ECE374 Fall2020

Lab4: Divide and Conquer

Name: Zhang Yichi 3180111309

Oct. 27th 2020

1 Introduction

In this lab, we will do two examples that show the Divide and Conquer idea, binary search and the Strassen algorithm. In both algorithms, a problem is divided into multiple smaller and easier problems but not affecting the final result.

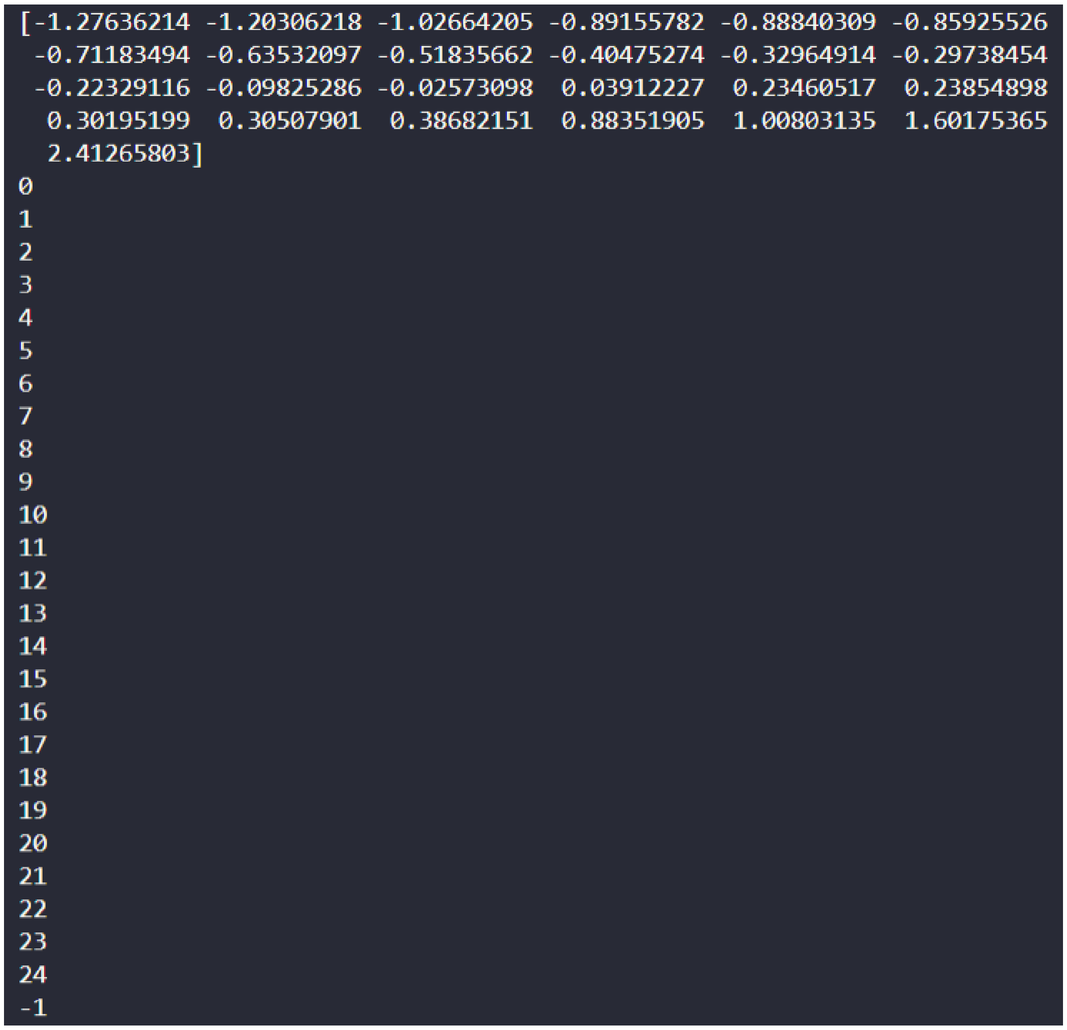
2 Python Code for Binary Search

```python
def binary_search(vec,i,j,v):
    if i>j:
        return -1
    mid = (j-i)//2+i
    if vec[mid] == v:
        return mid
    elif vec[mid] > v:
        return binary_search(vec,i,mid,v)
    else:
        return binary_search(vec,mid+1,j,v)
```

To find an element in a sorted list, each time we only check the element in the middle. If the element we want is smaller, then it must be ahead of the middle element. Otherwise, it must be behind the middle element. In this way, we can get rid of half of the elements for each round. Thus, the time complexity for this binary search is O(lgn).

3 Test Example for Binary Search

```
vec = np.random.randn(25)
mergesort(vec,5)
print(vec)
for i in vec:
    print(binary_search(vec,0,len(vec)-1,i))
print(binary_search(vec,0,len(vec)-1,99))
```



```
[-1.27636214 -1.20306218 -1.02664205 -0.89155782 -0.88840309 -0.85925526
 -0.71183494 -0.63532097 -0.51835662 -0.40475274 -0.32964914 -0.29738454
 -0.22329116 -0.09825286 -0.02573098  0.03912227  0.23460517  0.23854898
  0.30195199  0.30507901  0.38682151  0.88351905  1.00803135  1.60175365
  2.41265803]
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
-1
```

Figure 1: binary search test

In this test example, we first randomly generate 25 numbers, sort them using the merge sort we learned in the previous lab. To test the binary search, we run the function to search for each element in the list and one element that is definitely not in the list. The outputs are 0 to 24 and a -1 which is exactly what we expected.

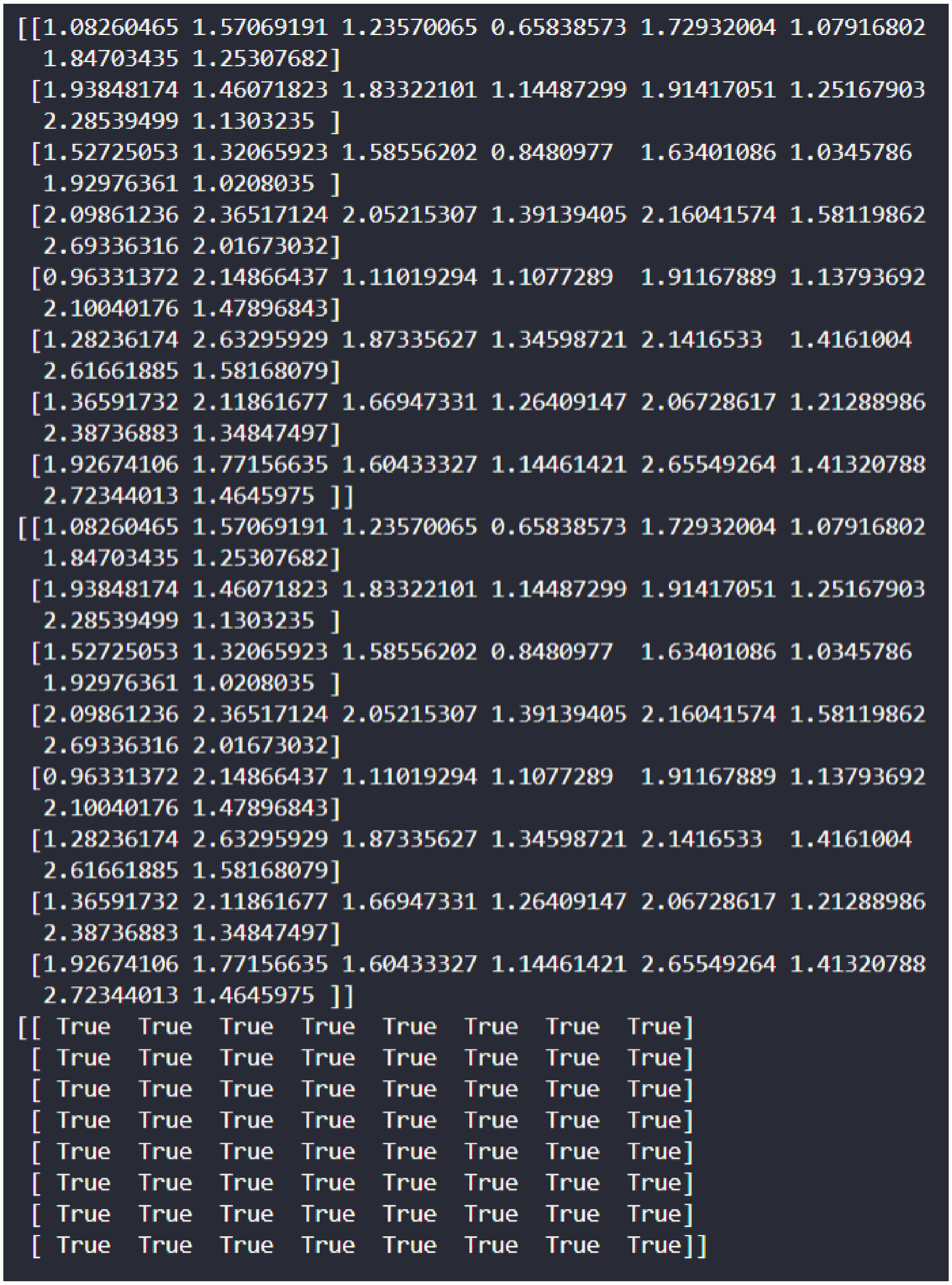
4 Python Code for Strassen Algorithm

```
def matrix_divide(A):
    midr = len(A) // 2
    midc = len(A[0]) // 2
    a = A[:midr,:midc]
    b = A[:midr,midc:]
    c = A[midr:,:midc]
    d = A[midr:,midc:]
    return a,b,c,d

def matrix_merge(a,b,c,d):
    r1 = len(a)
    r2 = len(c)
    c1 = len(a[0])
    c2 = len(b[0])
    A = np.zeros((r1+r2,c1+c2))
    A[:r1,:c1] = a
    A[:r1,c1:] = b
    A[r1:,:c1] = c
    A[r1:,c1:] = d
    return A

def Strassen(A,B):
    if len(A) == 1:
        return np.array([A*B])
    else:
        a,b,c,d = matrix_divide(A)
        e,f,g,h = matrix_divide(B)

        P11 = f-h
        P21 = a+b
        P31 = c+d
        P41 = g-e
        P51 = a+d
        P52 = e+h
        P61 = b-d
        P62 = g+h
        P71 = a-c
        P72 = e+f

        P1 = Strassen(a,P11)
        P2 = Strassen(P21,h)
        P3 = Strassen(P31,e)
        P4 = Strassen(d,P41)
        P5 = Strassen(P51,P52)
        P6 = Strassen(P61,P62)
        P7 = Strassen(P71,P72)

        r = P5+P4-P2+P6
        s = P1+P2
        t = P3+P4
        u = P5+P1-P3-P7

        return matrix_merge(r,s,t,u)
```

Instead of doing multiplication one element by one element in the original matrix, Strassen algorithm divide the matrix into four pieces. With calculations to the matrix blocks, we can obtain four new matrix blocks and merge them to the result. These calculations only involve 7 matrix multiplications which is one less than 8 (do the multiplication directly), consequently, reduce the time complexity from $O(n^{lg8}) = O(n^3)$ to $O(n^{lg7}) = O(n^{2.807})$

5 Test Example for Strassen Algorithm

```
A = np.random.rand(8,8)
B = np.random.rand(8,8)
res_np = np.matmul(A,B)
res_Strassen = Strassen(A,B)
print(res_np)
print(res_Strassen)
print(abs(res_np-res_Strassen)<0.0000001)
```



```
[[1.08260465 1.57069191 1.23570065 0.65838573 1.72932004 1.07916802
  1.84703435 1.25307682]
 [1.93848174 1.46071823 1.83322101 1.14487299 1.91417051 1.25167903
  2.28539499 1.1303235 ]
 [1.52725053 1.32065923 1.58556202 0.8480977  1.63401086 1.0345786
  1.92976361 1.0208035 ]
 [2.09861236 2.36517124 2.05215307 1.39139405 2.16041574 1.58119862
  2.69336316 2.01673032]
 [0.96331372 2.14866437 1.11019294 1.1077289  1.91167889 1.13793692
  2.10040176 1.47896843]
 [1.28236174 2.63295929 1.87335627 1.34598721 2.1416533  1.4161004
  2.61661885 1.58168079]
 [1.36591732 2.11861677 1.66947331 1.26409147 2.06728617 1.21288986
  2.38736883 1.34847497]
 [1.92674106 1.77156635 1.60433327 1.14461421 2.65549264 1.41320788
  2.72344013 1.4645975 ]]
[[1.08260465 1.57069191 1.23570065 0.65838573 1.72932004 1.07916802
  1.84703435 1.25307682]
 [1.93848174 1.46071823 1.83322101 1.14487299 1.91417051 1.25167903
  2.28539499 1.1303235 ]
 [1.52725053 1.32065923 1.58556202 0.8480977  1.63401086 1.0345786
  1.92976361 1.0208035 ]
 [2.09861236 2.36517124 2.05215307 1.39139405 2.16041574 1.58119862
  2.69336316 2.01673032]
 [0.96331372 2.14866437 1.11019294 1.1077289  1.91167889 1.13793692
  2.10040176 1.47896843]
 [1.28236174 2.63295929 1.87335627 1.34598721 2.1416533  1.4161004
  2.61661885 1.58168079]
 [1.36591732 2.11861677 1.66947331 1.26409147 2.06728617 1.21288986
  2.38736883 1.34847497]
 [1.92674106 1.77156635 1.60433327 1.14461421 2.65549264 1.41320788
  2.72344013 1.4645975 ]]
[[ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]]
```

Figure 2: Strassen algorithm test

As this algorithm only takes $n \times n$ matrices where $n = 2^x, x \in N$, so for the test, we randomly generate two 8×8 matrices A and B. Use numpy and Strassen algorithm to calculate matrix multiplication AB and get two results. If the algorithm works well, the answer should be the same as what we got from numpy. Due to the floating numbers' error, we use absolute difference to check if they are the same. As we can see, the difference is smaller than 0.0000001. From that, we conclude the Strassen algorithm we derive works well.