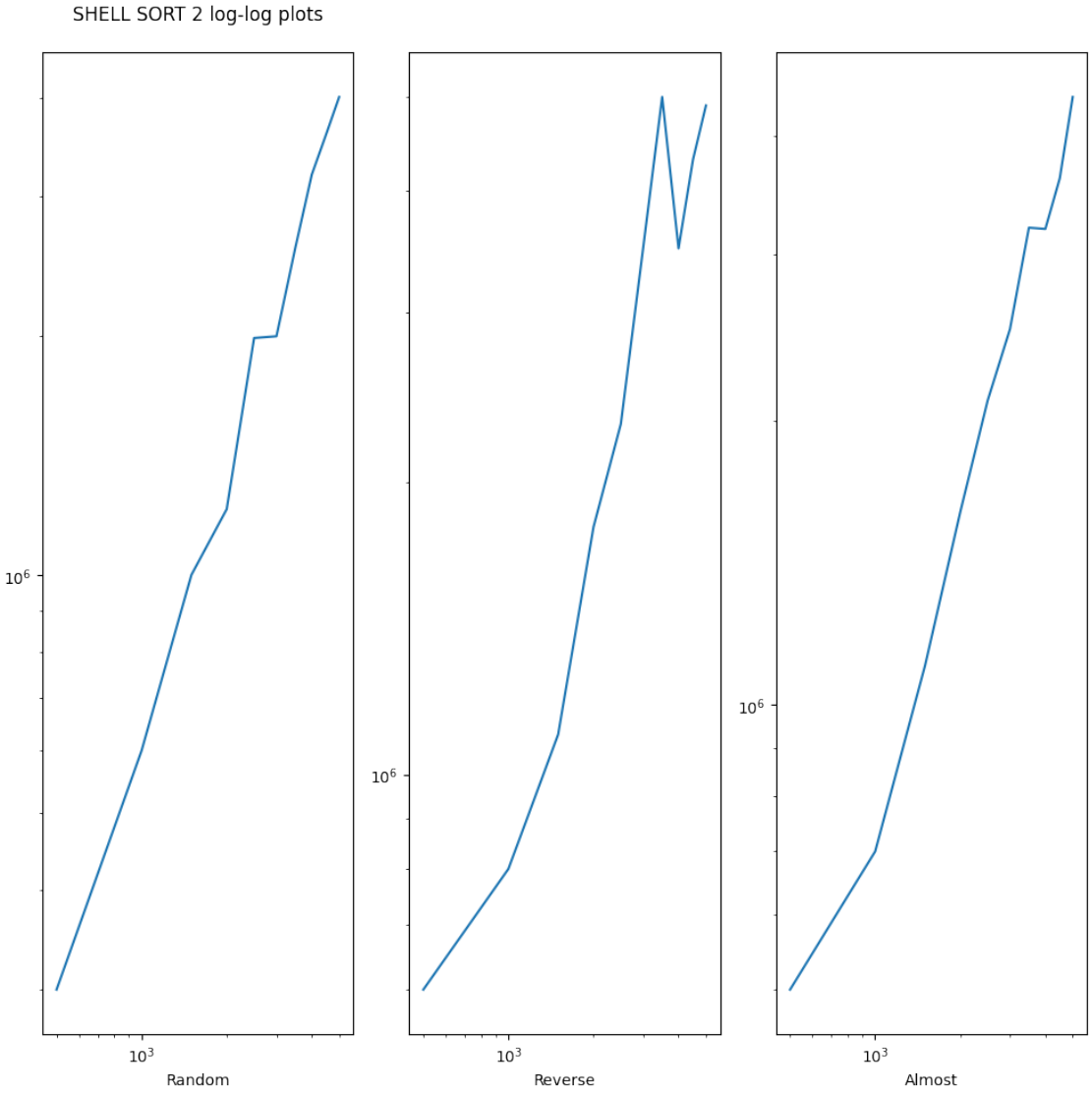
3.2 Shell Sort 2 (A083318)

[310]:
```
s2f1 = pd.read_csv("ShellSort2_Random.csv")
s2f2 = pd.read_csv("ShellSort2_Reverse.csv")
s2f3 = pd.read_csv("ShellSort2_Almost.csv")
# Xi = INPUT SIZE, Yi = Time (nanoseconds)
#RANDOM DATA
xs2 = s2f1['Size']
ys2 = s2f1['Time']
#REVERSE DATA
xs2_2 = s2f2['Size']
ys2_2 = s2f2['Time']
```

```
#ALMOST SORTED DATA
xs2_3 = s2f3['Size']
ys2_3 = s2f3['Time']
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("SHELL SORT 2 POLYNOMIAL PLOTS\n")
plt.plot(xs2, ys2)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.plot(xs2_2, ys2_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.plot(xs2_3, ys2_3)
plt.xlabel("Almost")
plt.tight_layout()
```

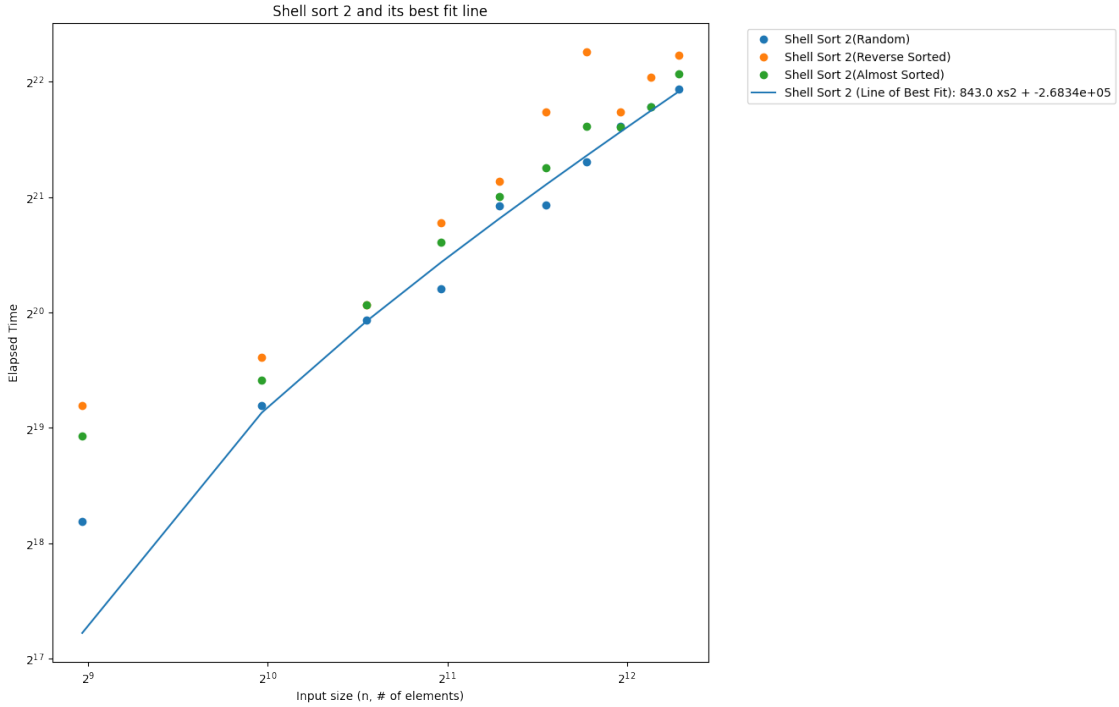


[311]:
```
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("SHELL SORT 2 log-log plots\n")
plt.loglog(xs2, ys2)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.loglog(xs2_2, ys2_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.loglog(xs2_3, ys2_3)
plt.xlabel("Almost")
plt.tight_layout()
```



3.2.1 Determining line of best fit

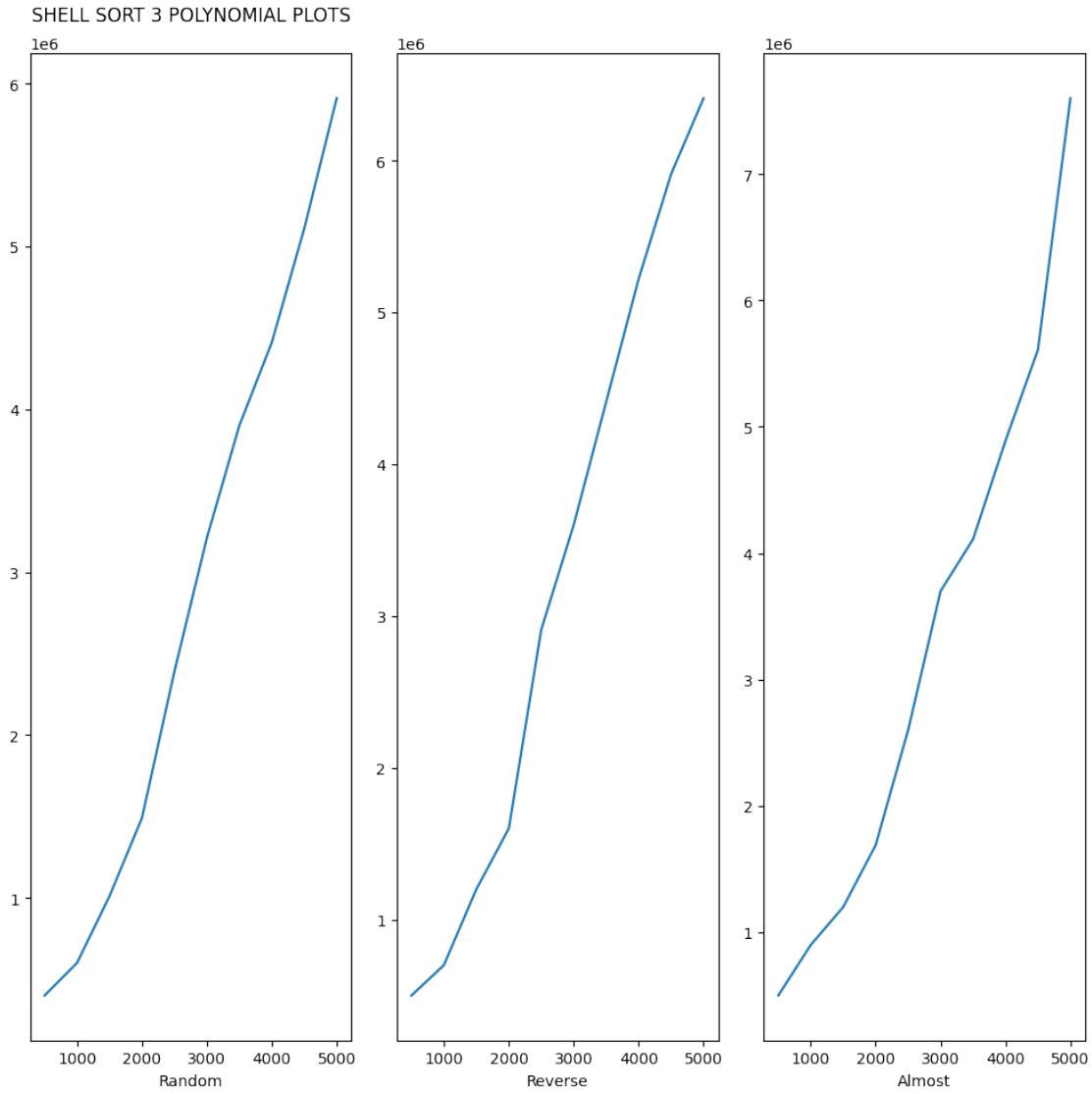
[312]:
```
logxs2, logys2 = np.log(xs2), np.log(ys2)
ms2, bs2 = np.polyfit(xs2, ys2, 1)
fit = np.poly1d((ms2, bs2))
expected_logys2 = fit(logxs2)
sort_name = "Shell Sort 2"
perm_name = "Line of Best Fit"
perm_name2 = "Reverse Input"
perm_name3 = "Almost Sorted Input"
figure(figsize=(10,10),dpi=100)
p = plt.loglog(xs2, ys2, '.', base = 2, markersize = 12,label= 'Shell Sort␣
↪2(Random)')
p_rev = plt.loglog(xs2_2, ys2_2, '.', base = 2, markersize = 12,label= 'Shell␣
↪Sort 2(Reverse Sorted)')
p_almost = plt.loglog(xs2_3, ys2_3, '.', base = 2, markersize = 12,label=␣
↪'Shell Sort 2(Almost Sorted)')
plt.plot(xs2, ms2*xs2 + (bs2), label = f'{sort_name} ({perm_name}): {ms2:0.5}␣
↪xs2 + {bs2:.5}',
                  markersize = 6, color = p[-1].get_color())
plt.title("Shell sort 2 and its best fit line")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



3.3 Shell Sort 3 (A003586)

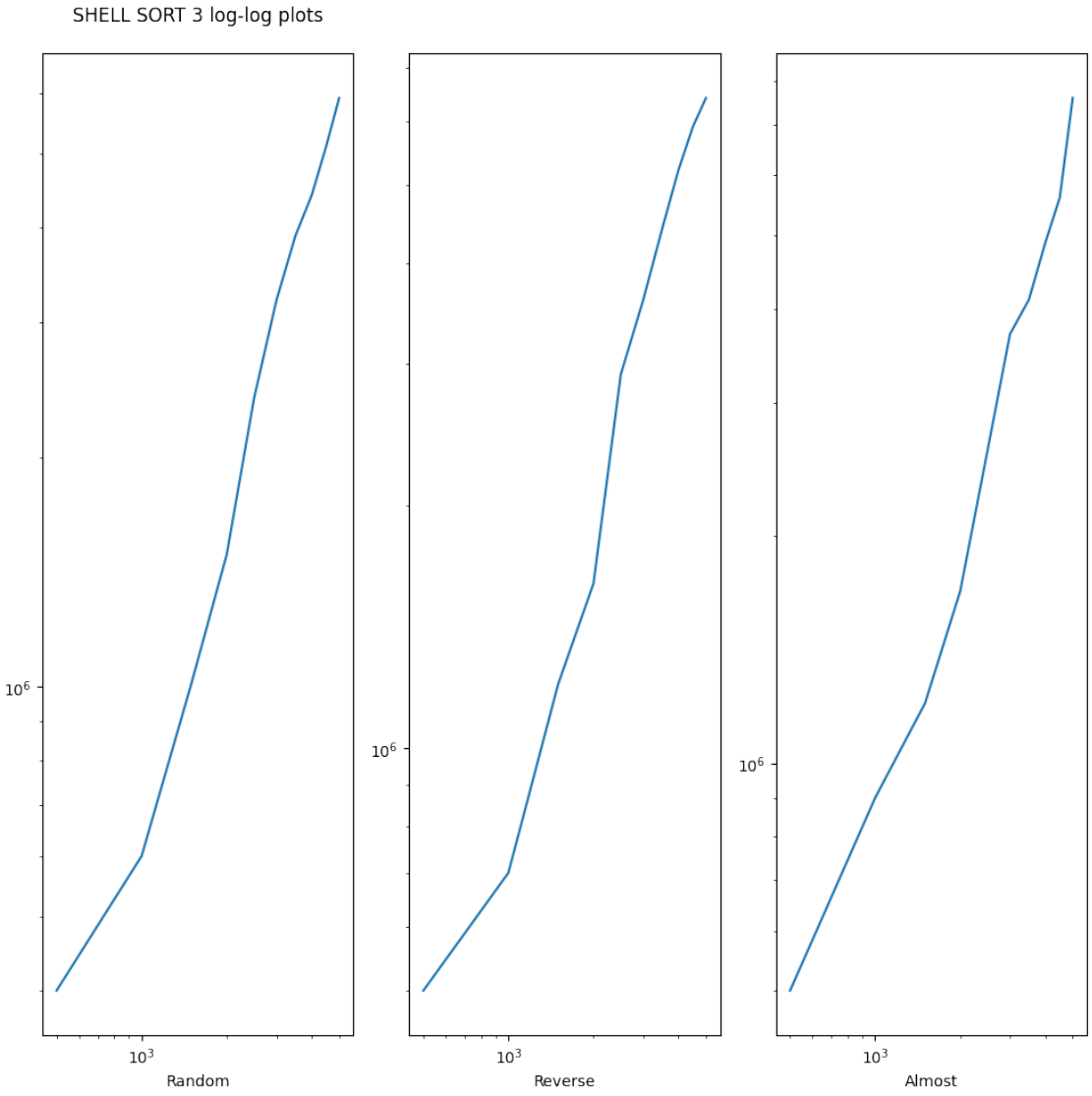
[313]:
```
s3f1 = pd.read_csv("ShellSort3_Random.csv")
s3f2 = pd.read_csv("ShellSort3_Reverse.csv")
s3f3 = pd.read_csv("ShellSort3_Almost.csv")
# Xi = INPUT SIZE, Yi = Time (nanoseconds)
#RANDOM DATA
xs3 = s3f1['Size']
ys3 = s3f1['Time']
#REVERSE DATA
xs3_2 = s3f2['Size']
ys3_2 = s3f2['Time']
#ALMOST SORTED DATA
xs3_3 = s3f3['Size']
ys3_3 = s3f3['Time']
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("SHELL SORT 3 POLYNOMIAL PLOTS\n")
plt.plot(xs3, ys3)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.plot(xs3_2, ys3_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
```

```
plt.plot(xs3_3, ys3_3)
plt.xlabel("Almost")
plt.tight_layout()
```



[314]:
```
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("SHELL SORT 3 log-log plots\n")
plt.loglog(xs3, ys3)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.loglog(xs3_2, ys3_2)
plt.xlabel("Reverse")
```

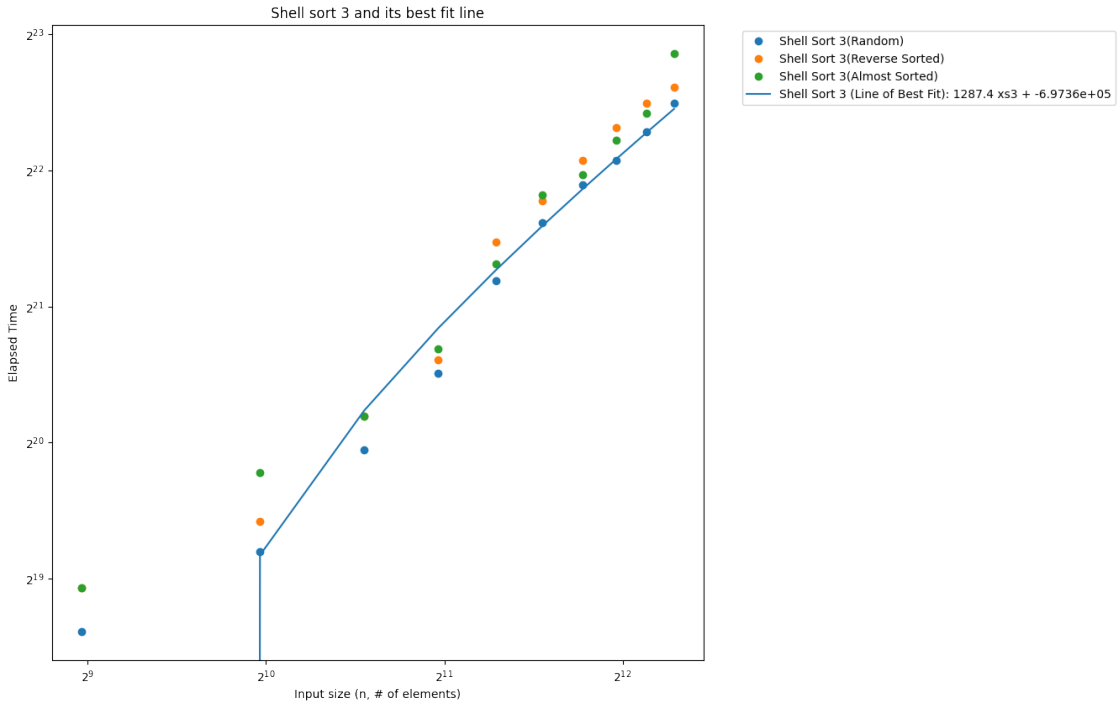
```
plt.subplot(1,3,3)
plt.loglog(xs3_3, ys3_3)
plt.xlabel("Almost")
plt.tight_layout()
```



3.3.1 Determining Line of best fit

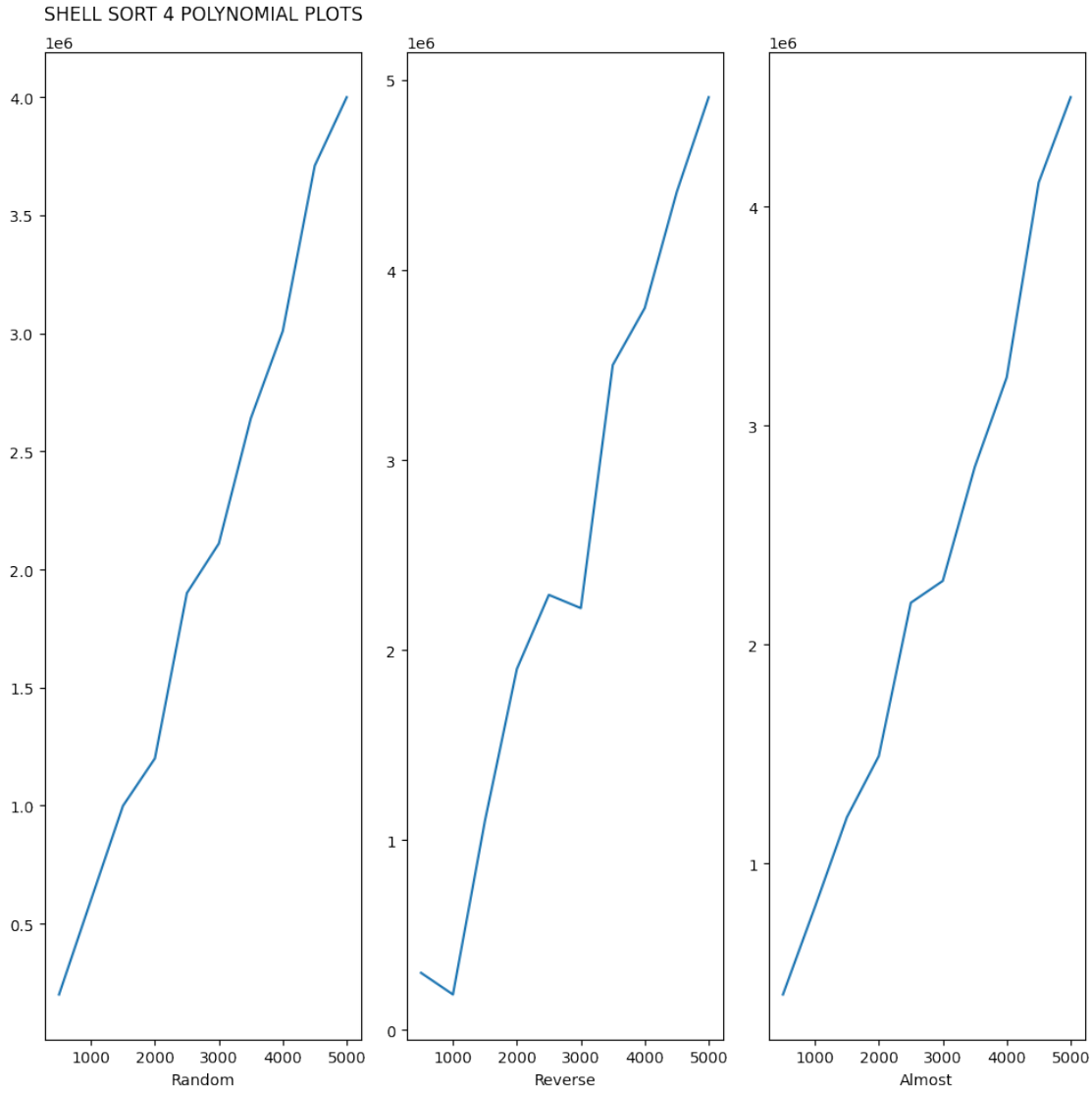
[315]:
```
logxs3, logys3 = np.log(xs3), np.log(ys3)
ms3, bs3 = np.polyfit(xs3, ys3, 1)
fit = np.poly1d((ms3, bs3))
expected_logys3 = fit(logxs3)
sort_name = "Shell Sort 3"
```

```
perm_name = "Line of Best Fit"
perm_name2 = "Reverse Input"
perm_name3 = "Almost Sorted Input"
figure(figsize=(10,10),dpi=100)
p = plt.loglog(xs3, ys3, '.', base= 2, markersize = 12,label= 'Shell Sort␣
 ↪3(Random)')
p_rev = plt.loglog(xs3_2, ys3_2, '.', base = 2, markersize = 12,label= 'Shell␣
 ↪Sort 3(Reverse Sorted)')
p_almost = plt.loglog(xs3_3, ys3_3, '.', base = 2, markersize = 12,label=␣
 ↪'Shell Sort 3(Almost Sorted)')
plt.plot(xs3, ms3*xs3 + (bs3), label = f'{sort_name} ({perm_name}): {ms3:0.5}␣
 ↪xs3 + {bs3:.5}',
                    markersize = 6, color = p[-1].get_color())
plt.title("Shell sort 3 and its best fit line")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

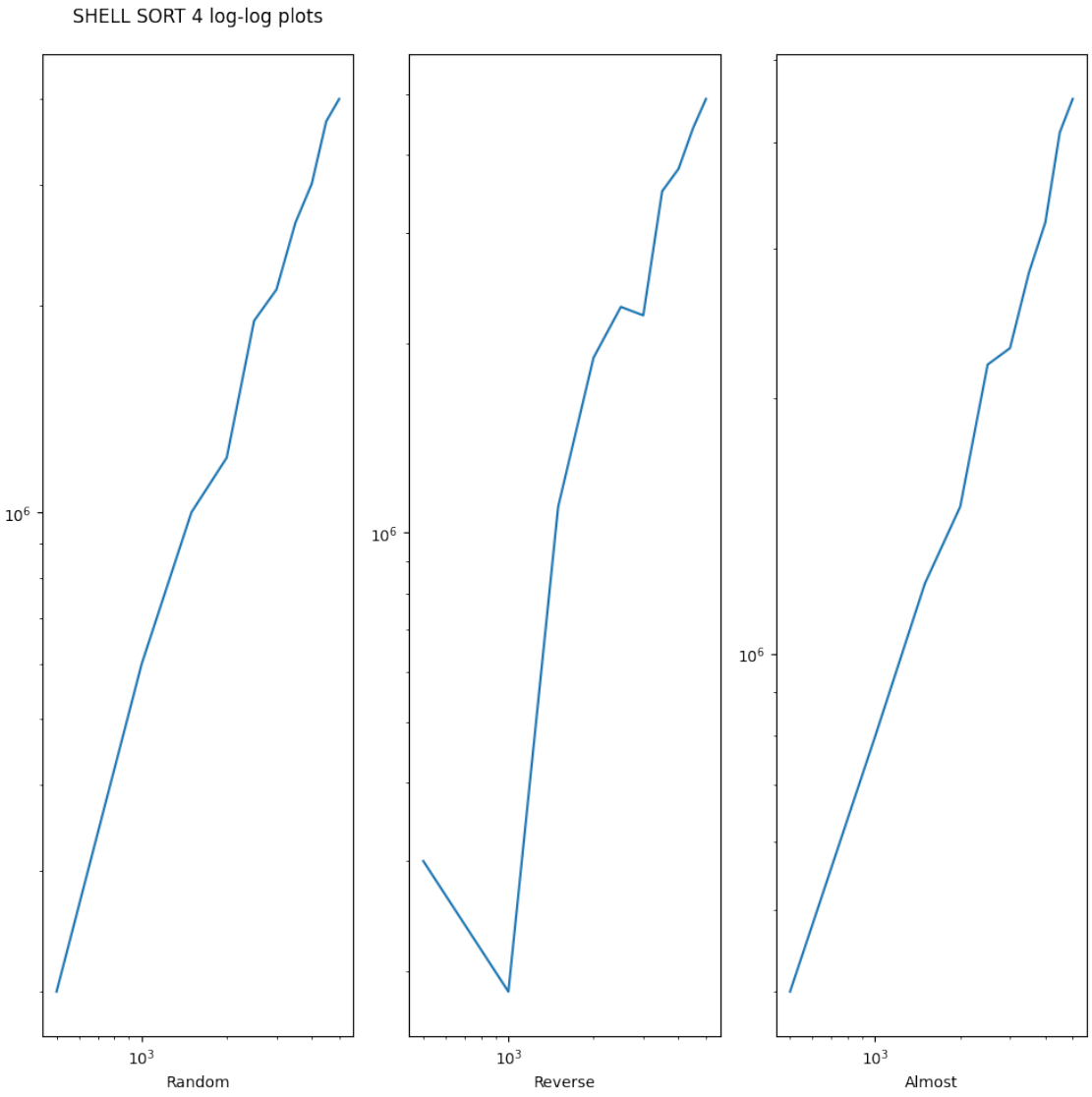


3.4 Shell Sort 4 (A033622)

[316]:
```
s4f1 = pd.read_csv("ShellSort4_Random.csv")
s4f2 = pd.read_csv("ShellSort4_Reverse.csv")
s4f3 = pd.read_csv("ShellSort4_Almost.csv")
# Xi = INPUT SIZE, Yi = Time (nanoseconds)
#RANDOM DATA
xs4 = s4f1['Size']
ys4 = s4f1['Time']
#REVERSE DATA
xs4_2 = s4f2['Size']
ys4_2 = s4f2['Time']
#ALMOST SORTED DATA
xs4_3 = s4f3['Size']
ys4_3 = s4f3['Time']
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("SHELL SORT 4 POLYNOMIAL PLOTS\n")
plt.plot(xs4, ys4)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.plot(xs4_2, ys4_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.plot(xs4_3, ys4_3)
plt.xlabel("Almost")
plt.tight_layout()
```



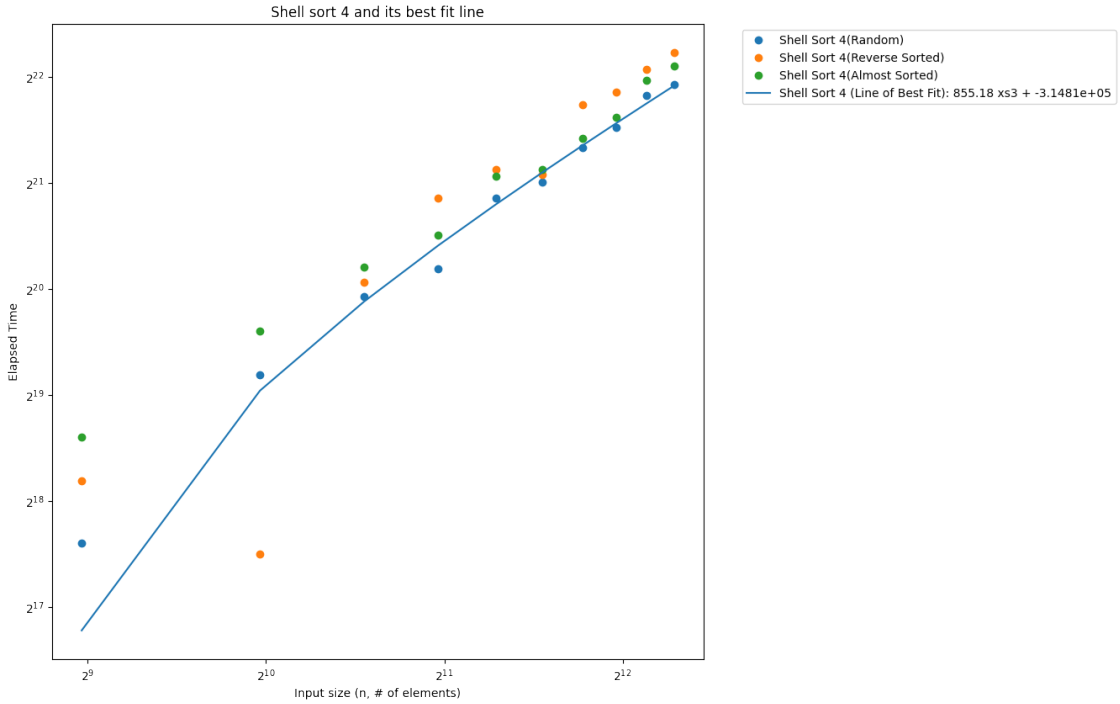
[317]:
```
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("SHELL SORT 4 log-log plots\n")
plt.loglog(xs4, ys4)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.loglog(xs4_2, ys4_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.loglog(xs4_3, ys4_3)
plt.xlabel("Almost")
plt.tight_layout()
```



3.4.1 Determining Line of Best Fit

[318]:
```
logxs4, logys4 = np.log(xs4), np.log(ys4)
ms4, bs4 = np.polyfit(xs4, ys4, 1)
fit = np.poly1d((ms4, bs4))
expected_logys4 = fit(logxs4)
sort_name = "Shell Sort 4"
perm_name = "Line of Best Fit"
perm_name2 = "Reverse Input"
perm_name3 = "Almost Sorted Input"
figure(figsize=(10,10),dpi=100)
p = plt.loglog(xs4, ys4, '.', base= 2, markersize = 12,label= 'Shell Sort␣
↪4(Random)')
```

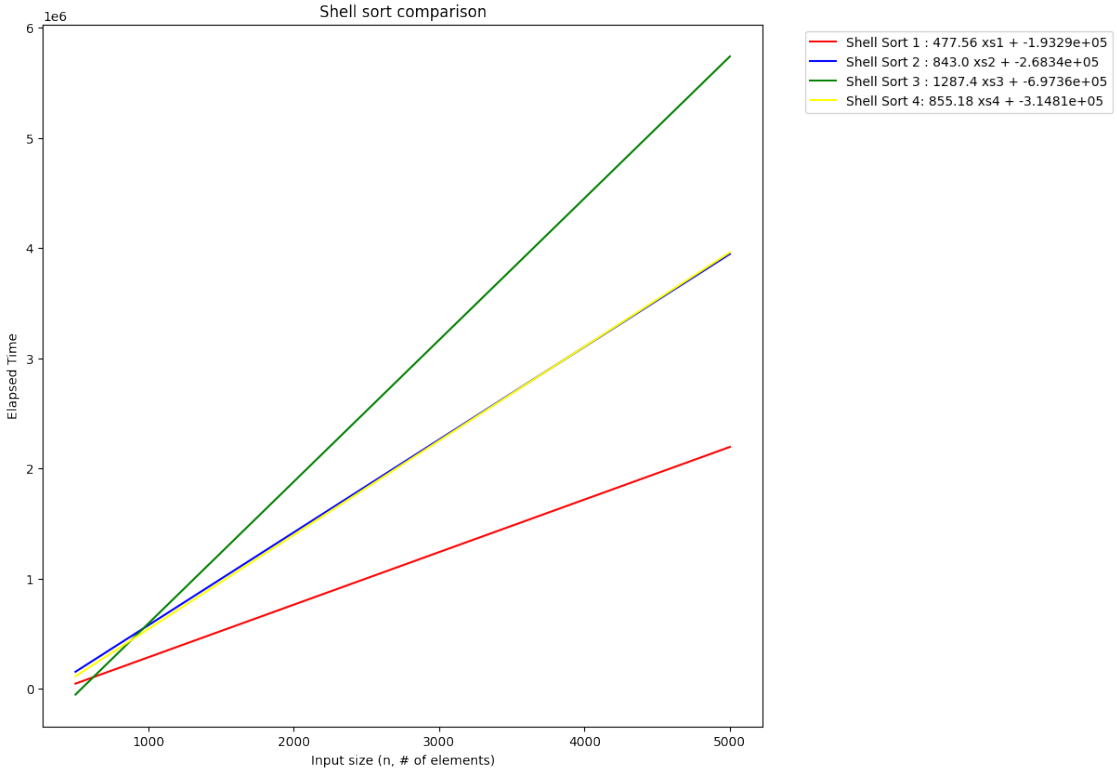
```
p_rev = plt.loglog(xs4_2, ys4_2, '.', base = 2, markersize = 12,label= 'Shell␣
 ↪Sort 4(Reverse Sorted)')
p_almost = plt.loglog(xs4_3, ys4_3, '.', base = 2, markersize = 12,label=␣
 ↪'Shell Sort 4(Almost Sorted)')
plt.plot(xs4, ms4*xs4 + (bs4), label = f'{sort_name} ({perm_name}): {ms4:0.5}␣
 ↪xs3 + {bs4:.5}',
                   markersize = 6, color = p[-1].get_color())
plt.title("Shell sort 4 and its best fit line")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



3.5 Comparing Different Shellsorts

[319]:
```
figure(figsize=(10,10),dpi=100)
s1 = "Shell Sort 1"
s2 = "Shell Sort 2"
s3 = "Shell Sort 3"
s4 = "Shell Sort 4"
plt.plot(xs1, ms1*xs1 + (bs1), label = f'{s1} : {ms1:0.5} xs1 + {bs1:.
 ↪5}',markersize = 6, color = "red")
plt.plot(xs2, ms2*xs2 + (bs2), label = f'{s2} : {ms2:0.5} xs2 + {bs2:.
 ↪5}',markersize = 6, color = "blue")
```

```
plt.plot(xs3, ms3*xs3 + (bs3), label = f'{s3} : {ms3:0.5} xs3 + {bs3:.
↪5}',markersize = 6, color = "green")
plt.plot(xs4, ms4*xs4 + (bs4), label = f'{s4}: {ms4:0.5} xs4 + {bs4:.5}',␣
↪markersize = 6, color = "yellow")
plt.title("Shell sort comparison")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

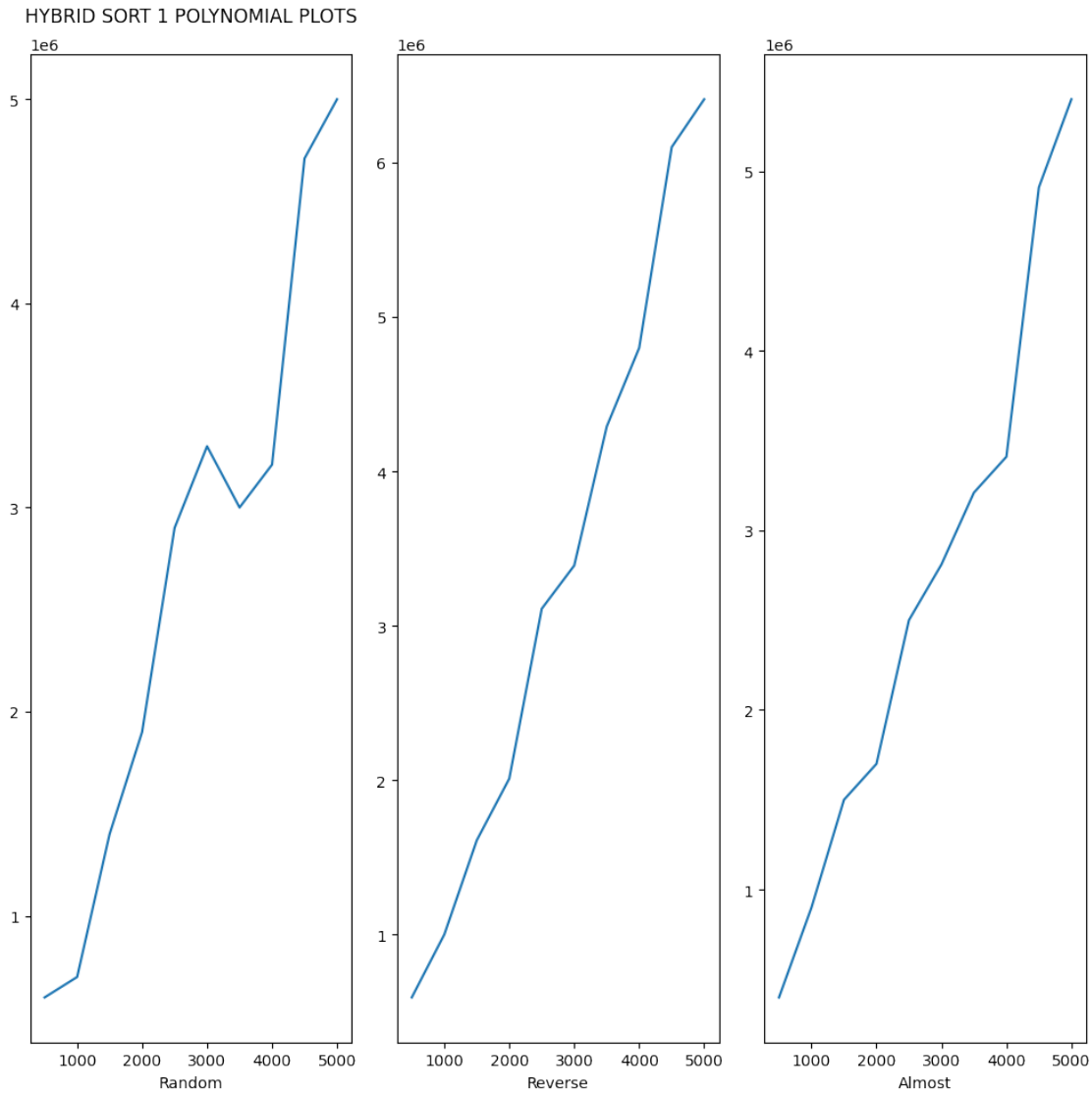


4 Hybrid Sort

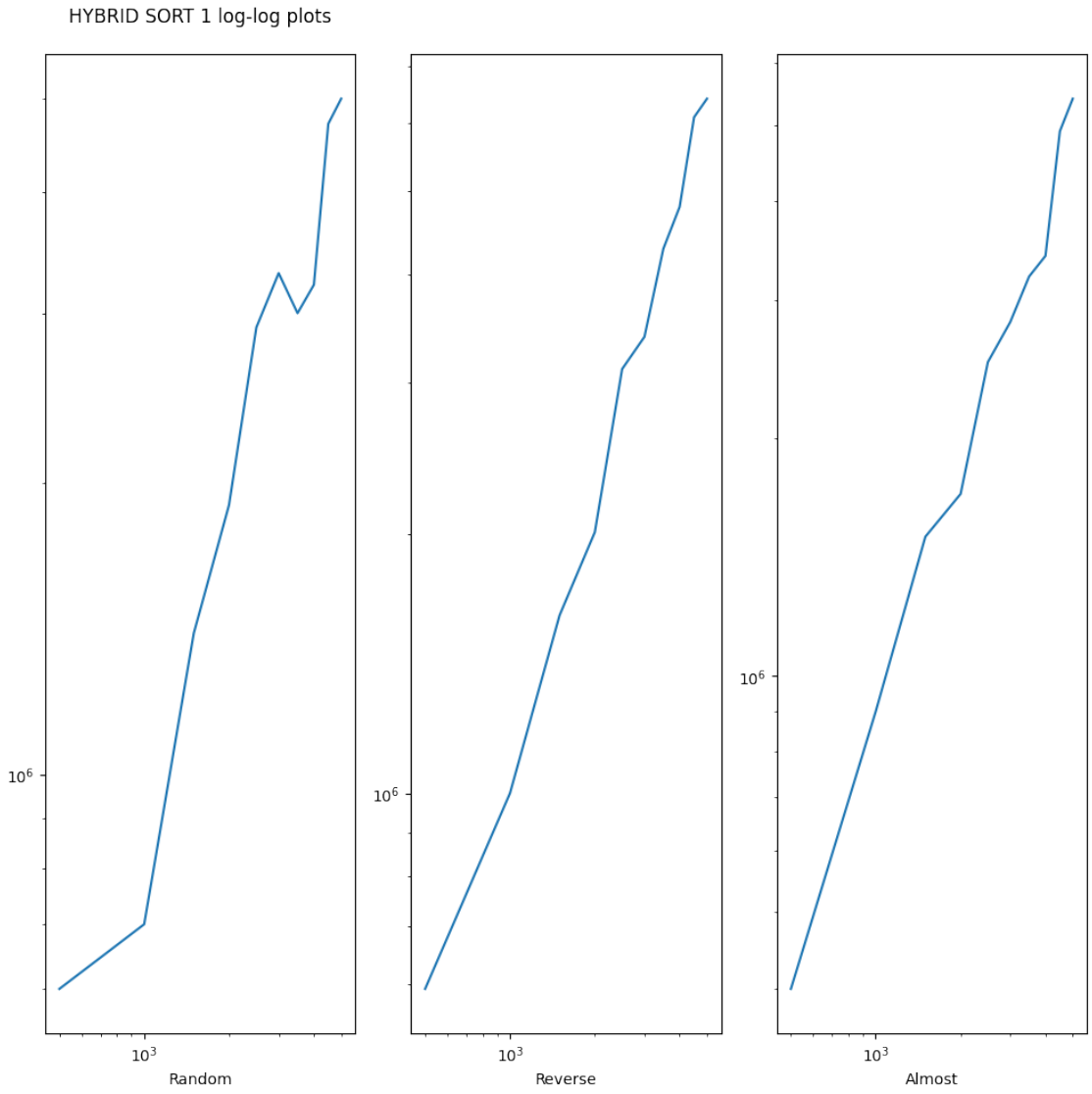
4.1 Hybrid Sort 1: H = n^0.5

[320]:
```
h1f1 = pd.read_csv("HybridSort1_Random.csv")
h1f2 = pd.read_csv("HybridSort1_Reverse.csv")
h1f3 = pd.read_csv("HybridSort1_Almost.csv")
# Xi = INPUT SIZE, Yi = Time (nanoseconds)
#RANDOM DATA
xh1 = h1f1['Size']
yh1 = h1f1['Time']
#REVERSE DATA
```

```
xh1_2 = h1f2['Size']
yh1_2 = h1f2['Time']
#ALMOST SORTED DATA
xh1_3 = h1f3['Size']
yh1_3 = h1f3['Time']
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("HYBRID SORT 1 POLYNOMIAL PLOTS\n")
plt.plot(xh1, yh1)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.plot(xh1_2, yh1_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.plot(xh1_3, yh1_3)
plt.xlabel("Almost")
plt.tight_layout()
```



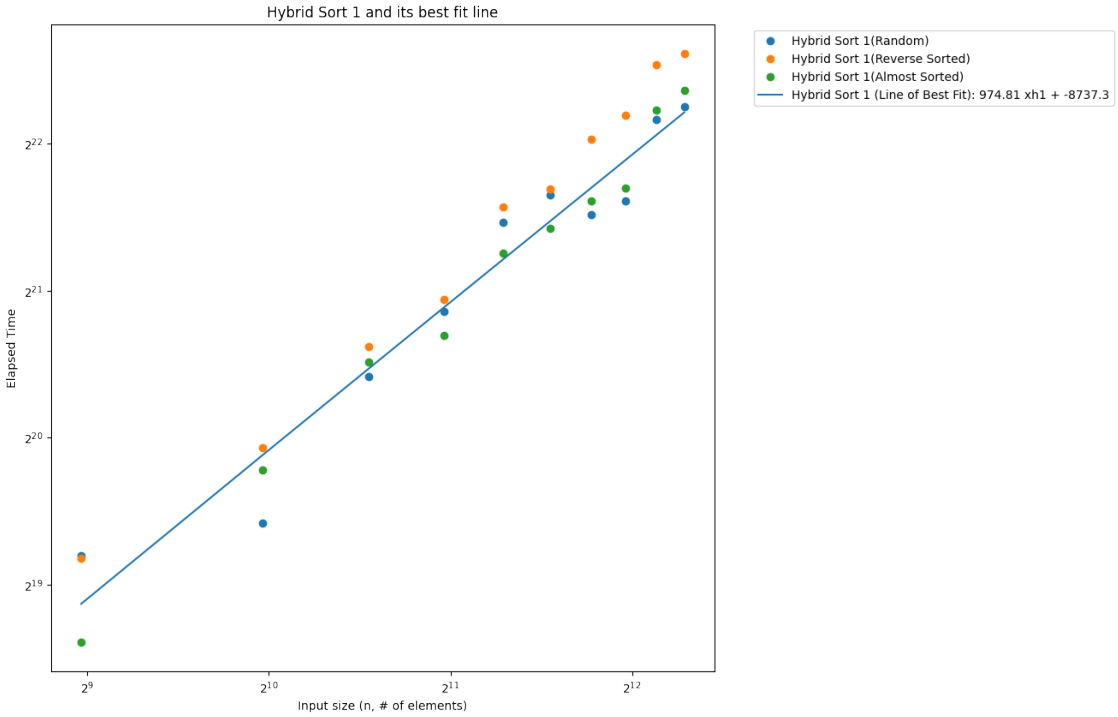
[321]:
```
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("HYBRID SORT 1 log-log plots\n")
plt.loglog(xh1, yh1)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.loglog(xh1_2, yh1_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.loglog(xh1_3, yh1_3)
plt.xlabel("Almost")
plt.tight_layout()
```



4.1.1 Determining Line of Best Fit

[322]:
```
logxh1, logyh1 = np.log(xh1), np.log(yh1)
mh1, bh1 = np.polyfit(xh1, yh1, 1)
fit = np.poly1d((mh1, bh1))
expected_logyh1 = fit(logxh1)
sort_name = "Hybrid Sort 1"
perm_name = "Line of Best Fit"
perm_name2 = "Reverse Input"
perm_name3 = "Almost Sorted Input"
figure(figsize=(10,10),dpi=100)
p = plt.loglog(xh1, yh1, '.', base= 2, markersize = 12,label= 'Hybrid Sort␣
↪1(Random)')
```

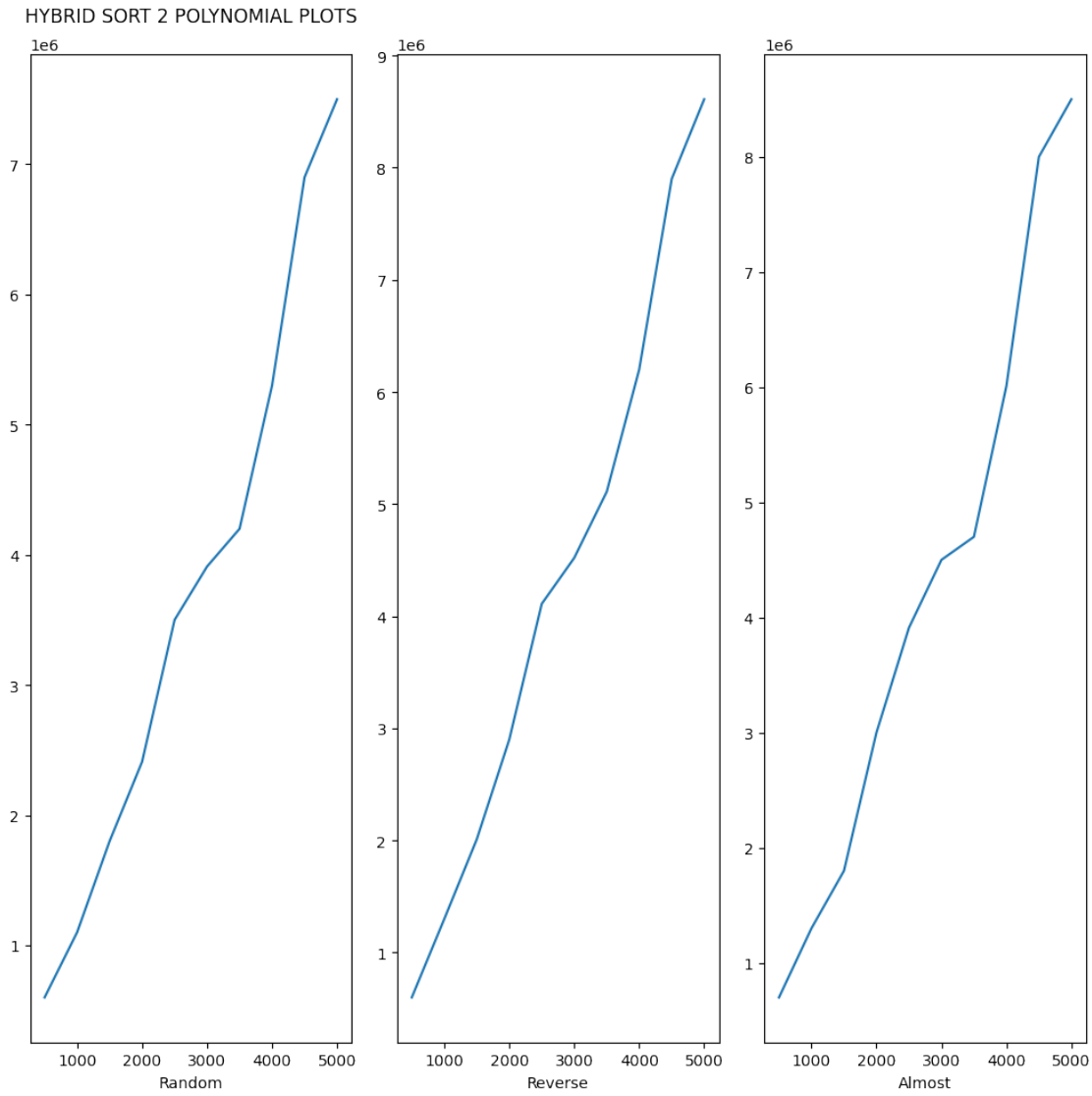
```
p_rev = plt.loglog(xh1_2, yh1_2, '.', base = 2, markersize = 12,label= 'Hybrid␣
 ↪Sort 1(Reverse Sorted)')
p_almost = plt.loglog(xh1_3, yh1_3, '.', base = 2, markersize = 12,label=␣
 ↪'Hybrid Sort 1(Almost Sorted)')
plt.plot(xh1, mh1*xh1 + (bh1), label = f'{sort_name} ({perm_name}): {mh1:0.5}␣
 ↪xh1 + {bh1:.5}',
                   markersize = 6, color = p[-1].get_color())
plt.title("Hybrid Sort 1 and its best fit line")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



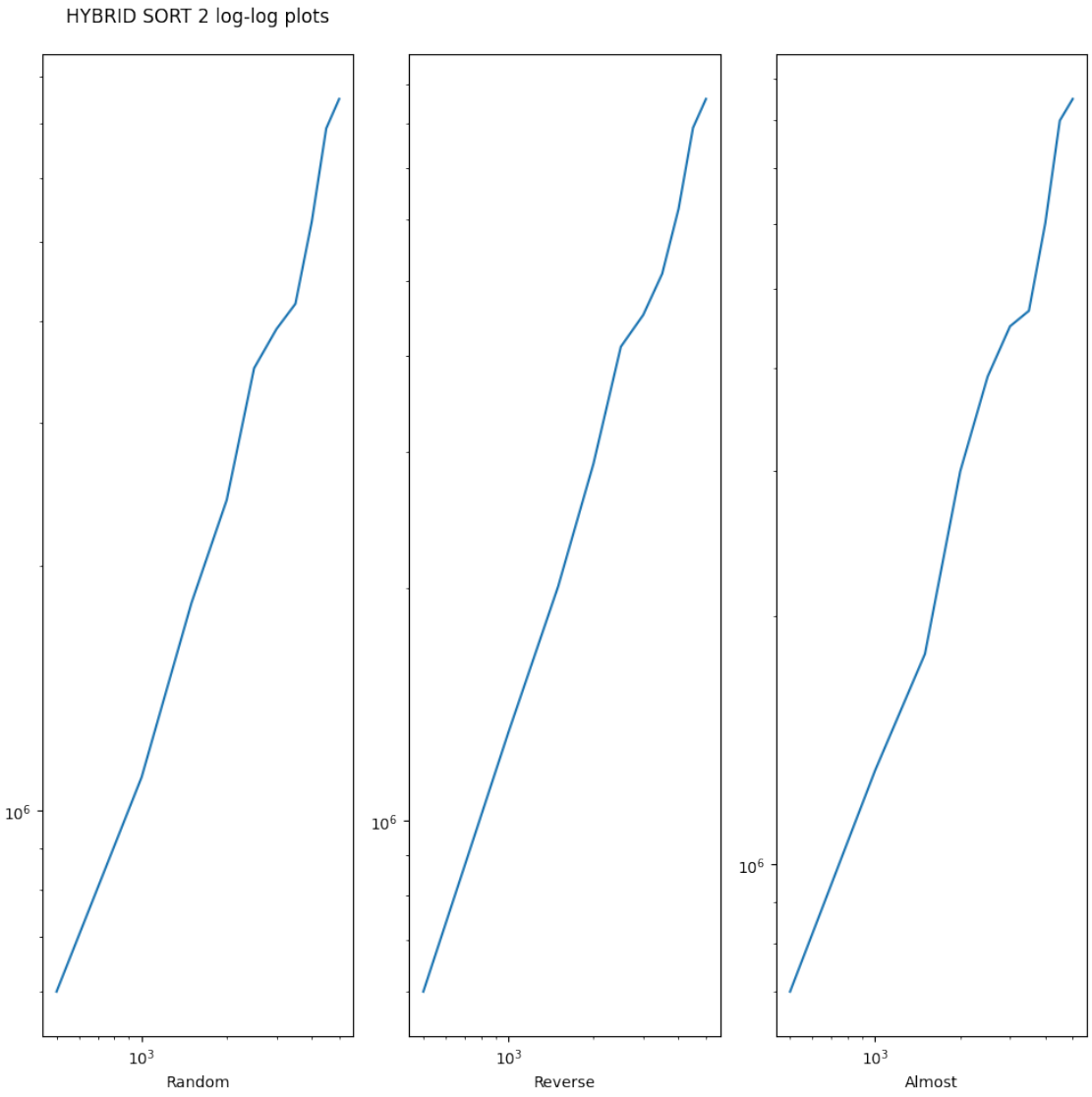
4.2 Hybrid Sort 2: H = n^0.25

[323]:
```
h2f1 = pd.read_csv("HybridSort2_Random.csv")
h2f2 = pd.read_csv("HybridSort2_Reverse.csv")
h2f3 = pd.read_csv("HybridSort2_Almost.csv")
# Xi = INPUT SIZE, Yi = Time (nanoseconds)
#RANDOM DATA
xh2 = h2f1['Size']
yh2 = h2f1['Time']
#REVERSE DATA
```

```
xh2_2 = h2f2['Size']
yh2_2 = h2f2['Time']
#ALMOST SORTED DATA
xh2_3 = h2f3['Size']
yh2_3 = h2f3['Time']
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("HYBRID SORT 2 POLYNOMIAL PLOTS\n")
plt.plot(xh2, yh2)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.plot(xh2_2, yh2_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.plot(xh2_3, yh2_3)
plt.xlabel("Almost")
plt.tight_layout()
```



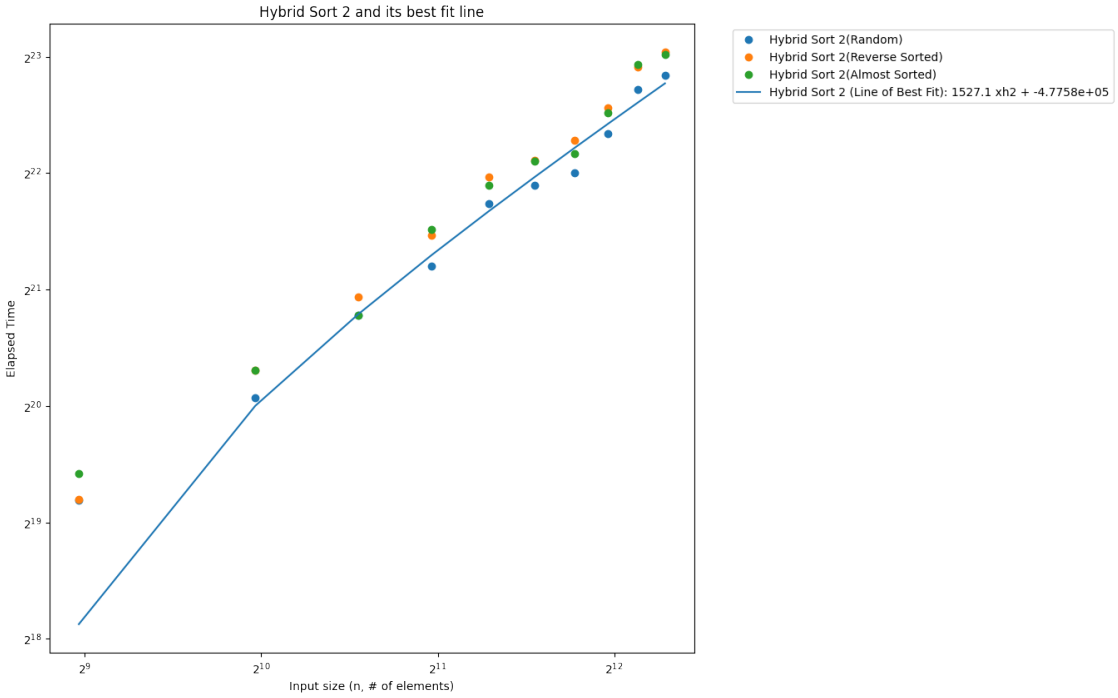
[324]:
```
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("HYBRID SORT 2 log-log plots\n")
plt.loglog(xh2, yh2)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.loglog(xh2_2, yh2_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.loglog(xh2_3, yh2_3)
plt.xlabel("Almost")
plt.tight_layout()
```



4.2.1 Determing line of best fit

[325]:
```
logxh2, logyh2 = np.log(xh2), np.log(yh2)
mh2, bh2 = np.polyfit(xh2, yh2, 1)
fit = np.poly1d((mh2, bh2))
expected_logyh2 = fit(logxh2)
sort_name = "Hybrid Sort 2"
perm_name = "Line of Best Fit"
perm_name2 = "Reverse Input"
perm_name3 = "Almost Sorted Input"
figure(figsize=(10,10),dpi=100)
p = plt.loglog(xh2, yh2, '.', base= 2, markersize = 12,label= 'Hybrid Sort␣
↪2(Random)')
```

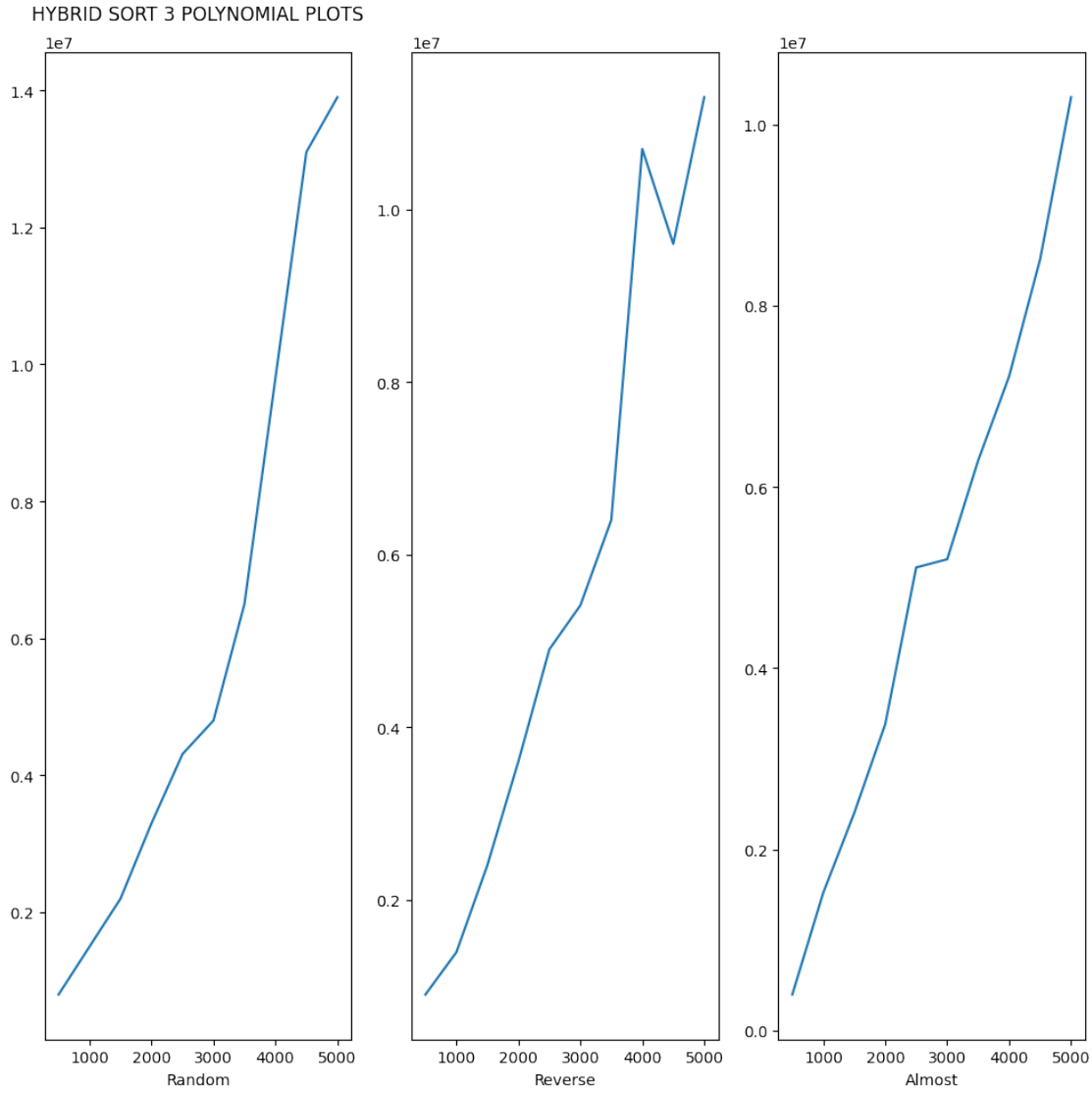
```
p_rev = plt.loglog(xh2_2, yh2_2, '.', base = 2, markersize = 12,label= 'Hybrid␣
↪Sort 2(Reverse Sorted)')
p_almost = plt.loglog(xh2_3, yh2_3, '.', base = 2, markersize = 12,label=␣
↪'Hybrid Sort 2(Almost Sorted)')
plt.plot(xh2, mh2*xh2 + (bh2), label = f'{sort_name} ({perm_name}): {mh2:0.5}␣
↪xh2 + {bh2:.5}',
                    markersize = 6, color = p[-1].get_color())
plt.title("Hybrid Sort 2 and its best fit line")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



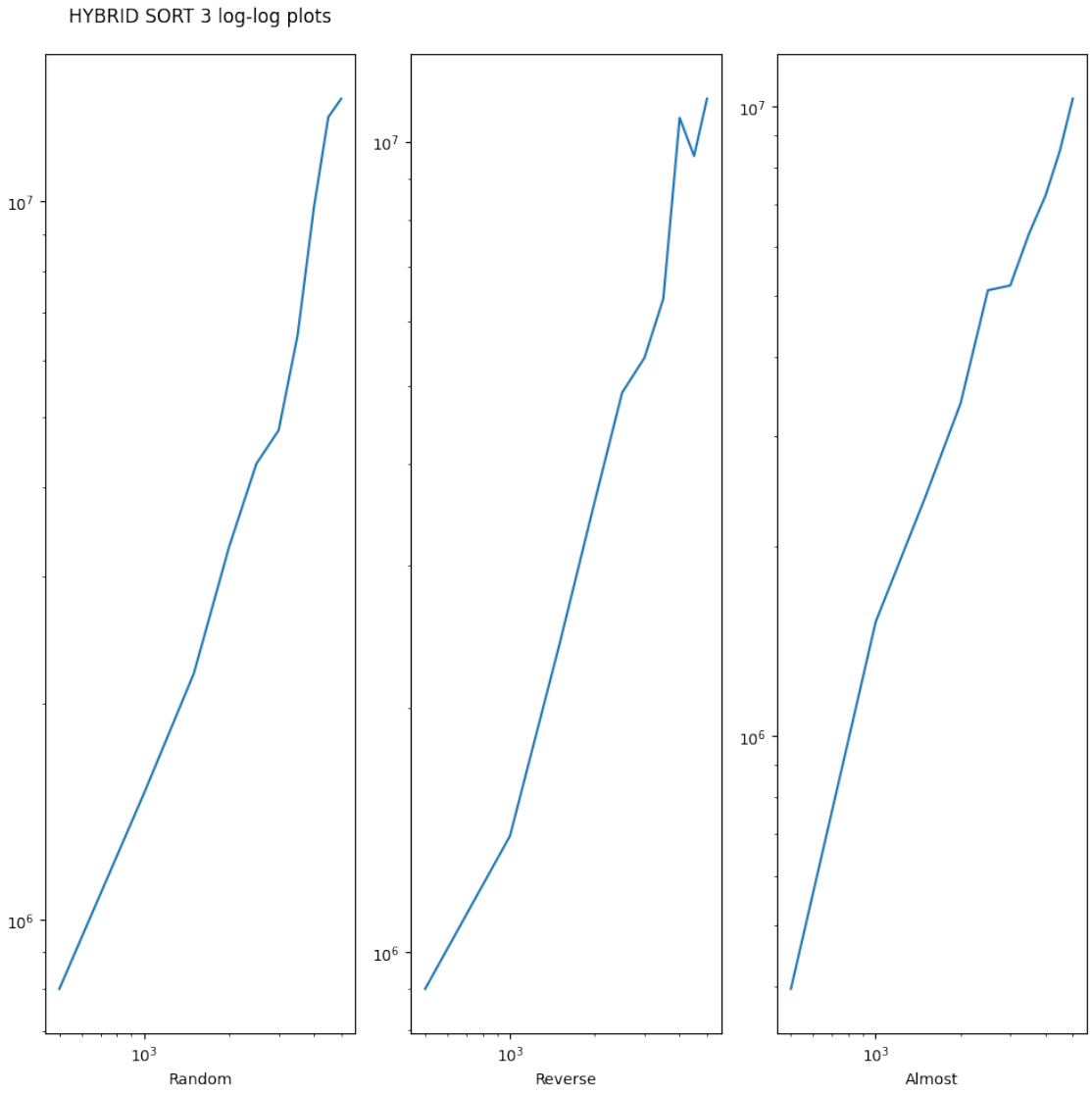
4.3 Hybrid Sort 3: H = n^0.1666

[326]:
```
h3f1 = pd.read_csv("HybridSort3_Random.csv")
h3f2 = pd.read_csv("HybridSort3_Reverse.csv")
h3f3 = pd.read_csv("HybridSort3_Almost.csv")
# Xi = INPUT SIZE, Yi = Time (nanoseconds)
#RANDOM DATA
xh3 = h3f1['Size']
yh3 = h3f1['Time']
#REVERSE DATA
xh3_2 = h3f2['Size']
```

```
yh3_2 = h3f2['Time']
#ALMOST SORTED DATA
xh3_3 = h3f3['Size']
yh3_3 = h3f3['Time']
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("HYBRID SORT 3 POLYNOMIAL PLOTS\n")
plt.plot(xh3, yh3)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.plot(xh3_2, yh3_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.plot(xh3_3, yh3_3)
plt.xlabel("Almost")
plt.tight_layout()
```



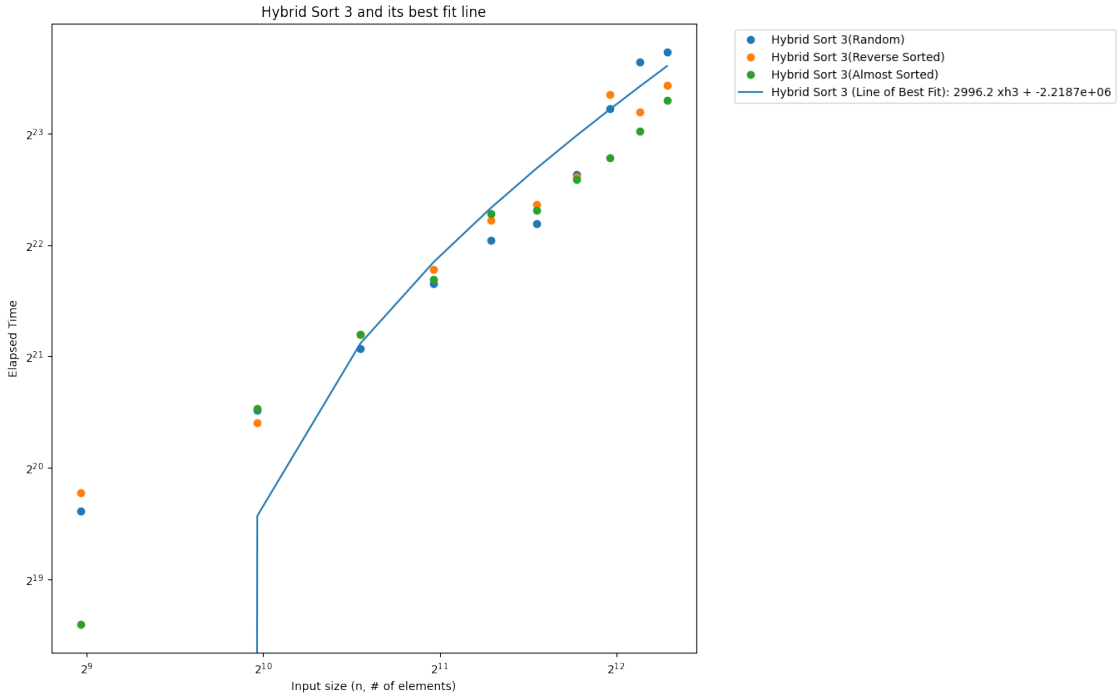
[327]:
```
figure(figsize=(10,10),dpi=100)
plt.subplot(1,3,1)
plt.title("HYBRID SORT 3 log-log plots\n")
plt.loglog(xh3, yh3)
plt.xlabel("Random")
plt.subplot(1,3,2)
plt.loglog(xh3_2, yh3_2)
plt.xlabel("Reverse")
plt.subplot(1,3,3)
plt.loglog(xh3_3, yh3_3)
plt.xlabel("Almost")
plt.tight_layout()
```



4.3.1 Determining line of best fit

[328]:
```
logxh3, logyh3 = np.log(xh3), np.log(yh3)
mh3, bh3 = np.polyfit(xh3, yh3, 1)
fit = np.poly1d((mh3, bh3))
expected_logyh3 = fit(logxh3)
sort_name = "Hybrid Sort 3"
perm_name = "Line of Best Fit"
perm_name2 = "Reverse Input"
perm_name3 = "Almost Sorted Input"
figure(figsize=(10,10),dpi=100)
p = plt.loglog(xh3, yh3, '.', base= 2, markersize = 12,label= 'Hybrid Sort␣
↪3(Random)')
```

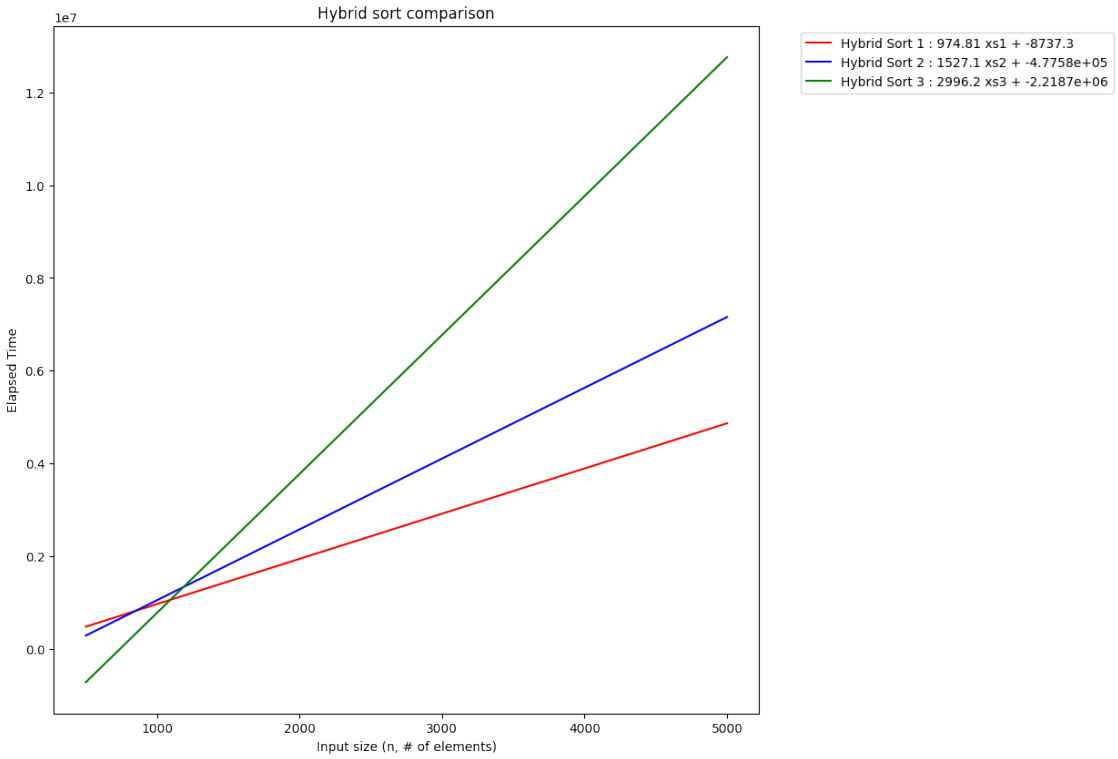
```
p_rev = plt.loglog(xh3_2, yh3_2, '.', base = 2, markersize = 12,label= 'Hybrid␣
↪Sort 3(Reverse Sorted)')
p_almost = plt.loglog(xh3_3, yh3_3, '.', base = 2, markersize = 12,label=␣
↪'Hybrid Sort 3(Almost Sorted)')
plt.plot(xh3, mh3*xh3 + (bh3), label = f'{sort_name} ({perm_name}): {mh3:0.5}␣
↪xh3 + {bh3:.5}',
                    markersize = 6, color = p[-1].get_color())
plt.title("Hybrid Sort 3 and its best fit line")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



4.4 Comparing Hybrid Sorts

[329]:
```
figure(figsize=(10,10),dpi=100)
h1 = "Hybrid Sort 1"
h2 = "Hybrid Sort 2"
h3 = "Hybrid Sort 3"
plt.plot(xh1, mh1*xh1 + (bh1), label = f'{h1} : {mh1:0.5} xs1 + {bh1:.
↪5}',markersize = 6, color = "red")
plt.plot(xh2, mh2*xh2 + (bh2), label = f'{h2} : {mh2:0.5} xs2 + {bh2:.
↪5}',markersize = 6, color = "blue")
```

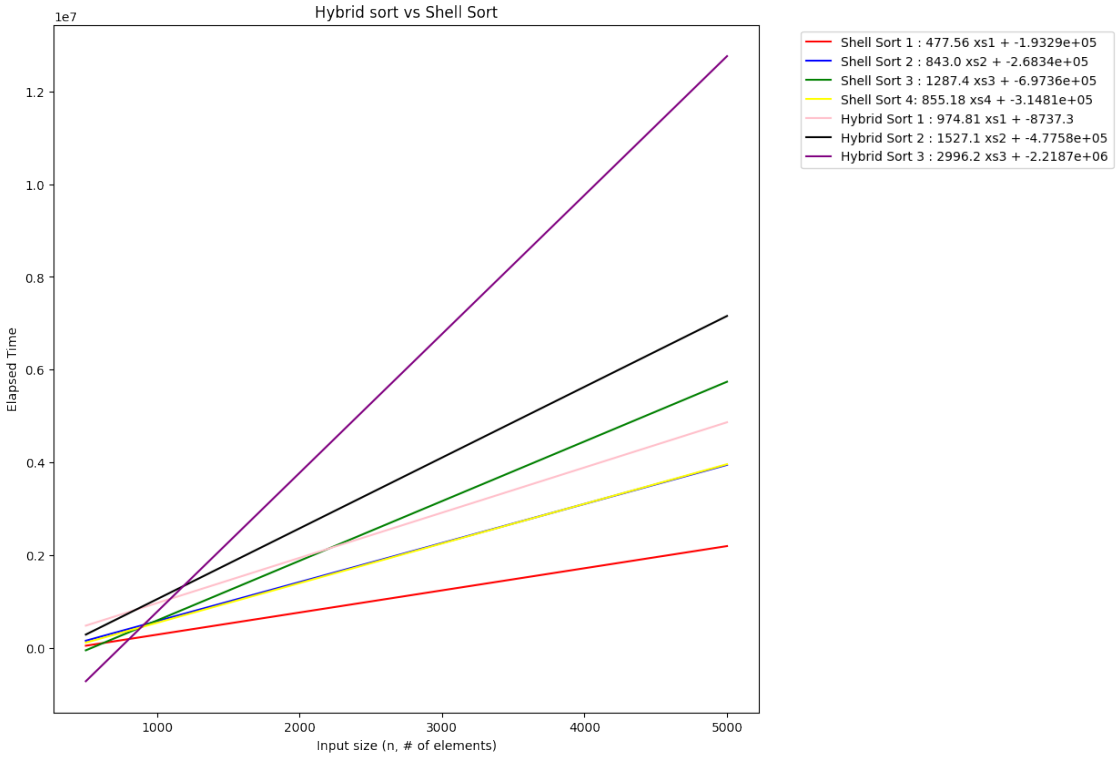
```
plt.plot(xh3, mh3*xh3 + (bh3), label = f'{h3} : {mh3:0.5} xs3 + {bh3:.
 ↪5}',markersize = 6, color = "green")
plt.title("Hybrid sort comparison")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



5 Hybrid vs Shell Sort

[331]:
```
figure(figsize=(10,10),dpi=100)
s1 = "Shell Sort 1"
s2 = "Shell Sort 2"
s3 = "Shell Sort 3"
s4 = "Shell Sort 4"
plt.plot(xs1, ms1*xs1 + (bs1), label = f'{s1} : {ms1:0.5} xs1 + {bs1:.
 ↪5}',markersize = 6, color = "red")
plt.plot(xs2, ms2*xs2 + (bs2), label = f'{s2} : {ms2:0.5} xs2 + {bs2:.
 ↪5}',markersize = 6, color = "blue")
plt.plot(xs3, ms3*xs3 + (bs3), label = f'{s3} : {ms3:0.5} xs3 + {bs3:.
 ↪5}',markersize = 6, color = "green")
```

```
plt.plot(xs4, ms4*xs4 + (bs4), label = f'{s4}: {ms4:0.5} xs4 + {bs4:.5}',␣
 ↪markersize = 6, color = "yellow")

h1 = "Hybrid Sort 1"
h2 = "Hybrid Sort 2"
h3 = "Hybrid Sort 3"
plt.plot(xh1, mh1*xh1 + (bh1), label = f'{h1} : {mh1:0.5} xs1 + {bh1:.
 ↪5}',markersize = 6, color = "pink")
plt.plot(xh2, mh2*xh2 + (bh2), label = f'{h2} : {mh2:0.5} xs2 + {bh2:.
 ↪5}',markersize = 6, color = "black")
plt.plot(xh3, mh3*xh3 + (bh3), label = f'{h3} : {mh3:0.5} xs3 + {bh3:.
 ↪5}',markersize = 6, color = "purple")

plt.title("Hybrid sort vs Shell Sort")
plt.xlabel('Input size (n, # of elements)')
plt.ylabel('Elapsed Time')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



6 Analysis Summary

Input Data/Distrubutions - For this project, each experiment was conducted for each algorihtm as follows: - For a certain input size and permutation type, we run the algorithm 10 times and average the performance times and record it - This process is repeated for every size (500 - 5000) and every type of permutation (random,reverse,almost)

Insertion Sort - From the plots above we can see that Insertion sort does very well for smaller datasets and for some cases the order of elements does not matter as the random and reverse permutation result in the same convergence. However, there are cases where Insertion sort does very poorly for almost sorted arrays but sometimes also does much better. So this is not a very stable sorting algorithm with the worst case being O(n^2) due to the input size and permutation senstivity . But it does have a best case of O(n) for some instances.

Merge Sort - This algorithm performs somewhat similar to Insertion sort from the comparisons, for smaller inputs we would prefer Insertion sort but as the input sizes increase drastically insertion sort starts performing worse while merge does better. But unlike Insertion sort, the order of the elements really does not matter here as it has a consistent performance for all the different permutations. So this is a more stable algorithm than Insertion sort so this would be the better algorithm for larger datasets with the worst, average and best case being O(n log n).

Shell Sort - This algorithm also has a similar performance to insertion sort but it varies quite a bit dependning upon the gap sequence. The 2nd and 4th versions of the gap sequence in the shell sort algorithm perform almost exactly the same as we can see in the graph the lines overlap. The 3rd version of the gap sequence is clearly the worst because for larger input sizes the performance drastically reduces. So if we were to use Shell Sort, the original gap sequence would be the best choice because it's not as sensitive to the input size or permutation so it's a more stable algorithm. However, because it follows insertion sort it has a similar worst case time complexity of O(n^2) but it does have an average case of O(n log n) so analyticallly this algorithm would still be better than the regular insertion sort.

Hybrid Sort - The hybrid sorting algorithm is derived from the merge and insertion sort algorithm so a lot of the characteristics from both algorithms apply to this one as well. Since this algorithm uses merge, it has an average case of O(n log n) from the merge sort algorithm. But we also use insertion sort so it can also have the best case performance of O(n). Although this algorithm is not too sensitive to the permutation type but the size of the data definitely factors in quite a bit. But combining both the techniques together definitely improves the performance time and stability.

6.1 Conclusion

From all the above algorithms, the Shell Sorting algorithm with the original gap sequence performs the best however the Hybrid Sorting algorithm is also a good and more stable algorithm for sorting. In my opinion, the Hybrid Sorting algorithm is the winner because stability in an algorithm has a higher prioirty than performance time. However we can further improve the Hybrid Sorting algorithm by diving the arrays into sub-arrays that are already sorted within the array, this will further reduce the time for this algorithm as insertion sort will only take O(n) time and since we are diving the array into many parts n will be relatively small.