[@KLWKS_BOT] THANOS

Date of the Session:__/__/_____ **Time of the Session:_____to_____**

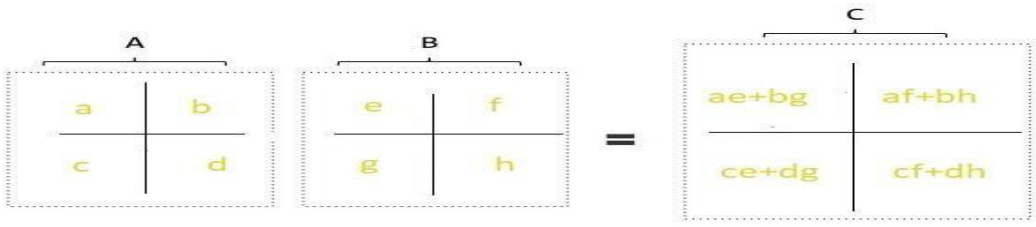
EX – 4 Application of Strassen's Matrix Multiplication and Convex Hull

Prerequisites:

- Basics of Data Structures and C Programming.
- Basic knowledge about Matrix Multiplication.

Pre-Lab:

1) Trace the output of the following matrix multiplication using Strassen's Multiplication Method



A, B and C are the Matrices of Size NxN
a, b, c and d are the sub-Matrices of A of size N/2xN/2
e, f, g and h are the sub-Matrices of B of size N/2xN/2

Step 1: Seven Products

Given the two matrices A and B:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

We calculate the seven products as follows:

1. M_1:

$$M_1 = (a+d)(e+h) = ae + ah + de + dh$$

2. M_2:

$$M_2 = (c+d)e = ce + de$$

3. M_3:

$$M_3 = a(f-h) = af - ah$$

4. M_4:

$$M_4 = d(g-e) = dg - de$$

5. M_5:

$$M_5 = (a+b)h = ah + bh$$

6. M_6:

$$M_6 = (c-a)(e+f) = ce + cf - ae - af$$

7. M_7:

$$M_7 = (b-d)(g+h) = bg + bh - dg - dh$$

Step 2: Calculate the Resulting Matrix C

Matrix C is:

$$C = \begin{bmatrix} p & q \\ r & s \end{bmatrix}$$

Now we use the seven products to compute each element of matrix C:

- p:

$$p = M_1 + M_4 - M_5 + M_7$$

Substituting the expressions for M_1, M_4, M_5, and M_7:

$$p = (ae + ah + de + dh) + (dg - de) - (ah + bh) + (bg + bh)$$

Simplifying:

$$p = ae + ah + de + dh + dg - de - ah - bh + bg + bh$$

After canceling out terms:

$$p = ae + dg + bg$$

- q:

$$q = M_3 + M_5$$

Substituting for M_3 and M_5:

$$q = (af - ah) + (ah + bh) = af + bh$$

- r:

$$r = M_2 + M_4$$

Substituting for M_2 and M_4:

$$r = (ce + de) + (dg - de) = ce + dg$$

- s:

$$s = M_1 - M_2 + M_3 + M_6$$

Substituting for M_1, M_2, M_3, and M_6:

$$s = (ae + ah + de + dh) - (ce + de) + (af - ah) + (ce + cf - ae - af)$$

Simplifying:

$$s = ae + ah + de + dh - ce - de + af - ah + ce + cf - ae - af$$

After canceling out terms:

$$s = dh + cf$$

Final Result

So the resulting matrix C is:

$$C = \begin{bmatrix} p & q \\ r & s \end{bmatrix} = \begin{bmatrix} ae + dg + bg & af + bh \\ ce + dg & dh + cf \end{bmatrix}$$

2) China had recently banned cryptocurrency. Due to this cryptocurrency cost has fallen drastically. Mr. Lee has lost a lot so he could not afford a stock advisor. He came to you (his friend) for help. He has the prices of stock on i'th day. Help him to find the maximum profit he can achieve. You may complete as many transactions as you like.

Input

7

[1,2,3,4,5,6,7]

Output

6

Explanation:

Buy the stock on 1st day at cost 1Rupee and sell the stock on 7th day at cost 7 Rupee and get profit of 6 rupees.

```
#include <stdio.h>

int maxProfit(int prices[], int n) {
   int profit = 0;

   for (int i = 1; i < n; i++) {
      if (prices[i] > prices[i - 1]) {
         profit += prices[i] - prices[i - 1];
      }
   }

   return profit;
}

int main() {
   int prices[] = {1, 2, 3, 4, 5, 6, 7};
   int n = sizeof(prices) / sizeof(prices[0]);

   int profit = maxProfit(prices, n);
   printf("Maximum Profit: %d\n", profit);

   return 0;
}
```

In-Lab:

1) You are given 2 matrices of any size N*N. Write a program to find the product of the two matrixes (use Strassen's Matrix Multiplication).

Source code:

```
#include <stdio.h>
#include <stdlib.h>

void add(int n, int A[n][n], int B[n][n], int result[n][n]) {
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         result[i][j] = A[i][j] + B[i][j];
      }
   }
}

void subtract(int n, int A[n][n], int B[n][n], int result[n][n]) {
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         result[i][j] = A[i][j] - B[i][j];
      }
   }
}

void strassen(int n, int A[n][n], int B[n][n], int result[n][n]) {
   if (n == 1) {
      result[0][0] = A[0][0] * B[0][0];
      return;
   }

   int mid = n / 2;

   int A11[mid][mid], A12[mid][mid], A21[mid][mid], A22[mid][mid];
   int B11[mid][mid], B12[mid][mid], B21[mid][mid], B22[mid][mid];
   int C11[mid][mid], C12[mid][mid], C21[mid][mid], C22[mid][mid];
   int M1[mid][mid], M2[mid][mid], M3[mid][mid], M4[mid][mid], M5[mid][mid],
M6[mid][mid], M7[mid][mid];
   int temp1[mid][mid], temp2[mid][mid];

   for (int i = 0; i < mid; i++) {
      for (int j = 0; j < mid; j++) {
```

```
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + mid];
            A21[i][j] = A[i + mid][j];
            A22[i][j] = A[i + mid][j + mid];

            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + mid];
            B21[i][j] = B[i + mid][j];
            B22[i][j] = B[i + mid][j + mid];
        }
    }

    add(mid, A11, A22, temp1);
    add(mid, B11, B22, temp2);
    strassen(mid, temp1, temp2, M1);

    add(mid, A21, A22, temp1);
    strassen(mid, temp1, B11, M2);

    subtract(mid, B12, B22, temp2);
    strassen(mid, A11, temp2, M3);

    subtract(mid, B21, B11, temp2);
    strassen(mid, A22, temp2, M4);

    add(mid, A11, A12, temp1);
    strassen(mid, temp1, B22, M5);

    subtract(mid, A21, A11, temp1);
    add(mid, B11, B12, temp2);
    strassen(mid, temp1, temp2, M6);

    subtract(mid, A12, A22, temp1);
    add(mid, B21, B22, temp2);
    strassen(mid, temp1, temp2, M7);

    add(mid, M1, M4, temp1);
    subtract(mid, M7, M5, temp2);
    add(mid, temp1, temp2, C11);

    add(mid, M3, M5, C12);

    add(mid, M2, M4, C21);
```

```
    add(mid, M1, M3, temp1);
    subtract(mid, M6, M2, temp2);
    add(mid, temp1, temp2, C22);

    for (int i = 0; i < mid; i++) {
        for (int j = 0; j < mid; j++) {
            result[i][j] = C11[i][j];
            result[i][j + mid] = C12[i][j];
            result[i + mid][j] = C21[i][j];
            result[i + mid][j + mid] = C22[i][j];
        }
    }
}

void printMatrix(int n, int matrix[n][n]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int n;
    printf("Enter the size of the matrix (N x N): ");
    scanf("%d", &n);

    int A[n][n], B[n][n], result[n][n];

    printf("Enter elements of matrix A:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &A[i][j]);
        }
    }

    printf("Enter elements of matrix B:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &B[i][j]);
        }
```

```
        }

        strassen(n, A, B, result);

        printf("Product of matrix A and B is:\n");
        printMatrix(n, result);

        return 0;
}
```

2) Mr. Lee is working in a construction where they had to fencing around trees in a field. The owner has asked a rough estimate to do fencing in that field such all the trees lie inside that region. Consider yourself as a cost estimator who works under Mr. Lee. You are given location of all the trees, and you need to find the points that include this fencing. You need to output the trees that are included in the fencing.

Input:
points = [[1,1], [2,2], [2,0], [2,4], [3,3], [4,2]]
Output:
[[1,1], [2,0], [3,3], [2,4], [4,2]]

Source code:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
   int x;
   int y;
} Point;

int compare(const void *a, const void *b) {
   Point *p1 = (Point *)a;
   Point *p2 = (Point *)b;
   if (p1->x == p2->x) {
      return p1->y - p2->y;
   }
   return p1->x - p2->x;
}

int cross(Point o, Point a, Point b) {
   return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x);
}

void convexHull(Point points[], int n) {
   qsort(points, n, sizeof(Point), compare);

   Point *lower = (Point *)malloc(n * sizeof(Point));
   int lower_size = 0;
   for (int i = 0; i < n; i++) {
      while (lower_size >= 2 && cross(lower[lower_size - 2], lower[lower_size - 1],
points[i]) < 0) {
         lower_size--;
      }
```

```
        lower[lower_size++] = points[i];
    }

    Point *upper = (Point *)malloc(n * sizeof(Point));
    int upper_size = 0;
    for (int i = n - 1; i >= 0; i--) {
        while (upper_size >= 2 && cross(upper[upper_size - 2], upper[upper_size - 1],
points[i]) < 0) {
            upper_size--;
        }
        upper[upper_size++] = points[i];
    }

    int total_size = lower_size + upper_size - 2;
    Point *hull = (Point *)malloc(total_size * sizeof(Point));
    int idx = 0;
    for (int i = 0; i < lower_size - 1; i++) {
        hull[idx++] = lower[i];
    }
    for (int i = 0; i < upper_size - 1; i++) {
        hull[idx++] = upper[i];
    }

    printf("Convex Hull: \n");
    for (int i = 0; i < total_size; i++) {
        printf("[%d, %d]\n", hull[i].x, hull[i].y);
    }

    free(lower);
    free(upper);
    free(hull);
}

int main() {
    Point points[] = {{1, 1}, {2, 2}, {2, 0}, {2, 4}, {3, 3}, {4, 2}};
    int n = sizeof(points) / sizeof(points[0]);

    convexHull(points, n);

    return 0;
}
```

3) In a school there was conducted a contest among two groups. As part of the contest each group must re-arrange the cards that had given to the members in ascending order. Consider yourself as a part of the team and find the best viable way to win that round.

Input

3

3

[[2,5,6], [4,8,7], [9,3,1]]

Output

[1,2,3,4,5,6,7,8,9]

Source code:

```
#include <stdio.h>
#include <stdlib.h>

int compare(const void *a, const void *b) {
   return (*(int *)a - *(int *)b);
}

int main() {
   int arr1[] = {2, 5, 6};
   int arr2[] = {4, 8, 7};
   int arr3[] = {9, 3, 1};

   int len1 = sizeof(arr1) / sizeof(arr1[0]);
   int len2 = sizeof(arr2) / sizeof(arr2[0]);
   int len3 = sizeof(arr3) / sizeof(arr3[0]);

   int total_len = len1 + len2 + len3;

   int merged[total_len];

   int index = 0;
   for (int i = 0; i < len1; i++) merged[index++] = arr1[i];
   for (int i = 0; i < len2; i++) merged[index++] = arr2[i];
   for (int i = 0; i < len3; i++) merged[index++] = arr3[i];

   qsort(merged, total_len, sizeof(int), compare);

   for (int i = 0; i < total_len; i++) {
      printf("%d ", merged[i]);
   }
   printf("\n");

   return 0;
}
```

Post-Lab:

1) Mr. Hari Kumar owns a fruit market. In the market there are many sellers who are selling many kinds of fruits. More than one fruits seller can sell same kind of fruit. Mr. Hari Kumar wants to arrange their information in the sorted order based on their names of the sellers and id of the fruits. You must arrange the same type of fruits in the same order as original order.
0-mangoes 1-apples
[Hint: Use counting sort algorithm]
Input
4
[[0, c], [1, b], [0, a], [1, d]]
Output

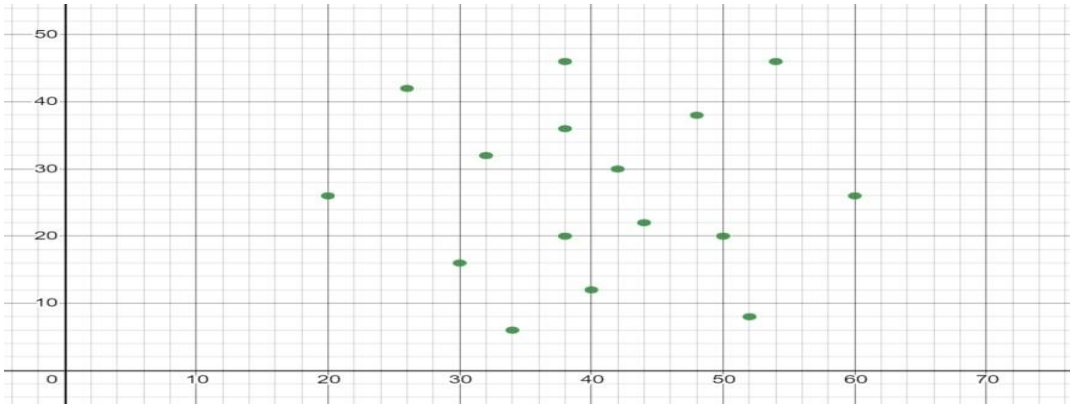
[[0, a], [0, c], [1, b], [1, d]]

Source code:

```
#include <stdio.h>
#include <string.h>

struct FruitSeller {
   int fruit_id;
   char seller_name[50];
};

void counting_sort(struct FruitSeller arr[], int n) {
   struct FruitSeller output[n];
   int count[26] = {0};

   for (int i = 0; i < n; i++) {
      count[arr[i].seller_name[0] - 'a']++;
   }

   for (int i = 1; i < 26; i++) {
      count[i] += count[i - 1];
   }

   for (int i = n - 1; i >= 0; i--) {
      int index = arr[i].seller_name[0] - 'a';
      output[count[index] - 1] = arr[i];
      count[index]--;
   }

   for (int i = 0; i < n; i++) {
      arr[i] = output[i];
```

```
        }
    }

    void print_result(struct FruitSeller arr[], int n) {
        for (int i = 0; i < n; i++) {
            printf("[%d, %s] ", arr[i].fruit_id, arr[i].seller_name);
        }
        printf("\n");
    }

    int main() {
        int n = 4;
        struct FruitSeller sellers[] = {{0, "c"}, {1, "b"}, {0, "a"}, {1, "d"}};

        counting_sort(sellers, n);

        print_result(sellers, n);

        return 0;
}
```

2) Given a set of points in the plane, apply convex hull algorithm to the given points and explain it step by step process.



Source code:

The convex hull algorithm identifies the smallest convex boundary that can enclose a set of points. Let me explain the step-by-step process to apply the convex hull algorithm to the points in the given image.

[@KLWKS_BOT] THANOS

Step-by-Step Process: Convex Hull Algorithm

1. **Identify Extreme Points:**
   - Begin by finding the point with the lowest y-coordinate (if there's a tie, use the x-coordinate). This point serves as the starting reference.
2. **Sort Points by Polar Angle:**
   - Sort all the points relative to the starting point by the angle they make with the x-axis. If two points form the same angle, the closer one is considered first.
3. **Build the Hull Using Orientation Check:**
   - Traverse through the sorted list of points, maintaining a stack to track the vertices of the convex hull.
   - For each point:
     - Check the orientation (left turn, right turn, or collinear) formed by the last two points in the stack and the current point.
     - If it's a right turn, pop the last point from the stack.
     - Push the current point onto the stack.
4. **Close the Loop:**
   - Continue until all points have been processed, and the stack contains the convex hull vertices in order.
5. **Output the Hull:**
   - The stack represents the convex hull in counterclockwise order.

Comments of the Evaluators (if Any)	Evaluator's Observation Marks Secured:_____out of [50]. Signature of the Evaluator Date of Evaluation: