

TOPIC 3 - DIVIDE & CONQUER

Q1. Merge Sort

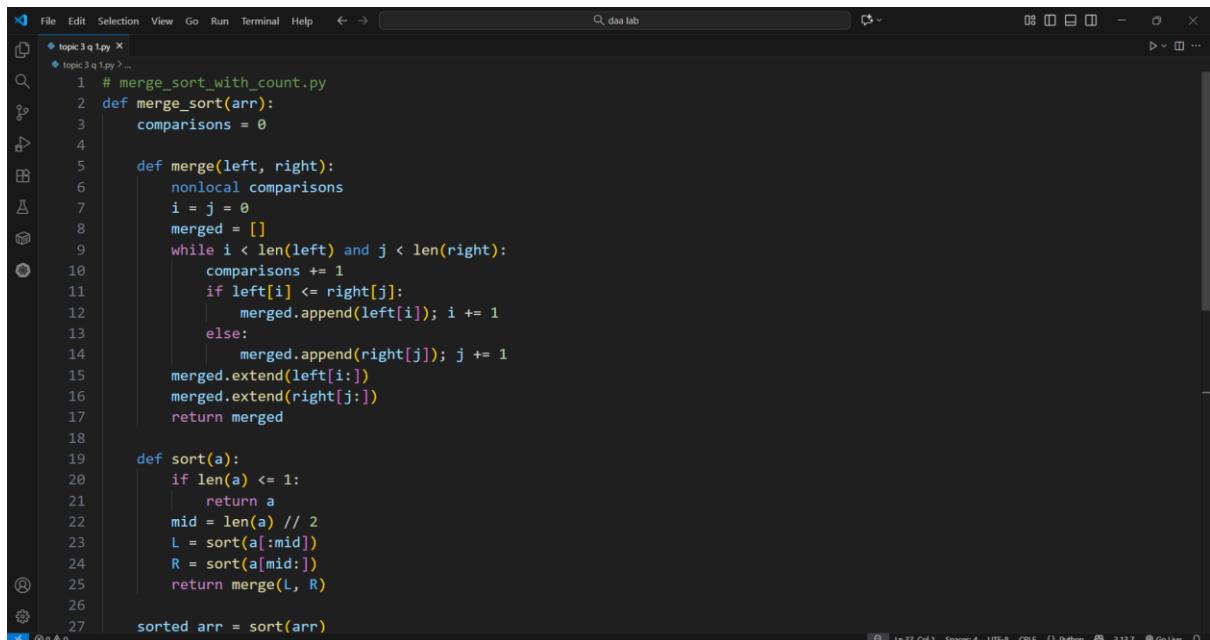
Aim:

To sort an array using divide-and-conquer (Merge Sort) technique.

Algorithm:

1. Divide array into halves.
2. Recursively sort each half.
3. Merge two sorted halves.
4. Compare elements while merging.
5. Return merged sorted array.

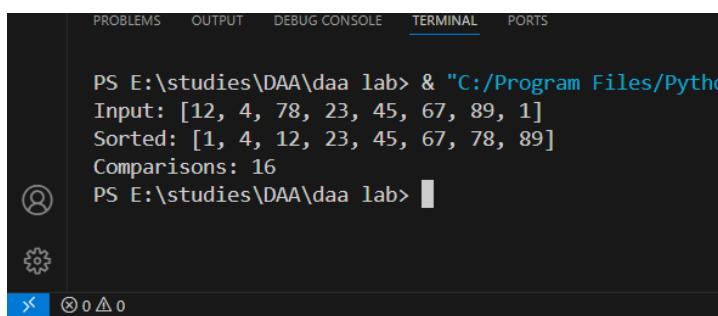
Code:



```
topic 3 q 1.py
topic 3 q 1.py ...
1 # merge_sort_with_count.py
2 def merge_sort(arr):
3     comparisons = 0
4
5     def merge(left, right):
6         nonlocal comparisons
7         i = j = 0
8         merged = []
9
10        while i < len(left) and j < len(right):
11            comparisons += 1
12            if left[i] <= right[j]:
13                merged.append(left[i]); i += 1
14            else:
15                merged.append(right[j]); j += 1
16        merged.extend(left[i:])
17        merged.extend(right[j:])
18    return merged
19
20    def sort(a):
21        if len(a) <= 1:
22            return a
23        mid = len(a) // 2
24        L = sort(a[:mid])
25        R = sort(a[mid:])
26        return merge(L, R)
27
sorted arr = sort(arr)
```

Input: [12, 4, 78, 23, 45, 67, 89, 1]

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS E:\studies\DAA\daa lab> & "C:/Program Files/Python
Input: [12, 4, 78, 23, 45, 67, 89, 1]
Sorted: [1, 4, 12, 23, 45, 67, 78, 89]
Comparisons: 16
PS E:\studies\DAA\daa lab>
```

Result: Array sorted successfully via merge sort.

Q2. Quick Sort

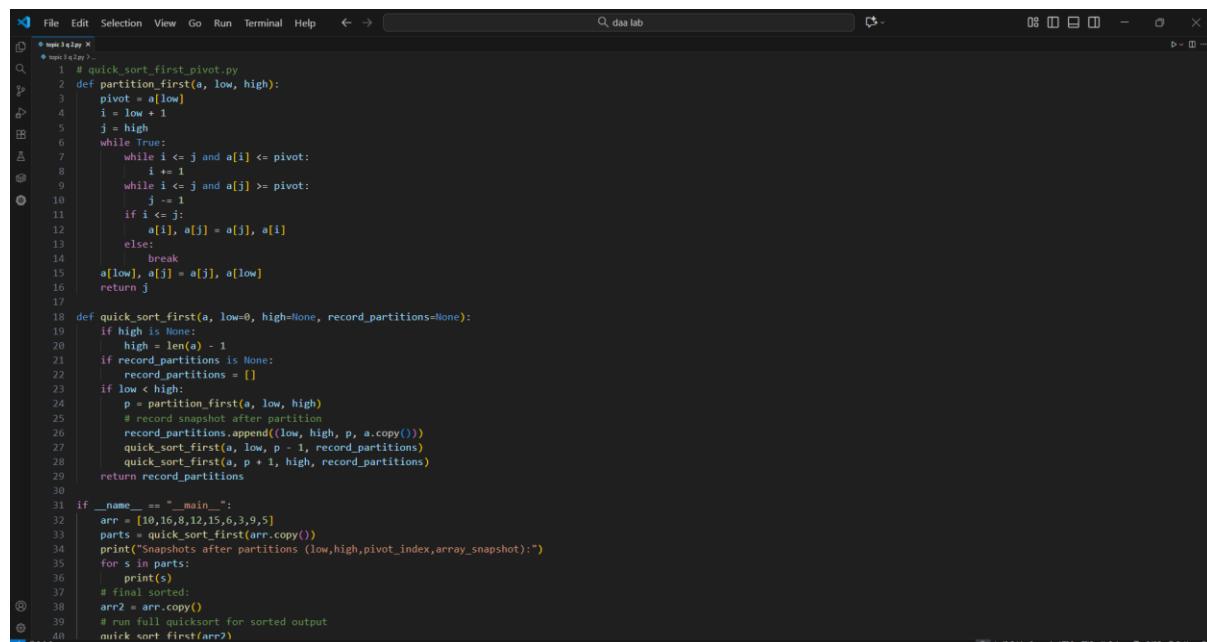
Aim:

To sort numbers using quick sort by selecting a pivot and partitioning recursively.

Algorithm:

1. Choose pivot (first or middle element).
2. Partition array into < pivot and > pivot groups.
3. Recursively sort subarrays.
4. Combine results.
5. Return sorted list.

Code:



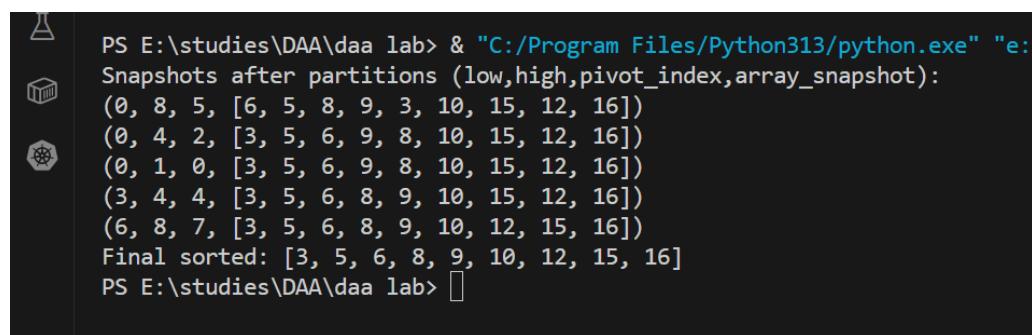
```
# quick sort first pivot
def partition_first(a, low, high):
    pivot = a[low]
    i = low + 1
    j = high
    while True:
        while i <= j and a[i] <= pivot:
            i += 1
        while i <= j and a[j] >= pivot:
            j -= 1
        if i <= j:
            a[i], a[j] = a[j], a[i]
        else:
            break
    a[low], a[j] = a[j], a[low]
    return j

def quick_sort_first(a, low=0, high=None, record_partitions=None):
    if high is None:
        high = len(a) - 1
    if record_partitions is None:
        record_partitions = []
    if low < high:
        p = partition_first(a, low, high)
        # record snapshot after partition
        record_partitions.append((low, high, p, a.copy()))
        quick_sort_first(a, low, p - 1, record_partitions)
        quick_sort_first(a, p + 1, high, record_partitions)
    return record_partitions

if __name__ == "__main__":
    arr = [10, 16, 8, 12, 15, 6, 3, 9, 5]
    parts = quick_sort_first(arr.copy())
    print("Snapshots after partitions (low,high,pivot_index,array_snapshot):")
    for s in parts:
        print(s)
    # final sorted:
    arr2 = arr.copy()
    quick_sort_first(arr2)
    print("Final sorted: [", arr2, "]")
```

Input: [10,16,8,12,15,6,3,9,5]

Output:



```
PS E:\studies\DAAl\daa lab> & "C:/Program Files/Python313/python.exe" "e:/
Solutions after partitions (low,high,pivot_index,array_snapshot):
(0, 8, 5, [6, 5, 8, 9, 3, 10, 15, 12, 16])
(0, 4, 2, [3, 5, 6, 9, 8, 10, 15, 12, 16])
(0, 1, 0, [3, 5, 6, 9, 8, 10, 15, 12, 16])
(3, 4, 4, [3, 5, 6, 8, 9, 10, 15, 12, 16])
(6, 8, 7, [3, 5, 6, 8, 9, 10, 12, 15, 16])
Final sorted: [3, 5, 6, 8, 9, 10, 12, 15, 16]
PS E:\studies\DAAl\daa lab>
```

Result: Array sorted with quick sort logic.

Q3. Binary Search

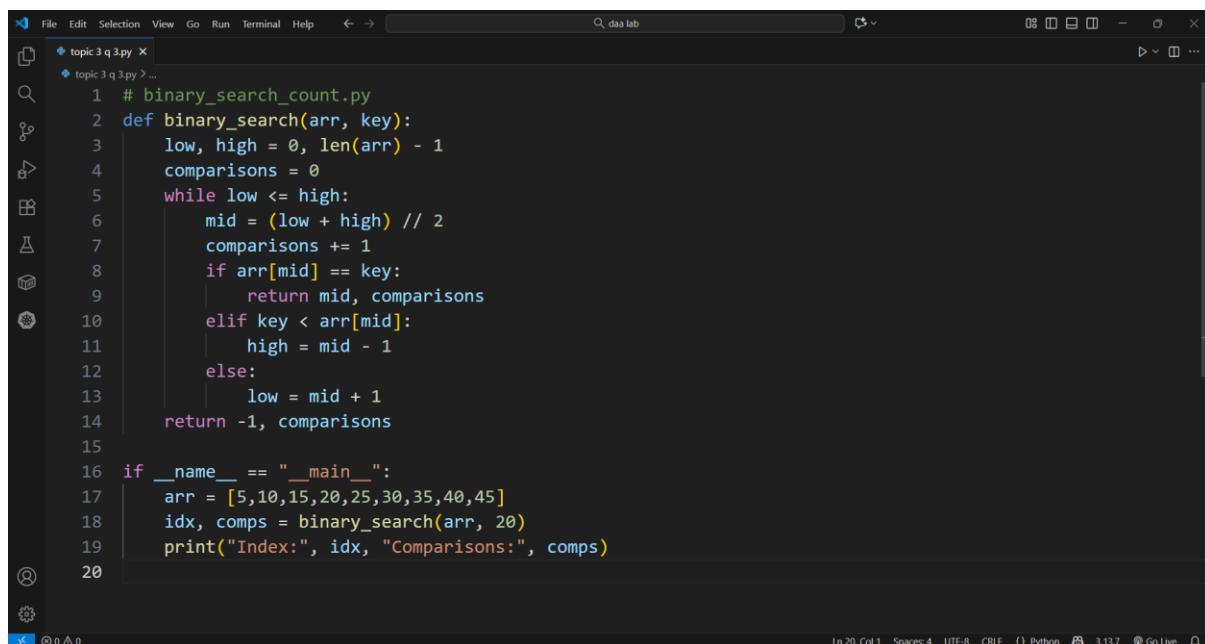
Aim:

To find an element in a sorted array efficiently using binary search.

Algorithm:

1. Set low=0, high=n-1.
2. Find mid index.
3. Compare mid value with key.
4. Adjust range accordingly.
5. Continue until found or range ends.

Code:



```
# binary_search_count.py
def binary_search(arr, key):
    low, high = 0, len(arr) - 1
    comparisons = 0
    while low <= high:
        mid = (low + high) // 2
        comparisons += 1
        if arr[mid] == key:
            return mid, comparisons
        elif key < arr[mid]:
            high = mid - 1
        else:
            low = mid + 1
    return -1, comparisons
if __name__ == "__main__":
    arr = [5,10,15,20,25,30,35,40,45]
    idx, comps = binary_search(arr, 20)
    print("Index:", idx, "Comparisons:", comps)
```

Input: [5,10,15,20,25,30,35,40,45], key=20

Output:

```
PS E:\studies\DAAl\daa lab> & "C:/Program Files/Python313/python.exe" binary_search_count.py
Index: 3 Comparisons: 4
PS E:\studies\DAAl\daa lab>
```

Result: Binary search found the element in O(log n).

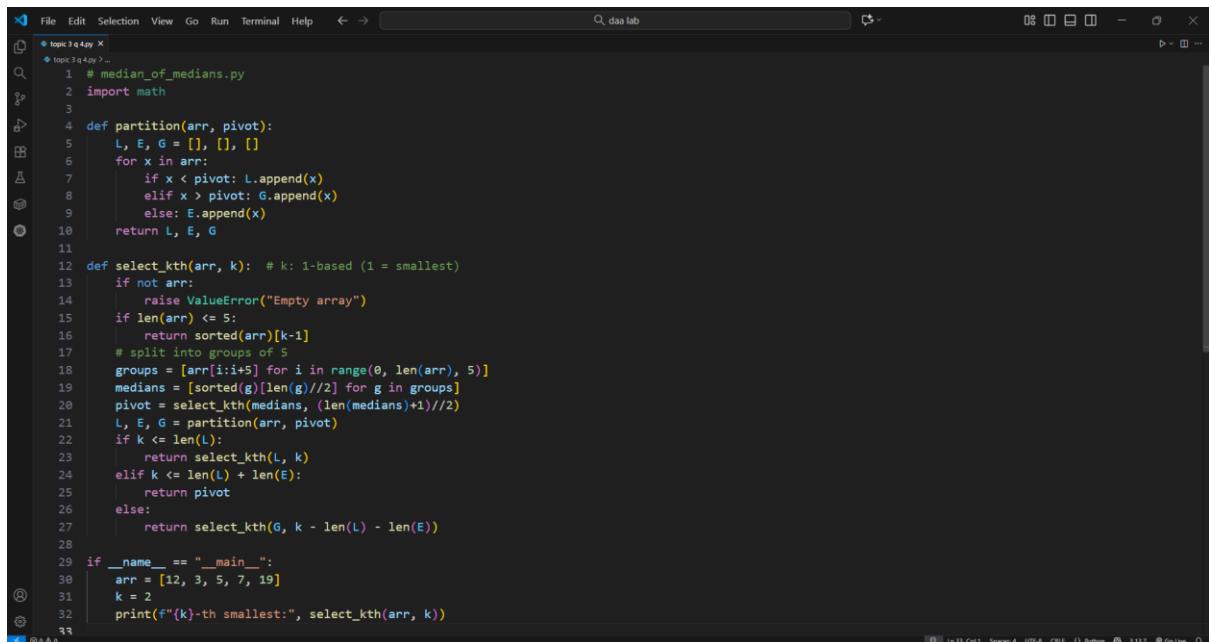
Q4. Median of Medians

Aim:

To find the k-th smallest element in an unsorted array using the median-of-medians method.

Algorithm:

1. Divide array into groups of 5.
2. Sort each group and find median.
3. Find median of medians recursively.
4. Partition array around pivot.
5. Recurse in desired partition.

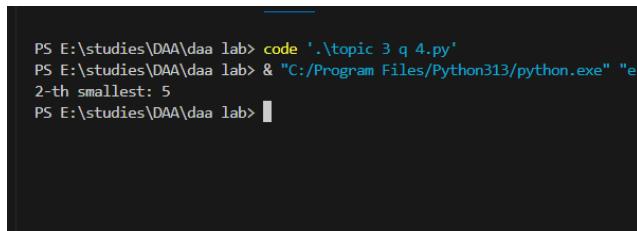


The screenshot shows a code editor window with the following Python code for the Median of Medians algorithm:

```
File Edit Selection View Go Run Terminal Help ← → 
topic 3 q 4.py
topic 3 q 4.py
1 # median_of_medians.py
2 import math
3
4 def partition(arr, pivot):
5     L, E, G = [], [], []
6     for x in arr:
7         if x < pivot: L.append(x)
8         elif x > pivot: G.append(x)
9         else: E.append(x)
10    return L, E, G
11
12 def select_kth(arr, k): # k: 1-based (1 = smallest)
13     if not arr:
14         raise ValueError("Empty array")
15     if len(arr) <= 5:
16         return sorted(arr)[k-1]
17     # split into groups of 5
18     groups = [arr[i:i+5] for i in range(0, len(arr), 5)]
19     medians = [sorted(g)[len(g)//2] for g in groups]
20     pivot = select_kth(medians, (len(medians)+1)//2)
21     L, E, G = partition(arr, pivot)
22     if k <= len(L):
23         return select_kth(L, k)
24     elif k <= len(L) + len(E):
25         return pivot
26     else:
27         return select_kth(G, k - len(L) - len(E))
28
29 if __name__ == "__main__":
30     arr = [12, 3, 5, 7, 19]
31     k = 2
32     print(f"{k}-th smallest:", select_kth(arr, k))
```

Input: [12, 3, 5, 7, 19], k=2

Output:



The screenshot shows a terminal window with the following output:

```
PS E:\studies\DAAlaa lab> code '.\topic 3 q 4.py'
PS E:\studies\DAAlaa lab> & "C:/Program Files/Python313/python.exe" "e:
2-th smallest: 5
PS E:\studies\DAAlaa lab>
```

Result: 2nd smallest element correctly found.

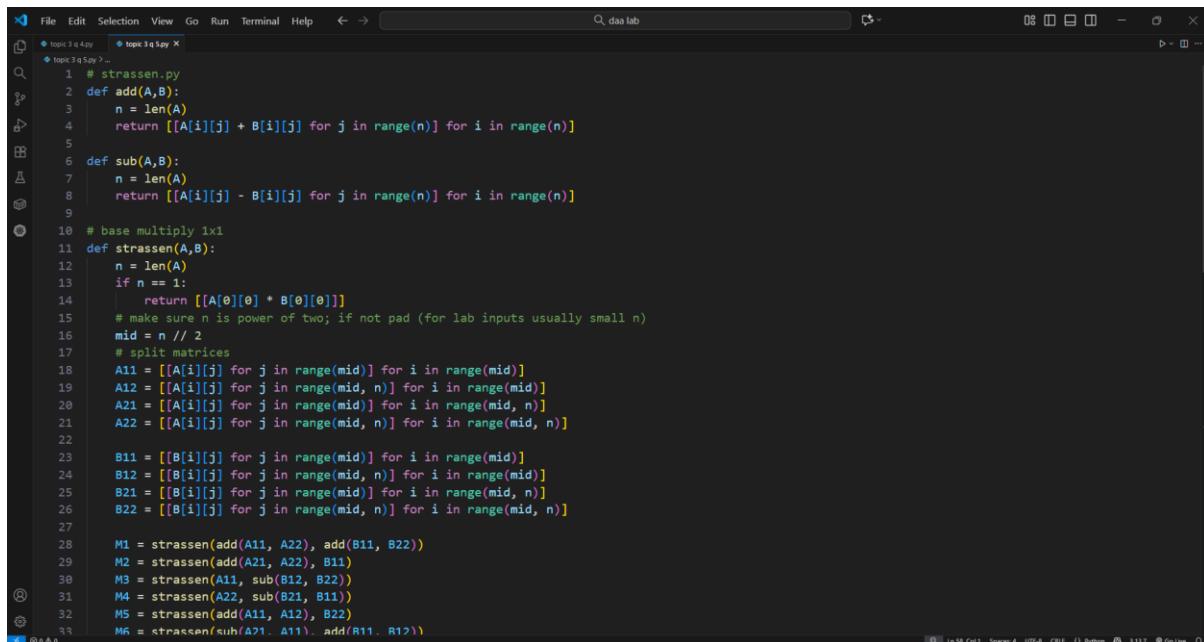
Q5. Strassen's Matrix Multiplication

Aim:

To multiply two matrices efficiently using Strassen's algorithm.

Algorithm:

1. Split matrices into four sub-matrices.
2. Compute 7 products recursively.
3. Combine them using Strassen's formulas.
4. Form resultant matrix.
5. Display output.



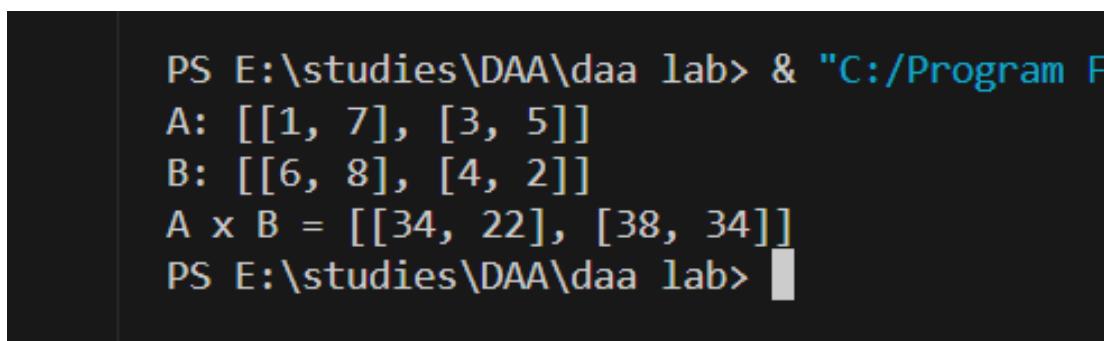
The screenshot shows a code editor window with the file 'strassen.py' open. The code implements Strassen's matrix multiplication algorithm. It includes functions for addition, subtraction, and recursive multiplication of 2x2 matrices. The code is well-structured with comments explaining the logic. The code editor interface includes a toolbar at the top, a search bar, and various status indicators at the bottom.

```
File Edit Selection View Go Run Terminal Help ← → Q das lab
topic 3 > topic 3 > strassen.py
1 # strassen.py
2 def add(A,B):
3     n = len(A)
4     return [[A[i][j] + B[i][j] for j in range(n)] for i in range(n)]
5
6 def sub(A,B):
7     n = len(A)
8     return [[A[i][j] - B[i][j] for j in range(n)] for i in range(n)]
9
10 # base multiply ix1
11 def strassen(A,B):
12     n = len(A)
13     if n == 1:
14         return [[A[0][0] * B[0][0]]]
15     # make sure n is power of two; if not pad (for lab inputs usually small n)
16     mid = n // 2
17     # split matrices
18     A11 = [[A[i][j] for j in range(mid)] for i in range(mid)]
19     A12 = [[A[i][j] for j in range(mid, n)] for i in range(mid)]
20     A21 = [[A[i][j] for j in range(mid)] for i in range(mid, n)]
21     A22 = [[A[i][j] for j in range(mid, n)] for i in range(mid, n)]
22
23     B11 = [[B[i][j] for j in range(mid)] for i in range(mid)]
24     B12 = [[B[i][j] for j in range(mid, n)] for i in range(mid)]
25     B21 = [[B[i][j] for j in range(mid)] for i in range(mid, n)]
26     B22 = [[B[i][j] for j in range(mid, n)] for i in range(mid, n)]
27
28     M1 = strassen(add(A11, A22), add(B11, B22))
29     M2 = strassen(add(A11, A22), B11)
30     M3 = strassen(A11, sub(B12, B22))
31     M4 = strassen(A22, sub(B21, B11))
32     M5 = strassen(add(A11, A12), B22)
33     M6 = strassen(sub(A21, A11), add(B11, B12))
34
35     result = [[0 for j in range(n)] for i in range(n)]
36     result[0][0] = M1
37     result[0][1] = M5
38     result[1][0] = M2
39     result[1][1] = M3
40     result[2][0] = M4
41     result[2][1] = M6
42
43     return result
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
99
```

Input:

$A = [[1, 7], [3, 5]]$, $B = [[6, 8], [4, 2]]$

Output:



The screenshot shows a terminal window running on Windows. The command 'python strassen.py' is entered, followed by the input matrices A and B. The output shows the multiplication results: $A \times B = [[34, 22], [38, 34]]$.

```
PS E:\studies\DAAl\daa_lab> & "C:/Program F
A: [[1, 7], [3, 5]]
B: [[6, 8], [4, 2]]
A x B = [[34, 22], [38, 34]]
PS E:\studies\DAAl\daa_lab>
```

Result: Matrix multiplication done efficiently.