F(n)

For the data set of f(n), the trend line looks like linear. From the chart below we can see that the R2 is 0.9992 for the linear equation. It’s already a very good result.

Therefore, we can conclude that f(n) ~ O(n).

Chart, line chart

Description automatically generated

G(n)

For the data set of g(n), the trend line looks not linear. From the chart below we can see that the R2 is only 0.7974 for the linear model.

Chart, scatter chart

Description automatically generated

With Polynomial model, R2 is 0.9883 for quadratic and 0.9999 for cubic.

Chart

Description automatically generated

Chart

Description automatically generated

Therefore, we can conclude g(n) ~ O(n3)

H(n)

For the data set of h(n), the trend line looks like linear, but it starts to deviate when the n is greater than 400,000.

Chart, line chart

Description automatically generated

Is it possible that it’s actually O(nlogn)? We created a new series by dividing the runtime h(n) by n, multiplying by 500,000, and fitting in a logarithmic model. The R2 is 0.9612. If h(n) ~ O(n), h(n)/n should be a constant; if h(n) ~ O(n2), h(n)/n should be linear. It seems O(nlogn) is better to describe h(n).

Therefore, we conclude that the best estimate with the given data is h(n) ~ O(nlogn)

Chart

Description automatically generated

**copy()**

**1. A description of how you chose to test the copy method.**

‘copy()’ method is copying an object. We copied a list of numbers. Depending on the length of a list, the method’s runtime is different. So we tested the method by increasing the length of a list by 10 at a time until it reached about 5000. And in order to obtain precision, we used average runtime data, which is 500 cycles per point.

\* This is the python code to test copy() method.

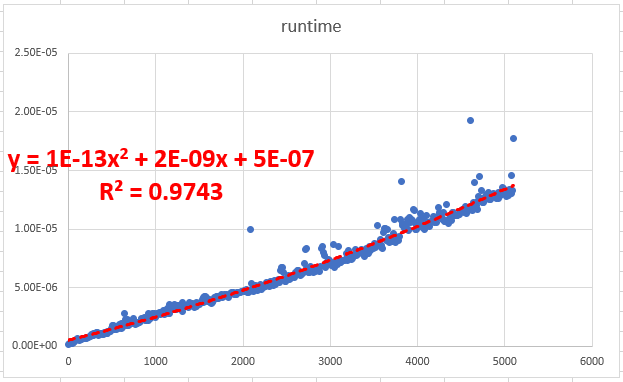
def create\_random\_list(list\_length, upper\_bound):  
 return [random.randint(0, upper\_bound) for \_ in range(list\_length)]  
  
def timetest(runs, list\_length, upper\_bound):  
 total = 0  
 for \_ in range(runs):  
 list1 = create\_random\_list(list\_length, upper\_bound)  
 start = timeit.default\_timer()  
 list1.copy()  
 end = timeit.default\_timer()  
 total += end - start  
 return total/runs  
  
for i in range(1, 10000, 10):  
 print(i, timetest(500, i, i+1))

**2. Observations and conclusions made from your experiments.**

We tested several times to obtain accurate data. Depending on the length of list, the number of average runtime cycle, and the size of increment, the result was different. But we tested several times, and finally we were able to get the best result.

**3. Evidence to back up your observations and conclusions (you should include at least one graph here).**

This graph shows that copy() method’s runtime increases based on x^2.



**4. An explanation as to why copy() performs in the manner it does.**

There are three kinds of copies in python, which are ‘mutate’, ‘shallow copy’, and ‘deep copy’. We thought at first it will just copy the address. At first we expected the copy() method’s runtime will be a constant. In that case regardless of the length of a list, runtime should be same because it will just copy the address. But it was wrong. Because Copy() method was shallow copy, and runtime increased based on n^2, which means . This is because the length of list increased from 1 to 5000 by 10 at a time, which means . Therefore, we conclude that the best estimate with the given data is copy() ~ O(n^2).