Empirical Analysis Report

Ayesh Reddy Kothinti
Sai Preetham reddy Alladu
Varun sai chilkuri

Title: Empirical Analysis of Matrix multiplication Algorithms

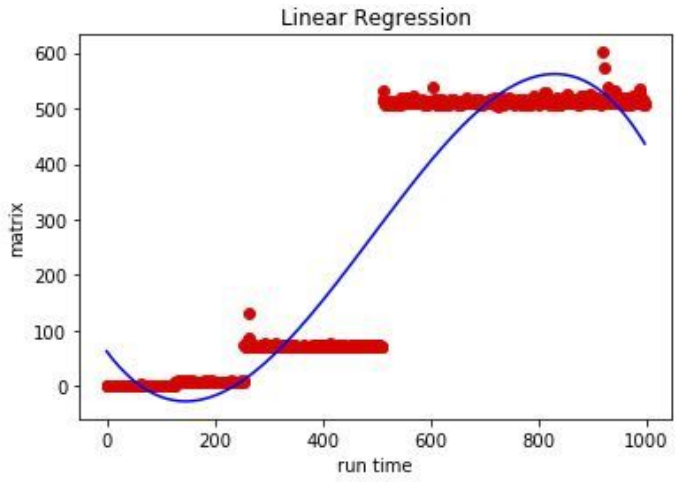
performance analysis of matrix multiplication using Strassen's matrix multiplication and normal matrix multiplication(n3)

Expectation:

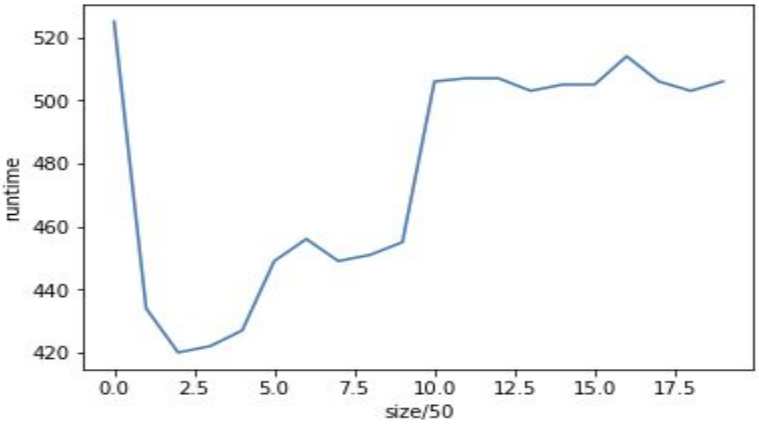
To find a break even point at which strassen's algorithm outperforms traditional n3 matrix multiplication algorithm by performing a dry run on square matrices of size 1 to 1000 rows and evaluating the runtimes respectively for comparison, thereby plotting the data points obtained using polynomial regression technique.

Observation:

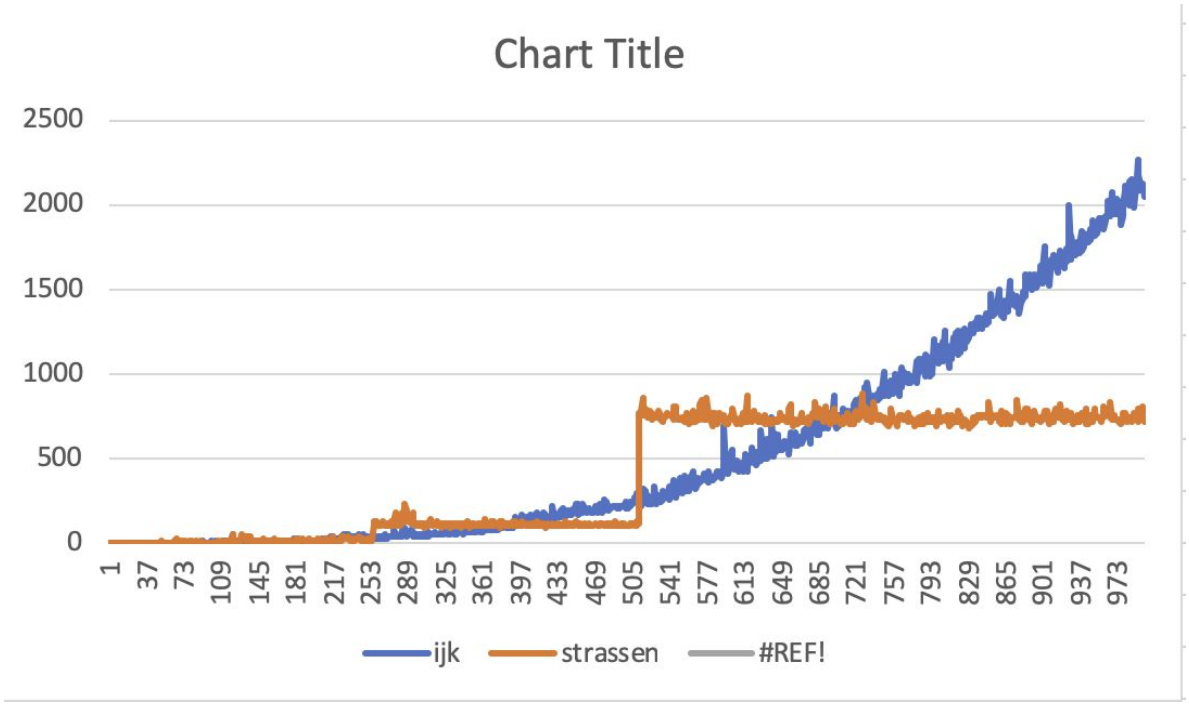
From the results obtained it is visible that for a particular size of a matrix, strassen's matrix multiplication algorithm completes the job in comparatively lesser time than the other Algorithm.



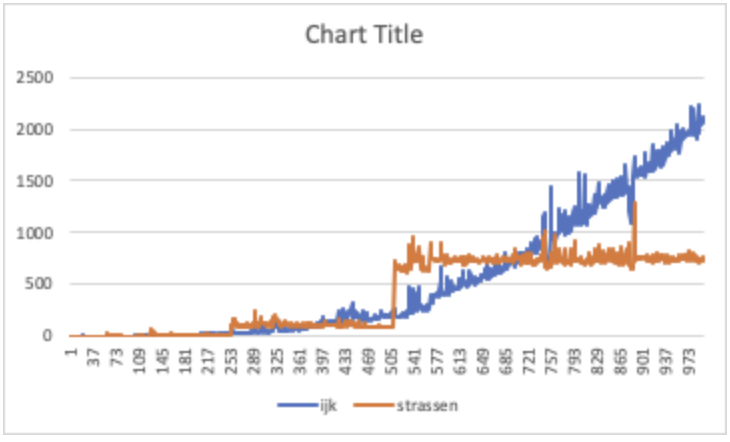
Regression analysis to plot the Points in strassen's algorithm analysis.



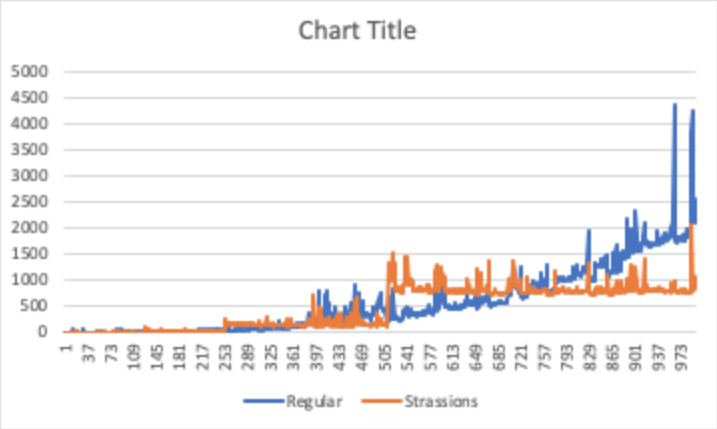
Strassen's algorithm for matrix size 1000 vs various leaf sizes.At leaf size 150 strassen's performs at its highest efficiency.



For a leaf size of 150 strassen's algorithm outperforms traditional matrix multiplication for the matrix size sections 400-505 and after 721 onwards continuously. The above graph summarises our findings about the performance comparison between two algorithms for matrix multiplication.



For a leaf size of 200 strassen's algorithm outperforms traditional matrix multiplication for the matrix size sections 397-505 and after 710 onwards continuously.



For a leaf size of 300 strassen's algorithm outperforms traditional matrix multiplication for the matrix size sections 397-510 and after 726 onwards continuously.

Console output:

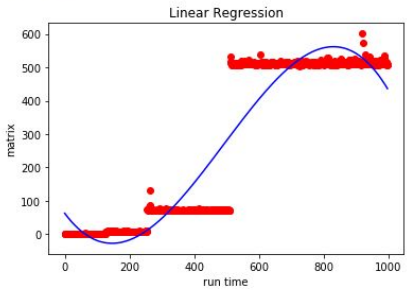
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

58]:
```
X=x.reshape(-1,1)
```

51]:
```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=4)
X_poly = poly_reg.fit_transform(X)
pol_reg = LinearRegression()
pol_reg.fit(X_poly, y)
plt.savefig('fitting_a_curve.png')
```

51]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

52]:
```
def viz_polymonial():
    plt.scatter(X, y, color='red')
    plt.plot(X, pol_reg.predict(poly_reg.fit_transform(X)), color='blue')
    plt.title('Linear Regression')
    plt.xlabel('run time')
    plt.ylabel('matrix')
    plt.show()
    return
viz_polymonial()
```



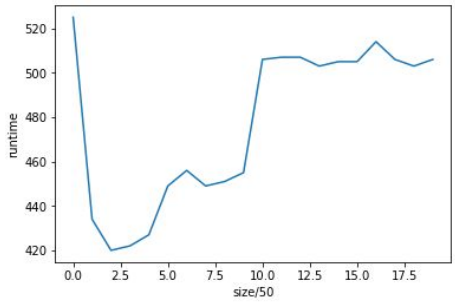
```
[6]: import matplotlib.pyplot as plt
     import pandas as pd
     import numpy as np

     data = pd.read_csv("test_for_leaf_size.csv")
```

```
[13]: y = data.iloc[:, 1].values
      x = np.array(list(range(50, 1000, 50)))

      plt.plot(y)

      plt.xlabel("size/50")
      plt.ylabel("runtime")
      plt.show()
      plt.savefig('best_leaf.jpg')
```



Through repeated runs we observe that the observations align and show similar Result.

Conclusion:

From the observation for the range of matrix size(n) in the range {[400-505],[720,∞]} traditional matrix multiplication performs well with a better run time and later on with strassens gives a better run time. The optimal size of a matrix for strassens is around 720 for a leaf size of 150.