NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
(KARACHI CAMPUS)
FAST School of Computing
FALL 2024

DESIGN AND ANALYSIS OF ALGORITHMS : PROJECT

PROJECT MEMBERS:
22k-4413 Syeda Fakhira Saghir
22k-4448 Aafreen Mughal
22k-4499 Muhammad Raza

REPORT SUBMITTED TO: Sir Abu Zuhran Qaiser

TABLE OF CONTENTS

Question 1: **1**
- A: Part 1) 1
- A: Part 2) 3
- A: Part 3) 3
- B: Part 2) 3
- B: Part 3) 3
- C : 4
- D: 5
- E: 6
- F: 8
- G: 10
- H Exercises (1, 5, 6) 13
- Exercise 1. 13
- Exercise 5. 16
- Exercise 6. 20

Question 2: **23**
- i 23
- Ii 27

Question 3: **31**
- i 31
- ii 34

Question 4: **37**
- i 37
- ii 43

Question 1:

A: part 1)

```
#include <iostream>
using namespace std;
void LargestAndSmallest(int A[], int low, int high, int &largest, int &smallest) {
    if (low == high) {
        largest = A[low];
        smallest = A[low];
        return;
    }
    else if (high == low + 1) {
        if (A[low] > A[high]) {
            largest = A[low];
            smallest = A[high];
        } else {
            largest = A[high];
            smallest = A[low];
        }
        return;
    }
    else {
        int mid = (low + high) / 2;
        int largest1, smallest1, largest2, smallest2;
        LargestAndSmallest(A, low, mid, largest1, smallest1);
        LargestAndSmallest(A, mid + 1, high, largest2, smallest2);
        largest = max(largest1, largest2);
        smallest = min(smallest1, smallest2);
    }
}
int main() {
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    int A[n];
    cout << "Enter the elements of the array:\n";
    for (int i = 0; i < n; i++) {
```

```
    cin >> A[i];
  }
  int largest, smallest;
  LargestAndSmallest(A, 0, n - 1, largest, smallest);
  cout << "Largest element: " << largest << endl;
  cout << "Smallest element: " << smallest << endl;
  return 0;
}
```

Time Complexity 0(N)

A: part 2)

The recurrence relation for the number of comparisons made by the algorithm is:
T(n) = 2T(n/2) + 2
Solution for n = 2^k:
T(n) = O(3n/2 - 2)

A: part 3)

In brute force algorithm, we will iterate through the entire array once to find minimum and iterate once to find maximum. This is done in time complexity O(n).

However, the divide and conquer algorithm also gives time complexity of O(n), which means that the two algorithms are somehow equal in terms of complexity.

The difference between both divide & conquer and brute force is that divide & conquer performs comparisons in a structural manner, while brute force performs comparisons in a single pass.

B: part 1)

```
Algorithm Exponentiation(a, n):
    if n == 0:
    return 1
    else if n is even:
    half = Exponentiation(a, n // 2)
    return half * half
    else:
    return a * Exponentiation(a, n - 1)
```

B:Part 2)

Recurrence Relation for Multiplications:
T(n) = T(n/2) + 1 for even n
The solution to this recurrence is T(n) = O(log n)

B:Part 3)

Comparison with Brute-Force Algorithm:
The brute-force algorithm requires n multiplications, while this divide-and-conquer method performs O(log n) multiplications, making it more efficient.

C :

```
#include <iostream>
using namespace std;
int merge(int arr[], int start, int end, int mid) {
   int i = start;
   int j = mid + 1;
   int k = 0;
   int inversions = 0;
   int temp[end - start + 1];
   while (i <= mid && j <= end) {
      if (arr[i] <= arr[j]) {
         temp[k++] = arr[i++];
      } else {
         temp[k++] = arr[j++];
         inversions += (mid - i + 1);
      }
   }
   while (i <= mid) {
      temp[k++] = arr[i++];
   }
   while (j <= end) {
      temp[k++] = arr[j++];
   }
   for (i = start; i <= end; i++) {
      arr[i] = temp[i - start];
```

```
    }
    return inversions;
}
int mergeSort(int arr[], int start, int end) {
    if (start >= end) {
        return 0;
    }
    int mid = (start + end) / 2;
    int inversions = 0;
    inversions += mergeSort(arr, start, mid);
    inversions += mergeSort(arr, mid + 1, end);
    inversions += merge(arr, start, end, mid);
    return inversions;
}
int main() {
    int arr[] = {8, 4, 2, 1,5,7,8,9,15,4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int inversionCount = mergeSort(arr, 0, n - 1);
    cout << "Sorted Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    cout << "Number of Inversions: " << inversionCount << endl;
    return 0;
}
// Time Complexity:
// Best: O(n log n)
// Avg: O(n log n)
// Worst: O(n log n)
```

D:

Best case:

The best case for quicksort occurs when the pivot element divides the array into two nearly equal halves each time. This results in a balanced division, where each partition step splits the array as evenly as possible. In such a scenario:

1. Each partition splits the array in half
2. The height of the recursion tree therefore is log_2 n.
3. At each level of the tree, the algorithm performs O(n) comparisons.

So, height * comparisons = O(nlog_2n)

<u>Worst case</u>:

The worst case for quicksort occurs when the pivot element is either the smallest or largest element in the array, causing one partition to have n−1 elements and the other partition to be empty (or have just one element) every time. This results in highly unbalanced partitioning. In such a scenario:

1. Each partition only reduces the problem size by one element.
2. The recursion tree has n levels.
3. At each level the algorithm performs O(n) comparisons.

So, height * comparisons = O(n*n) = O(n^2)

<u>E:</u>

1.

<u>Steps to follow:</u>

1. Sort the array
2. Divide the array
3. Base case
4. Recursive closest distance calculation
5. Cross-Pair check (closest pair across the boundary)
6. Check midpoint-crossing pairs
7. Return the final minimum distance

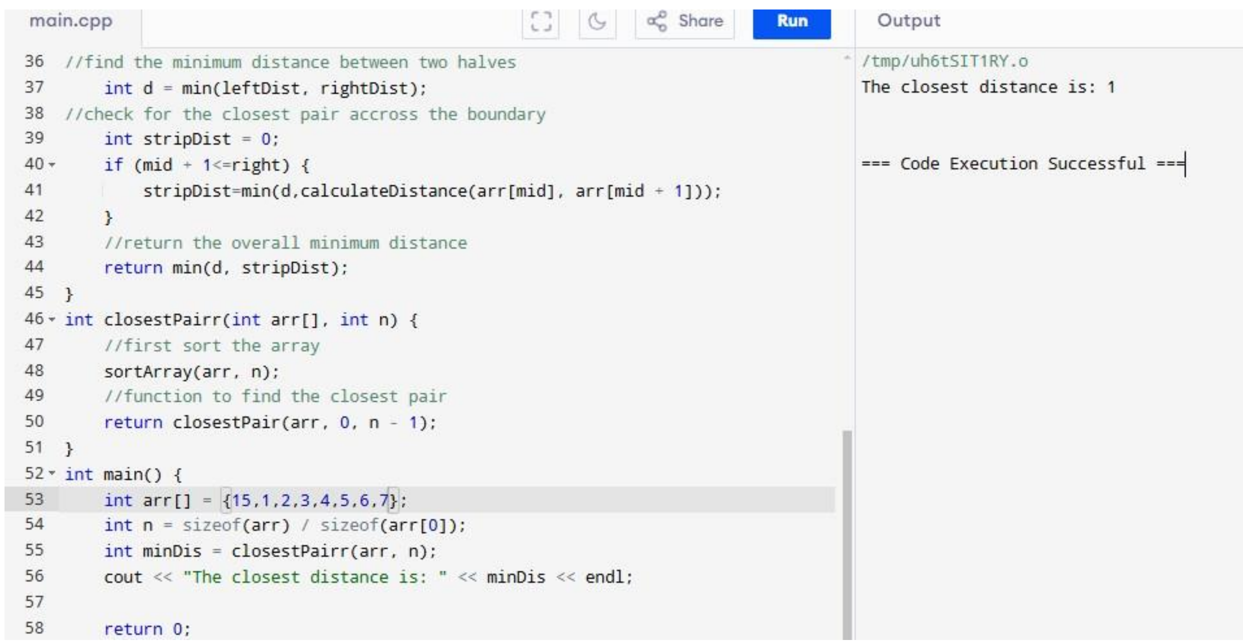
<u>Code:</u>

```
#include <iostream>
#include <math.h>
#include <climits>
using namespace std;
//calculate distance between two points
int calculateDistance(int a, int b) {
```

```
    return abs(a - b);
}
//to sort the points
void sortArray(int arr[], int n){
    for(int i = 0; i < n - 1; i++){
        for(int j = i + 1; j < n; j++){
            if(arr[i] > arr[j]){
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
//to calculate closest pair in the given array
int closestPair(int arr[], int left, int right){
//no need to divide as there are only two elements
    if(right-left==1){
        return calculateDistance(arr[left],arr[right]);
 }
 //if there is only one element in the array
   if (left>=right){
        return -1;
    }
    //using divide and conquer approach
    int mid=left+(right-left)/2;
//find recursively the minimum distance on left and right side
    int leftDist=closestPair(arr,left,mid);
    int rightDist=closestPair(arr,mid + 1,right);
//find the minimum distance between two halves
    int d = min(leftDist, rightDist);
//check for the closest pair across the boundary
    int stripDist = 0;
    if (mid + 1<=right) {
        stripDist=min(d,calculateDistance(arr[mid], arr[mid + 1]));
    }
    //return the overall minimum distance
    return min(d, stripDist);
```

```
}
int closestPairr(int arr[], int n) {
    //first sort the array
    sortArray(arr, n);
    //function to find the closest pair
    return closestPair(arr, 0, n - 1);
}
int main() {
    int arr[] = {7, 1, 9, 3, 14};
    int n = sizeof(arr) / sizeof(arr[0]);
    int minDis = closestPairr(arr, n);
    cout << "The closest distance is: " << minDis << endl;

    return 0;
}
```

<u>Output:</u>



This divide-and-conquer algorithm operates with a time complexity of O(nlogn) due to the sorting step and the recursive splitting. The cross-boundary check is limited to O(n)in each recursion level, making it efficient.

2.

Yes, for one dimensional closest pair this approach takes less time i.e O(nlogn) as compared to brute force which takes O(n^2) times.

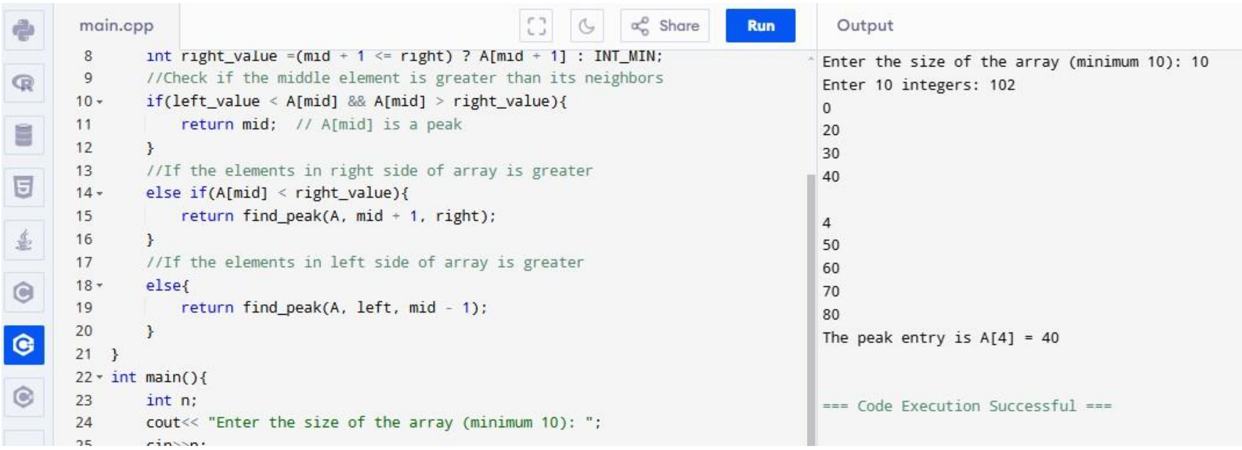
F:

CODE:

```
#include<iostream>
#include<limits.h>
using namespace std;
int find_peak(int A[], int left, int right){
   int mid=(left + right) / 2;
   //Handle boundaries
   int left_value =(mid - 1 >= left) ? A[mid - 1] : INT_MIN;
   int right_value =(mid + 1 <= right) ? A[mid + 1] : INT_MIN;
   //Check if the middle element is greater than its neighbors
   if(left_value < A[mid] && A[mid] > right_value){
      return mid;  // A[mid] is a peak
   }
   //If the elements in right side of array is greater
   else if(A[mid] < right_value){
      return find_peak(A, mid + 1, right);
   }
   //If the elements in left side of array is greater
   else{
      return find_peak(A, left, mid - 1);
   }
}
int main(){
   int n;
   cout<< "Enter the size of the array (minimum 10): ";
   cin>>n;
   if(n < 10){
      cout<< "Size must be at least 10." << endl;
      return 1;
   }
   int A[n];
   cout<< "Enter " << n << " integers: ";
   for(int i = 0; i < n; i++){
      cin>>A[i];
   }
   int peak_index=find_peak(A, 0, n - 1);
```

```
    cout<<"The peak entry is A[" << peak_index << "] = "<<A[peak_index]<<endl;
    return 0;
}
```

<u>OUTPUT:</u>



<u>G:</u>

So we split the array into two parts: search for minimum price on the left and store the index and maximum price on the right and store index.

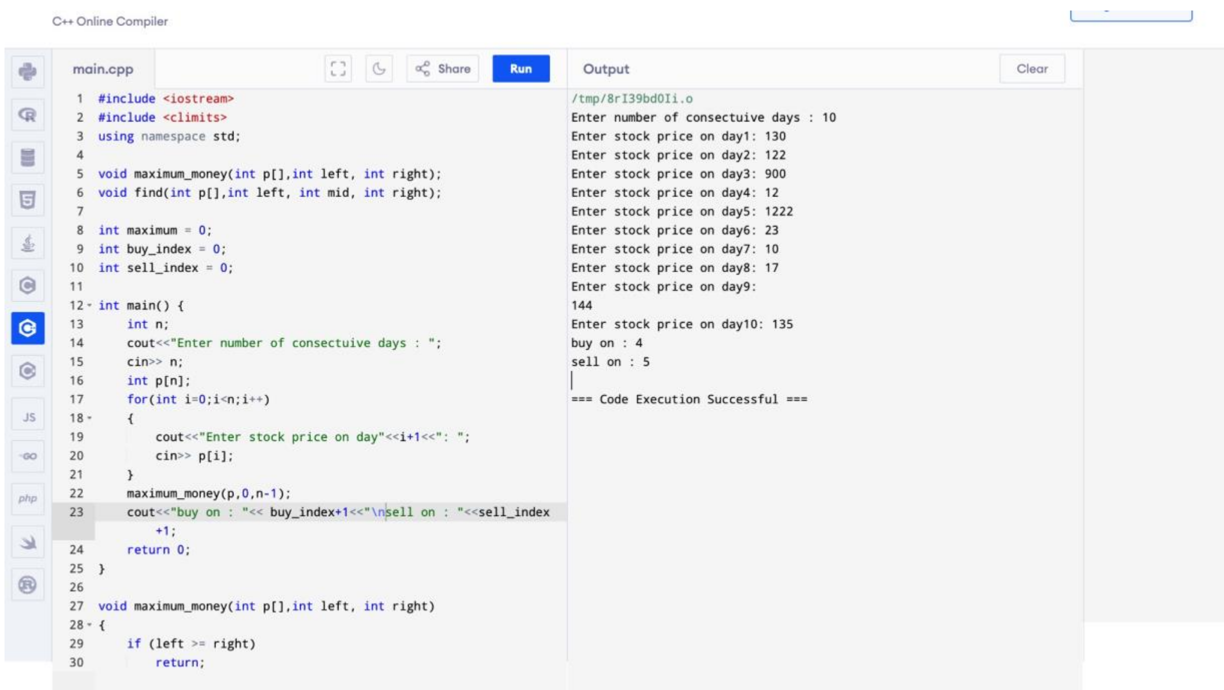
Code according to example in book:

```
#include <iostream>
#include <climits>
using namespace std;

void maximum_money(int p[],int left, int right);
void find(int p[],int left, int mid, int right);

int maximum = 0;
int buy_index = 0;
int sell_index = 0;

int main() {
    int n;
    cout<<"Enter number of consecutive days : ";
    cin>> n;
    int p[n];
```

```
    for(int i=0;i<n;i++)
    {
        cout<<"Enter stock price on day"<<i+1<<": ";
        cin>> p[i];
    }
    maximum_money(p,0,n-1);
    cout<<"buy on : "<< buy_index+1<<"\nsell on : "<<sell_index+1;
    return 0;
}

void maximum_money(int p[],int left, int right)
{
    if (left >= right)
        return;

    int mid = left + (right - left) / 2;
    maximum_money(p, left, mid);
    maximum_money(p, mid + 1, right);
    find(p, left, mid, right);
}

void find(int p[],int left, int mid, int right)
{
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int highL=0,highR=0;
    int lowL=INT_MAX,lowR=INT_MAX;
    int index_HL, index_HR;
    int index_LL, index_LR;
    int L[n1],R[n2];

    for (int i = left; i <= mid; i++)
    {
        if (p[i] < lowL) {
            lowL = p[i];
            index_LL = i;
        }
    }
```

```
for (int j = mid + 1; j <= right; j++)
{
    if (p[j] > highR) {
        highR = p[j];
        index_HR = j;
    }
}

int current_profit = highR - lowL;
if (current_profit > maximum)
{
    maximum = current_profit;
    buy_index = index_LL;
    sell_index = index_HR;
}

}
```

Output screen:



H Exercises (1, 5, 6)

Exercise 1.

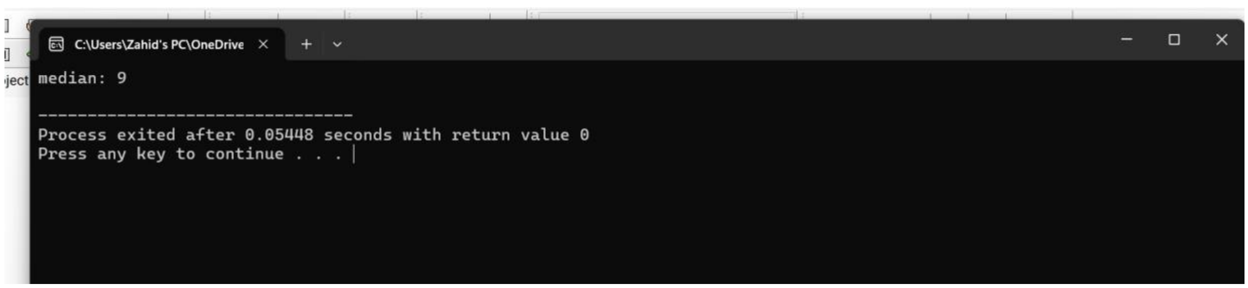
You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values—so there are 2n values total—and you may assume that no two values are the same. You'd like to determine the median of this set of 2n values, which we will define here to be the nth smallest value. However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the kth smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible. Give an algorithm that finds the median value using at most O(log n) queries.

```
//ALGO PROJECT 22K-4413 22K-4448 22K-4499
#include <iostream>
#include <functional>
#include <limits>
using namespace std;

int median(int n, function<int(int)> query_db1, function<int(int)> query_db2) {}

int main() {
   //sorted databases with different data values in each db
   int db1[] = {1, 3, 5, 7, 9, 11, 13, 15,17,19};
   int db2[] = {0, 2, 4, 6, 8, 10, 12, 14,16,18};
   //assuming data bases have same sizes
   int n = sizeof(db1) / sizeof(db1[0]);
   //to access k-th smallest element in each database
   function<int(int)> queryDb1 = [&](int k) -> int { return db1[k]; };
   function<int(int)> queryDb2 = [&](int k) -> int { return db2[k]; };
   cout << "median: " << median(n, queryDb1, queryDb2) << endl;
   return 0;
}

//using binary search
int median(int n, function<int(int)> query_db1, function<int(int)> query_db2)
{
   int l_index = 0, r_index = n;
   //store values of split points
```

```
int db1_leftval, db1_rightval, db2_leftval, db2_rightval;
//find partition between databases
while (l_index < r_index)
{
  // midpoint in first database
  int midpointDb1 = (l_index + r_index) / 2;
  // corresponding index in second database
  int midpointDb2 = n - midpointDb1;
  // largest value in the left of database 1
  if (midpointDb1 > 0)
  {
    db1_leftval = query_db1(midpointDb1 - 1);
  }
  else
  {
    // min value if out of bounds
    db1_leftval = numeric_limits<int>::min();
  }
  // smallest value in the right of database 1
  if (midpointDb1 < n)
  {
    db1_rightval = query_db1(midpointDb1);
  }
  else
  {
    // max value if out of bounds
    db1_rightval = numeric_limits<int>::max();
  }
  // largest value in the left of database 2
  if (midpointDb2 > 0)
  {
    db2_leftval = query_db2(midpointDb2 - 1);
  }
  else
  {
    // min value if out of bounds
    db2_leftval = numeric_limits<int>::min();
  }
```

```
        // minimum value in the right of database 2
        if (midpointDb2 < n)
        {
            db2_rightval = query_db2(midpointDb2);
        }
        else
        {
            // max value if out of bounds
            db2_rightval = numeric_limits<int>::max();
        }

        //if partition valid return max of left values else if db1_leftval > db2_rightval adjust search
range to midpint of database 1 else search right
        if (db1_leftval <= db2_rightval && db2_leftval <= db1_rightval)
        {
            return max(db1_leftval, db2_leftval);
        }
        else if (db1_leftval > db2_rightval)
        {
            r_index = midpointDb1;
        }
        else
        {
            l_index = midpointDb1 + 1;
        }
    }

    // converge binary search to one point
    int finaldb1_leftval, finaldb2_leftval;

    // get left value from database 1
    if (l_index > 0)
    {
        finaldb1_leftval = query_db1(l_index - 1);
    }
    else
    {
        finaldb1_leftval = numeric_limits<int>::min();
```

```
    }

    // get left value from database 2
    if (n - l_index > 0)
    {
        finaldb2_leftval = query_db2(n - l_index - 1);
    }
    else
    {
        finaldb2_leftval = numeric_limits<int>::min();
    }

    // return the larger of the two left values as the median
    return max(finaldb1_leftval, finaldb2_leftval);
}
```



Exercise 5.

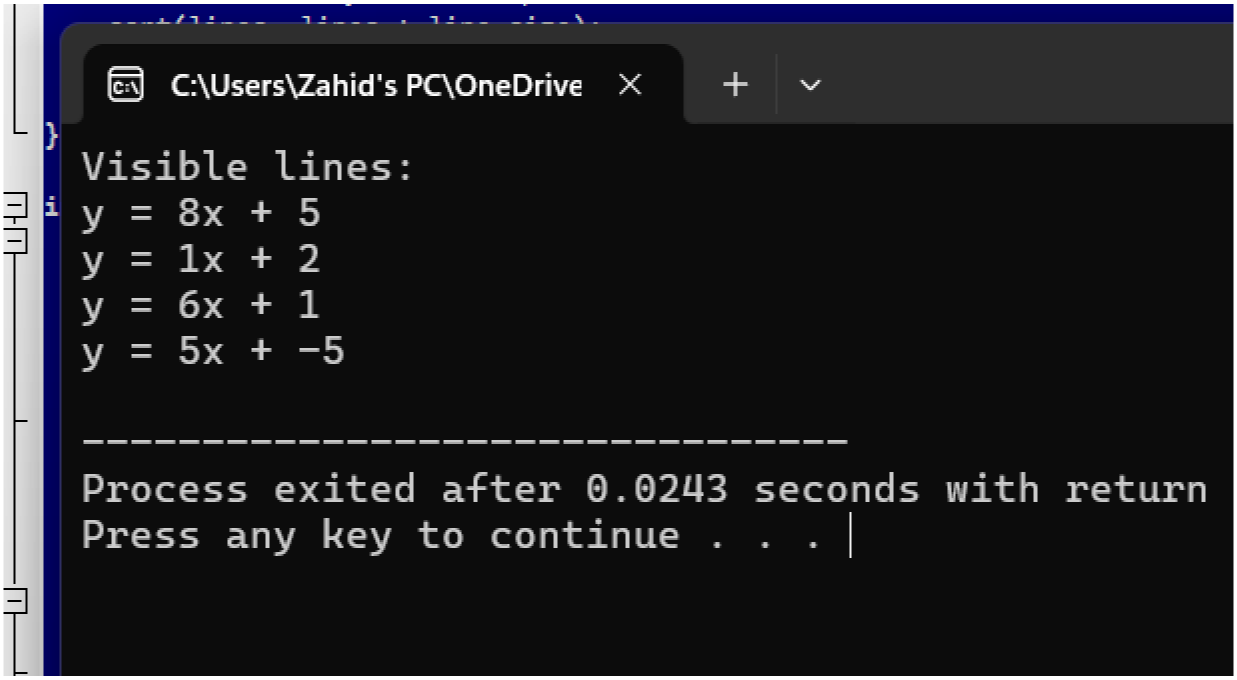
5. Hidden surface removal is a problem in computer graphics that scarcely needs an introduction: when Woody is standing in front of Buzz, you should be able to see Woody but not Buzz; when Buzz is standing in front of Woody,... well, you get the idea. The magic of hidden surface removal is that you can often compute things faster than your intuition suggests. Here's a clean geometric example to illustrate a basic speed-up that can be achieved. You are given n nonvertical lines in the plane, labeled L1,..., Ln, with the i th line specified by the equation y = aix + bi. We will make the assumption that no three of the lines all meet at a single point. We say line Li is uppermost at a given x-coordinate x0 if its y-coordinate at x0 is greater than the y-coordinates of all the other lines at x0: aix0 + bi > ajx0 + bj for all j = i. We say line Li is visible if there is some x-coordinate at which it is uppermost—intuitively, some portion of it can be seen

if you look down from “y = ∞.” Give an algorithm that takes n lines as input and in O(n log n) time returns all of the ones that are visible. Figure 5.10 gives an example.

```
//ALGO PROJECT 22K-4413 22K-4448 22K-4499
#include <iostream>
#include <algorithm>
//Sort lines by slope
//sweep from left to right
//maintain a set of lines that are currently visible at each x-coordinate. As we process each
line:If a new line intersects the current set of visible lines, we check if it is visible at any
x-coordinate. Maintain the lines in a balanced tree structure that allows efficient insertion,
deletion, and querying of the uppermost line.

using namespace std;

class Line {
public:
    double slope, intercept;
    Line() : slope(0), intercept(0) {}

    //constructor with parameters
    Line(double slope, double intercept) : slope(slope), intercept(intercept) {}

    // calculate the y-coordinate of the line at a given x
    double y_at(double x) const {
        return slope * x + intercept;
    }

    // compare lines based on their slope
    bool operator<(const Line& other) const {
        return slope < other.slope;
    }
};

Line* merge_lines(const Line* left, int left_size, const Line* right, int right_size, int& size) {}
Line* find_visible_lines_rec(const Line* lines, int start, int end, int& size) {}
Line* find_visible_lines(Line* lines, int line_size, int& visible_size) {}

int main() {
```

```
    Line lines[] = {
        Line(6, 1), // y = 6x + 1
        Line(8, 5), // y = 8x + 5
        Line(5, -5), // y = 5x - 5
        Line(1, 2) // y = x + 2
    };
    int line_size = sizeof(lines) / sizeof(lines[0]);
    int visible_size = 0;
    Line* visible_lines = find_visible_lines(lines, line_size, visible_size);
    cout << "Visible lines:\n";
    for (int i = 0; i < visible_size; i++) {
        cout << "y = " << visible_lines[i].slope << "x + " << visible_lines[i].intercept << "\n";
    }
    delete[] visible_lines;
    return 0;
}


// merge two sets of visible lines, maintaining only the upper envelope
Line* merge_lines(const Line* left, int left_size, const Line* right, int right_size, int& size) {
    Line* res = new Line[left_size + right_size];
    int i = 0, j = 0, k = 0;
    // compare lines from the left and right halves, keeping the uppermost
    while (i < left_size && j < right_size) {
        if (left[i].y_at(0) > right[j].y_at(0)) {
            res[k++] = left[i++];
        } else {
            res[k++] = right[j++];
        }
    }
    // append any remaining lines from either half
    while (i < left_size) {
        res[k++] = left[i++];
    }
    while (j < right_size) {
        res[k++] = right[j++];
    }
    size = k;
```

```
    return res;
}

// recursive function to find visible lines using divide and conquer
Line* find_visible_lines_rec(const Line* lines, int start, int end, int& size) {
    // base case: if there is one line, it is visible
    if (end - start == 1) {
        size = 1;
        return new Line[1]{lines[start]};
    }
    // split the lines into two halves
    int mid = (start + end) / 2;
    int left_size = 0, right_size = 0;
    Line* left_visible = find_visible_lines_rec(lines, start, mid, left_size);
    Line* right_visible = find_visible_lines_rec(lines, mid, end, right_size);
    // merge the visible lines from the left and right halves
    Line* merged_visible_lines = merge_lines(left_visible, left_size, right_visible, right_size, size);
    delete[] left_visible;
    delete[] right_visible;
    return merged_visible_lines;
}

Line* find_visible_lines(Line* lines, int line_size, int& visible_size) {// find all visible lines in O(n
log n) time
    // sort lines by their slopes
    sort(lines, lines + line_size);
    // use divide and conquer to find visible lines
    return find_visible_lines_rec(lines, 0, line_size, visible_size);
}
```



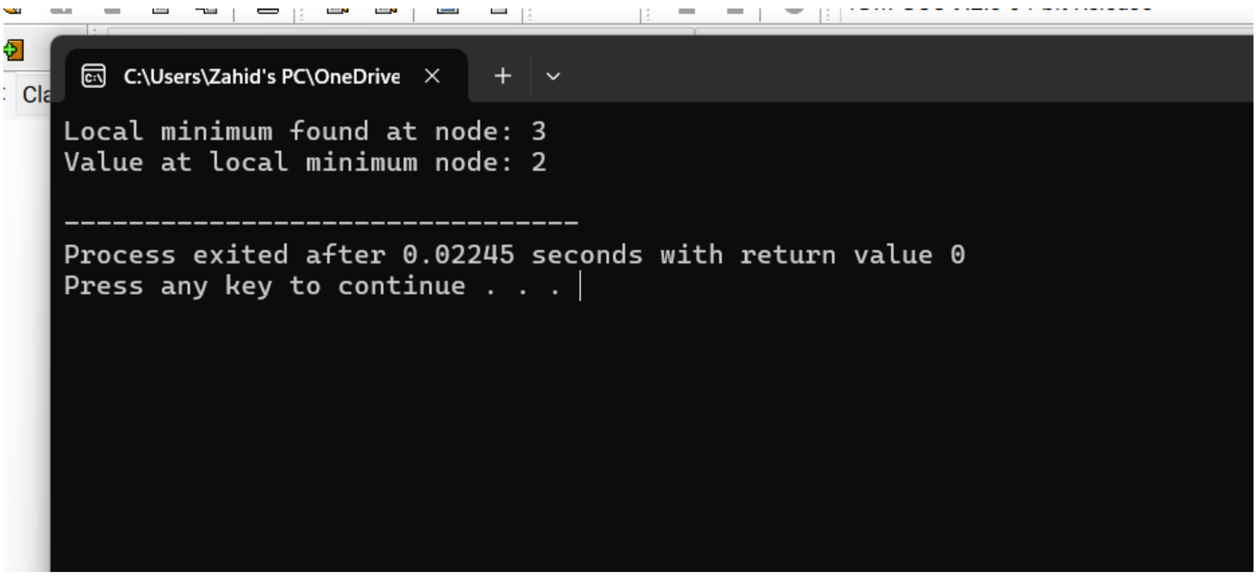
Exercise 6.

6. Consider an n-node complete binary tree T, where n = 2d − 1 for some d. Each node v of T is labeled with a real number xv. You may assume that the real numbers labeling the nodes are all distinct. A node v of T is a local minimum if the label xv is less than the label xw for all nodes w that are joined to v by an edge. You are given such a complete binary tree T, but the labeling is only specified in the following implicit way: for each node v, you can determine the value xv by probing the node v. Show how to find a local minimum of T using only O(log n) probes to the nodes of T.

```
//ALGO PROJECT 22K-4413 22K-4448 22K-4499
#include <iostream>
#include <limits>
using namespace std;

// global array for node values
const int MAX_NODES = 1000; // max nodes
int values[MAX_NODES];
// get value of node v
int probe(int v)
```

```
{
    return values[v];
}

int findlocalMin(int n) {
    // start at root
    int v = 1;
    while (true) {
        // get current node value
        int currentValue = probe(v);
        // track min neighbor
        int minNeighbor = v;
        int minValue = currentValue;
        // check left child
        int left = 2 * v;
        if (left <= n) {
            int leftValue = probe(left);
            if (leftValue < minValue) {
                minNeighbor = left;
                minValue = leftValue;
            }
        }
        // check right child
        int right = 2 * v + 1;
        if (right <= n) {
            int rightValue = probe(right);
            if (rightValue < minValue) {
                minNeighbor = right;
                minValue = rightValue;
            }
        }
        // check parent
        int parent = v / 2;
        if (parent >= 1) {
            int parentValue = probe(parent);
            if (parentValue < minValue) {
                minNeighbor = parent;
                minValue = parentValue;
```

```
            }
        }
        // if current node is local min, return it
        if (minNeighbor == v) {
            return v;
        }
        // move to min neighbor
        v = minNeighbor;
    }
}

int main() {
    // example tree in binary format
    values[1] = 20;
            values[2] = 11;
            values[3] = 2;
            values[4] = 8;
    values[5] = 13;
            values[6] = 18;
            values[7] = 40;
    int n = 7;  // number of nodes
    cout << "Local minimum found at node: " << findlocalMin(n) << "\nValue at local minimum
node: " << probe(findlocalMin(n)) << endl;
    return 0;
}
```



Question 2:

i :

```
#include<iostream>
#include<fstream>
#include<sstream>
#include<ctime>
#include<cmath>
#include<cstdlib>
#include<algorithm>
#include<string>
using namespace std;
class Point{
public:
   int x, y;
};
Point closestPair[2];//for storing closest pair
//sort X coordinate
bool sortX(const Point& p1, const Point& p2){
   return p1.x < p2.x;
}
//sort y coordinate
```

```
bool sortY(const Point& p1, const Point& p2){
    return p1.y < p2.y;
}
//calculate the distance between two points
float distancee(Point p1, Point p2) {
    return sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));
}

//brute force to return the smallest distance between two points
float bruteForce(Point P[], int n){
    float minD = INT_MAX;
    for(int i = 0; i < n; ++i){
        for(int j = i + 1; j < n; ++j){
            if (distancee(P[i], P[j]) < minD){
                minD = distancee(P[i], P[j]);
                closestPair[0]=P[i];
                closestPair[1]=P[j];
            }
        }
    }
    return minD;
}
// A utility function to find the minimum of two float values
float min(float x, float y){
    return (x < y) ? x : y;
}
//find the distance between the closest points in a strip
float stripClosest(Point strip[], int size, float d){
    float minD=d; //initialize the minimum distance as d
    sort(strip, strip + size, sortY); //sort by Y-coordinate

    for(int i = 0; i < size; ++i){
        for(int j = i + 1; j < size && (strip[j].y - strip[i].y) < minD; ++j){
            if(distancee(strip[i], strip[j]) < minD){
                minD = distancee(strip[i], strip[j]);
                closestPair[0]=strip[i];
                closestPair[1]=strip[j];
            }
```

```
        }
    }
    return minD;
}
//recursive function to find the smallest distance
float closestDtil(Point P[], int n) {
    if (n <= 3) {//BASE CASE
        return bruteForce(P, n);
    }
    int mid = n / 2;
    Point midPoint=P[mid];
//LEFT DISTANCE CALCULATE
    float dl=closestDtil(P, mid);
    float dr=closestDtil(P + mid, n - mid);//RIGHT DISTANCE CALCULATE
    float d=min(dl, dr);
    Point strip[n];
    int j = 0;
    //strip point distance calculate
    for(int i = 0; i < n; i++){
        if(abs(P[i].x - midPoint.x) < d){
            strip[j] = P[i];
            j++;
        }
    }
    return min(d, stripClosest(strip, j, d));
}

//closest point find
float closest(Point P[], int n){
    sort(P, P + n, sortX);
    return closestDtil(P, n);
}
// function to generate filenames
string filename(int k){
    stringstream fileName;
    fileName<<"file"<<k<<".txt";
    return fileName.str();
}
```

```
int main(){
    ofstream fout;
    srand(time(0));
    double x,y;
    for(int k=0;k<10;k++){
    string file =filename(k);
    fout.open(file.c_str());
    if(!fout){
        cout<<"Failed to open file.\n";
        return 1;
    }

    int n=rand()%150+101;
   for(int i=0;i<n;i++){
       float x = rand() % 1000 + 0.0;
      float y = rand() % 1000 + 0.0;
      fout << x << " " << y << endl;
    }
    fout.close();}
    //DISPLAY FILES NAMES
    string fileName;
    int count=0;
    while(count!=10){
    cout<<"Enter file name from below listed Files.\n";
    for(int i=0;i<10;i++){
     cout<<i+1<<". file"<<i<<".txt"<<endl;
    }
    while(true){
        cin>>fileName;
        ifstream file(fileName.c_str());
        if(file){
            file.close();
            break;
        } else {
            cout << "File \"" << fileName << "\" does not exist. Please try again: ";
        }
    }
```

```
    ifstream file(fileName.c_str());
    int i=0;
     Point P[1000];
    while (file >> P[i].x >> P[i].y) {
        i++;
    }
    file.close();

    cout << "The smallest distance is: " << closest(P, i) << endl;
    cout << "The closest pair of points is: (" << closestPair[0].x << ", " << closestPair[0].y << ") and
("
         << closestPair[1].x << ", " << closestPair[1].y << ")" << endl;
                        count++;}

    return 0;
}
```

Ii:

```
//ALGO PROJECT 22K-4413 22K-4448 22K-4499
#include <iostream>
#include <fstream>
#include <ctime>
#include <cstdlib>
#include <sstream>
#include<math.h>
using namespace std;

// function to generate filenames
string filename(int i) {
    stringstream fileName;
    fileName << "iifile" << i + 1 << ".txt";
    return fileName.str();
}

// function to create 20 files with random numbers
int createfiles() {
```

```
    srand(time(0));
    int n = rand() % 100 + 50; // number of random integers between 50 and 150
    for (int i = 0; i < 20; i++) {
        string file = filename(i);
        ofstream out(file, ios::out);
        if (!out) {
            cout << "error opening file " << i + 1 << " for writing." << endl;
            return 0; // return 0 if file can't be created
        }
        // write n random integers to the file
        for (int j = 0; j < n; j++) {
            out << rand() % 200 + 10 << endl;
        }
        out.close();
    }
    return n; // return the number of integers generated in each file
}

// function to implement karatsuba's algorithm
long long karatsuba(long long x, long long y) {
    if (x < 10 || y < 10) { // base case: single-digit multiplication
        return x * y;
    }

    // calculate the size of the numbers
    int n = max(to_string(x).length(), to_string(y).length());
    int m = n / 2;

    // split the numbers into two halves
    long long high_x = x / (long long)pow(10, m);
    long long low_x = x % (long long)pow(10, m);
    long long high_y = y / (long long)pow(10, m);
    long long low_y = y % (long long)pow(10, m);

    // recursively compute the three products
    long long p1 = karatsuba(high_x, high_y);
    long long p2 = karatsuba(low_x, low_y);
    long long p3 = karatsuba(high_x + low_x, high_y + low_y);
```

```
    // combine the results using karatsuba's formula
    return p1 * pow(10, 2 * m) + (p3 - p1 - p2) * pow(10, m) + p2;
}

int main() {
    string file1, file2;
    int n = createfiles(); // create files and get the number of elements

    if (n == 0) {
        cout << "error: files not created properly." << endl;
        return 1;
    }

    int x[n] = {0}, y[n] = {0};
    int count1 = 0, count2 = 0;

    // display file names
    cout << "displaying file names: \n";
    for (int i = 0; i < 10; i++) {
        cout << filename(i) << " ";
    }

    // user input for file names
    cout << "\nenter file 1 to choose inputs for x: ";
    cin >> file1;
    cout << "\nenter file 2 to choose inputs for y: ";
    cin >> file2;

    // open the files for reading
    ifstream f1(file1);
    ifstream f2(file2);

    if (!f1) {
        cout << "error opening file 1!" << endl;
        return 1;
    }
    if (!f2) {
```

```
    cout << "error opening file 2!" << endl;
    return 1;
}

string line1, line2;

// read lines from both files and convert them to integers
while (getline(f1, line1) && getline(f2, line2)) {
    stringstream ss1(line1), ss2(line2);
    int num1, num2;

    // convert line 1 to an integer and store it in x
    if (ss1 >> num1) {
        x[count1++] = num1;
    } else {
        cout << "can't convert line to integer: " << line1 << endl;
    }

    // convert line 2 to an integer and store it in y
    if (ss2 >> num2) {
        y[count2++] = num2;
    } else {
        cout << "can't convert line to integer: " << line2 << endl;
    }

    // print the product of the values from both arrays using karatsuba's algorithm
    if (count1 > 0 && count2 > 0) {
        long long result = karatsuba(x[count1 - 1], y[count2 - 1]);
        cout << result << " ";
        ofstream out("OUTPUTQ2(ii)", ios::out);
                  if (!out) {
                      cout << "error opening file for writing." << endl;
                      return 0; // return 0 if file can't be created
                  }
                  out << result << " ";
                  out.close();
    }
}
```

```
    // close the files after reading
    f1.close();
    f2.close();

    return 0;
}
```

Question 3:

i

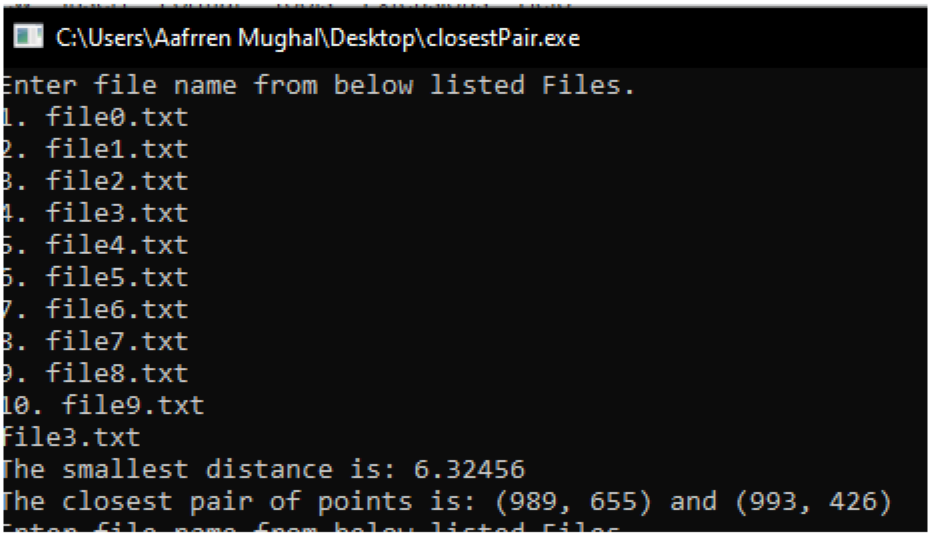
```
C:\Users\Aafrren Mughal\Desktop\closestPair.exe
Enter file name from below listed Files.
1. file0.txt
2. file1.txt
3. file2.txt
4. file3.txt
5. file4.txt
6. file5.txt
7. file6.txt
8. file7.txt
9. file8.txt
10. file9.txt
file8.txt
The smallest distance is: 5.38516
The closest pair of points is: (992, 584) and (970, 591)
```

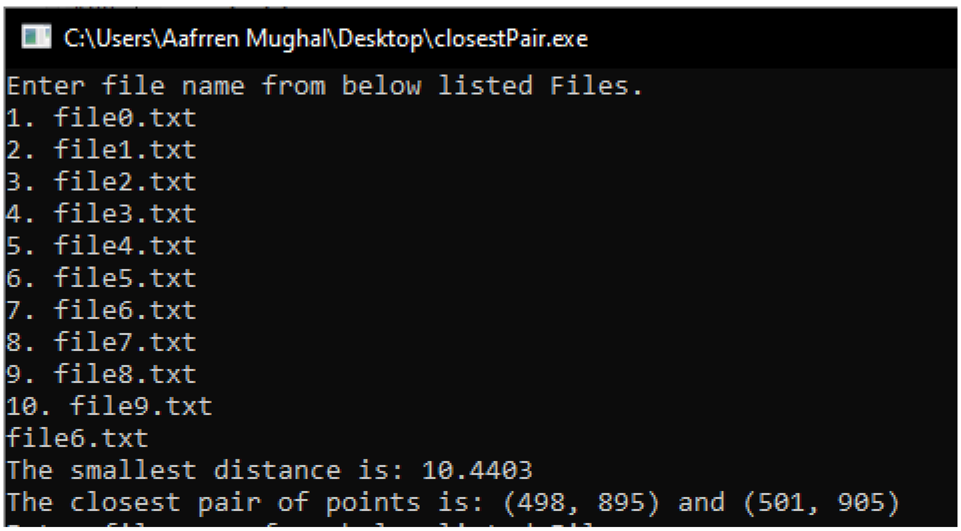
```
C:\Users\Aafrren Mughal\Desktop\closestPair.exe
Enter file name from below listed Files.
1. file0.txt
2. file1.txt
3. file2.txt
4. file3.txt
5. file4.txt
6. file5.txt
7. file6.txt
8. file7.txt
9. file8.txt
10. file9.txt
file9.txt
The smallest distance is: 4
The closest pair of points is: (995, 459) and (997, 480)
```

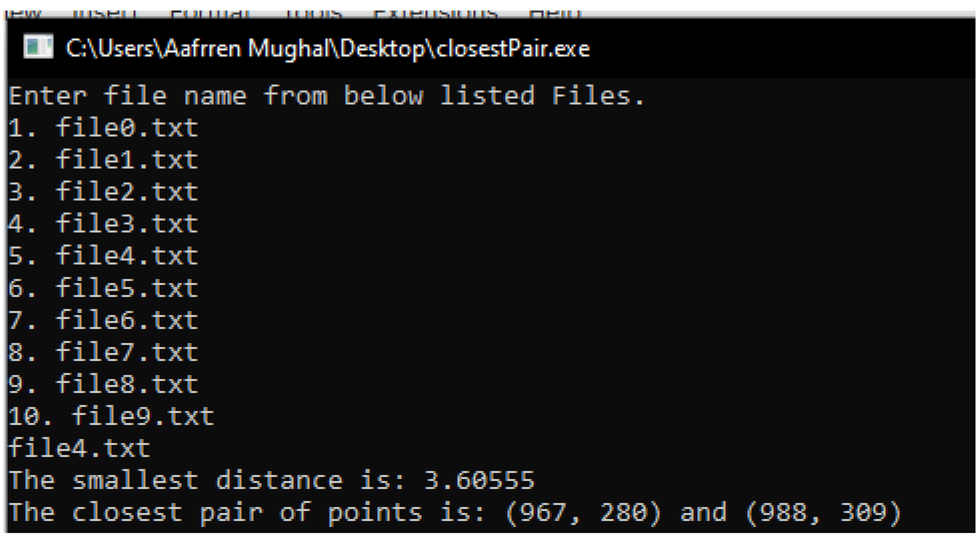
```
C:\Users\Aafrren Mughal\Desktop\closestPair.exe
Enter file name from below listed Files.
1. file0.txt
2. file1.txt
3. file2.txt
4. file3.txt
5. file4.txt
6. file5.txt
7. file6.txt
8. file7.txt
9. file8.txt
10. file9.txt
file7.txt
The smallest distance is: 1
The closest pair of points is: (992, 489) and (998, 665)
```

```
C:\Users\Aafrren Mughal\Desktop\closestPair.exe
Enter file name from below listed Files.
1. file0.txt
2. file1.txt
3. file2.txt
4. file3.txt
5. file4.txt
6. file5.txt
7. file6.txt
8. file7.txt
9. file8.txt
10. file9.txt
file2.txt
The smallest distance is: 10
The closest pair of points is: (974, 300) and (988, 314)
```

```
Enter file name from below listed Files.
1. file0.txt
2. file1.txt
3. file2.txt
4. file3.txt
5. file4.txt
6. file5.txt
7. file6.txt
8. file7.txt
9. file8.txt
10. file9.txt
file0.txt
The smallest distance is: 4.12311
The closest pair of points is: (990, 642) and (998, 656)
```


C:\Users\Aafrren Mughal\Desktop\closestPair.exe
Enter file name from below listed Files.
1. file0.txt
2. file1.txt
3. file2.txt
4. file3.txt
5. file4.txt
6. file5.txt
7. file6.txt
8. file7.txt
9. file8.txt
10. file9.txt
file3.txt
The smallest distance is: 6.32456
The closest pair of points is: (989, 655) and (993, 426)

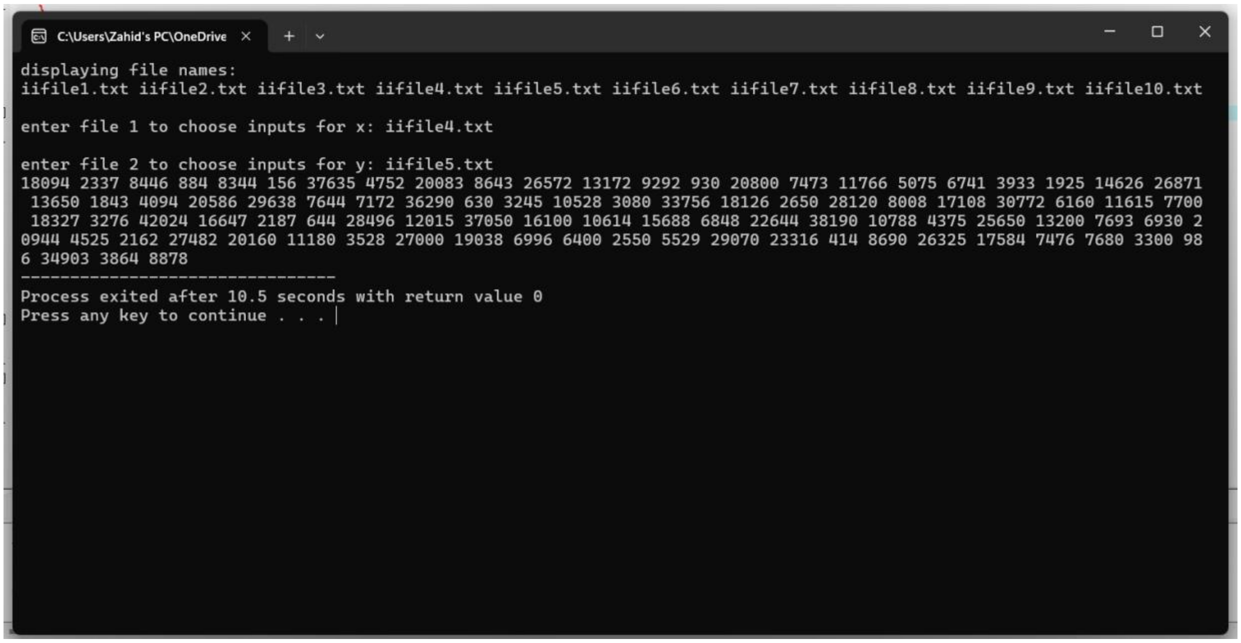

C:\Users\Aafrren Mughal\Desktop\closestPair.exe
Enter file name from below listed Files.
1. file0.txt
2. file1.txt
3. file2.txt
4. file3.txt
5. file4.txt
6. file5.txt
7. file6.txt
8. file7.txt
9. file8.txt
10. file9.txt
file6.txt
The smallest distance is: 10.4403
The closest pair of points is: (498, 895) and (501, 905)


C:\Users\Aafrren Mughal\Desktop\closestPair.exe
Enter file name from below listed Files.
1. file0.txt
2. file1.txt
3. file2.txt
4. file3.txt
5. file4.txt
6. file5.txt
7. file6.txt
8. file7.txt
9. file8.txt
10. file9.txt
file4.txt
The smallest distance is: 3.60555
The closest pair of points is: (967, 280) and (988, 309)

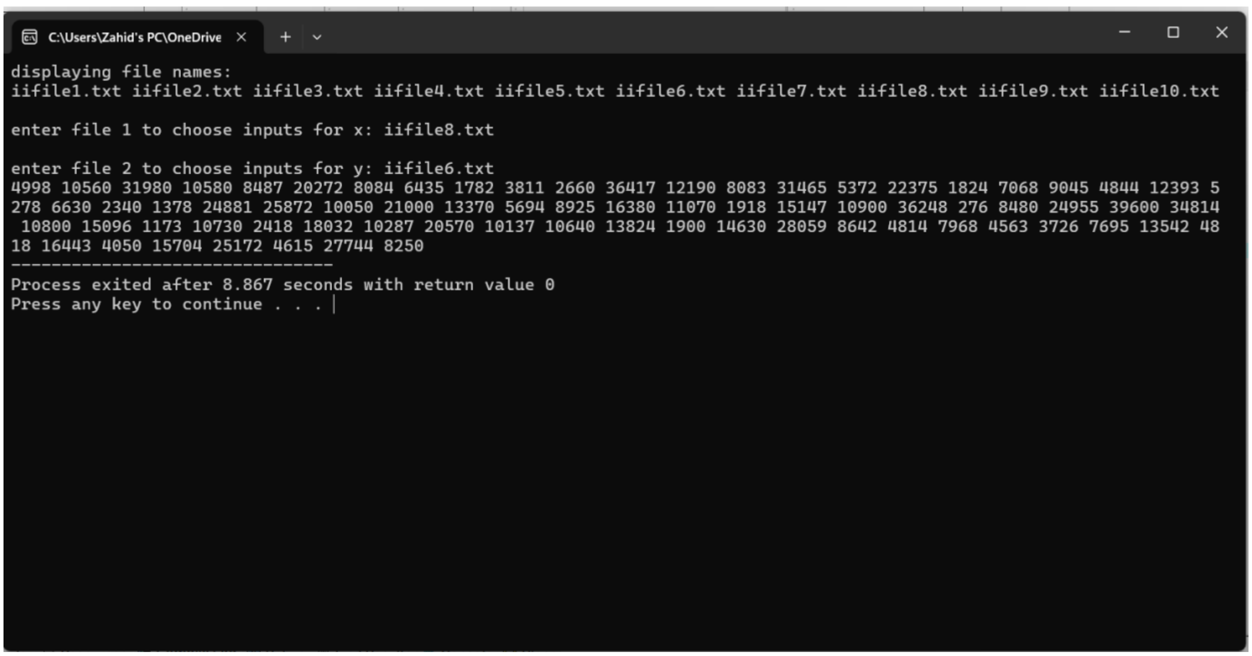
```
C:\Users\Aafrren Mughal\Desktop\closestPair.exe
Enter file name from below listed Files.
1. file0.txt
2. file1.txt
3. file2.txt
4. file3.txt
5. file4.txt
6. file5.txt
7. file6.txt
8. file7.txt
9. file8.txt
10. file9.txt
file1.txt
The smallest distance is: 2
The closest pair of points is: (982, 264) and (994, 284)
```

ii

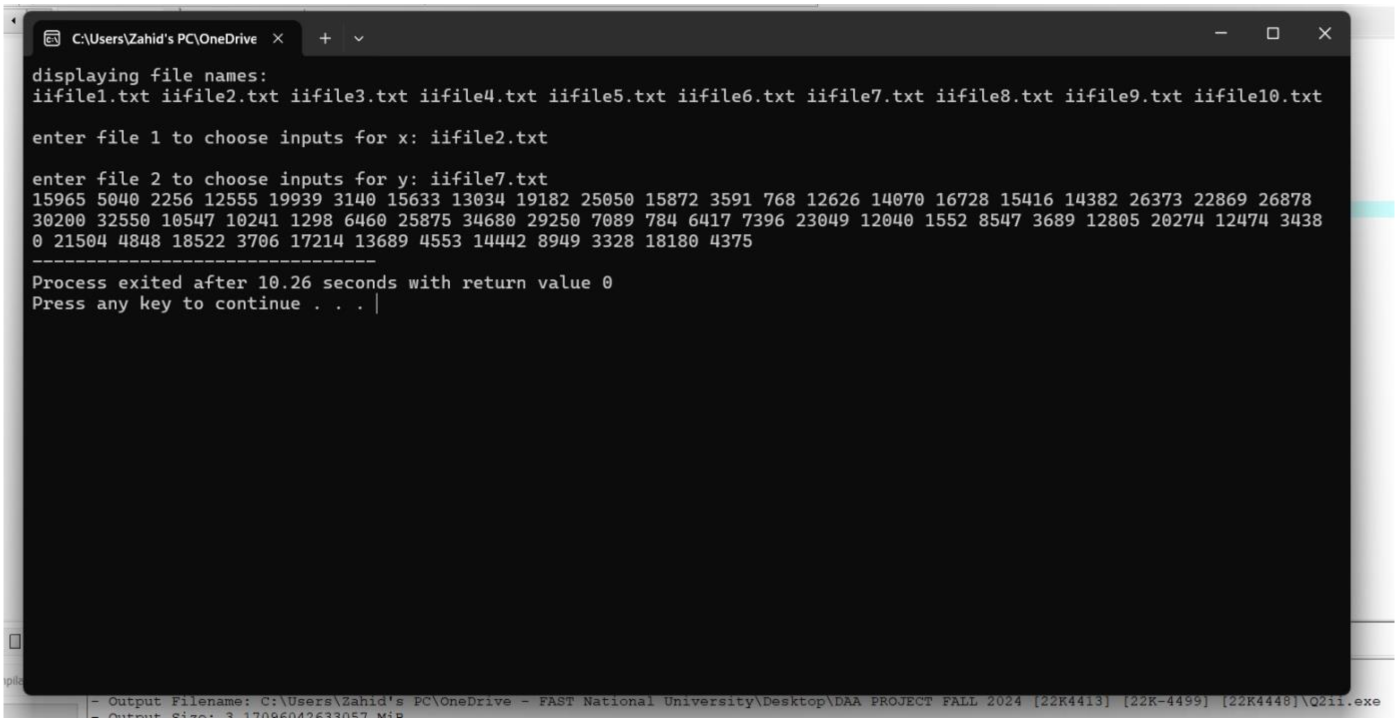
```
C:\Users\Zahid's PC\OneDrive
displaying file names:
iifile1.txt iifile2.txt iifile3.txt iifile4.txt iifile5.txt iifile6.txt iifile7.txt iifile8.txt iifile9.txt iifile10.txt

enter file 1 to choose inputs for x: iifile1.txt

enter file 2 to choose inputs for y: iifile10.txt
528 5952 7636 29400 1395 12993 15504 6150 13286 30394 1625 12709 38745 11557 14700 17952 13875 14784 1176 24596 5312 187
20 14400 22089 1207 18584 3781 2655 2624 3439 6802 24776 7134 10250 16065 7475 3509 10260 16000 11529 14382 16954 9009 5
148 6751 31897 13104 3834 4752 15087 9072 14544 17374 15957 1488 3564 6919 20406 1804 2660 14703 25760 18120 12765 32164
 32032 8556 496 24130 28518 26986 684 3423 7705 5282 5177 7540 7869 3564 22192 16215 9514 5508 6789 25883 16700 4992 171
5 17018 1575
--------------------------------
Process exited after 12.7 seconds with return value 0
Press any key to continue . . .
```


C:\Users\Zahid's PC\OneDrive
displaying file names:
iifile1.txt iifile2.txt iifile3.txt iifile4.txt iifile5.txt iifile6.txt iifile7.txt iifile8.txt iifile9.txt iifile10.txt
enter file 1 to choose inputs for x: iifile4.txt
enter file 2 to choose inputs for y: iifile5.txt
18094 2337 8446 884 8344 156 37635 4752 20083 8643 26572 13172 9292 930 20800 7473 11766 5075 6741 3933 1925 14626 26871
13650 1843 4094 20586 29638 7644 7172 36290 630 3245 10528 3080 33756 18126 2650 28120 8008 17108 30772 6160 11615 7700
18327 3276 42024 16647 2187 644 28496 12015 37050 16100 10614 15688 6848 22644 38190 10788 4375 25650 13200 7693 6930 2
0944 4525 2162 27482 20160 11180 3528 27000 19038 6996 6400 2550 5529 29070 23316 414 8690 26325 17584 7476 7680 3300 98
6 34903 3864 8878

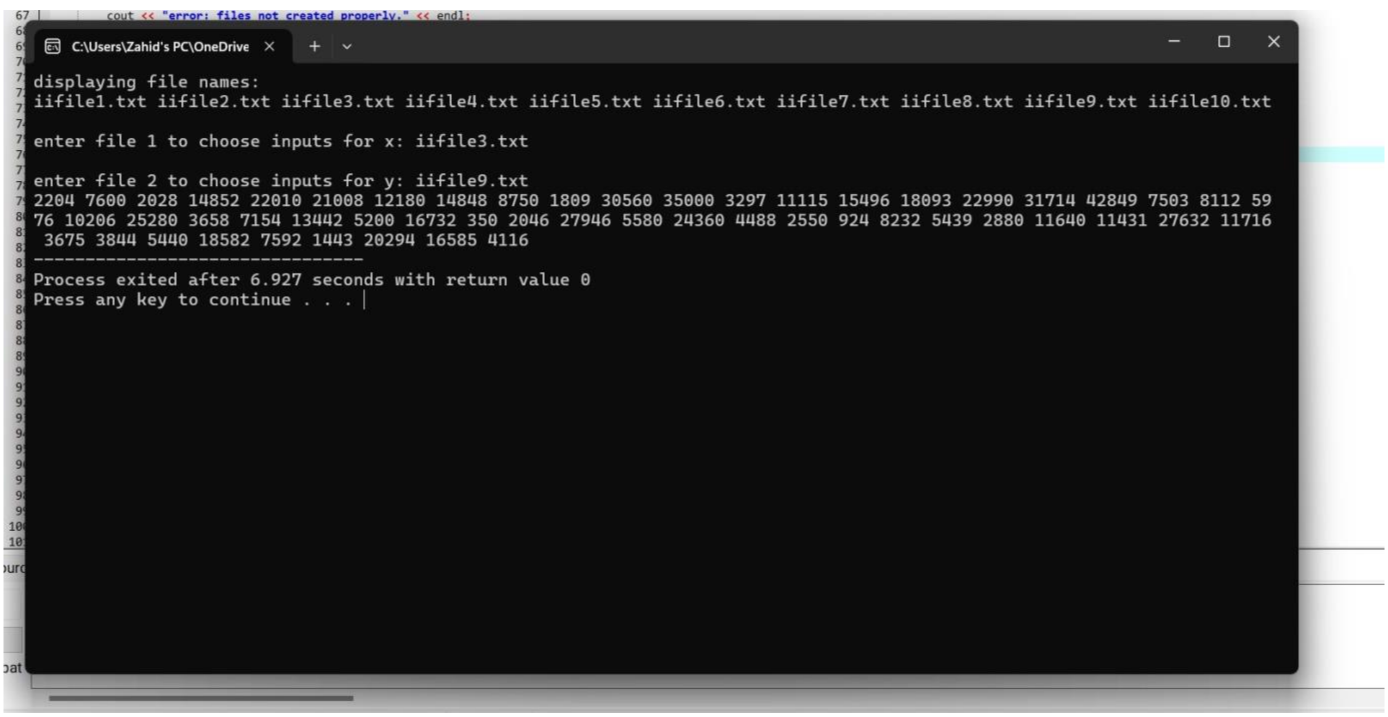
Process exited after 10.5 seconds with return value 0
Press any key to continue . . .


C:\Users\Zahid's PC\OneDrive
displaying file names:
iifile1.txt iifile2.txt iifile3.txt iifile4.txt iifile5.txt iifile6.txt iifile7.txt iifile8.txt iifile9.txt iifile10.txt
enter file 1 to choose inputs for x: iifile8.txt
enter file 2 to choose inputs for y: iifile6.txt
4998 10560 31980 10580 8487 20272 8084 6435 1782 3811 2660 36417 12190 8083 31465 5372 22375 1824 7068 9045 4844 12393 5
278 6630 2340 1378 24881 25872 10050 21000 13370 5694 8925 16380 11070 1918 15147 10900 36248 276 8480 24955 39600 34814
10800 15096 1173 10730 2418 18032 10287 20570 10137 10640 13824 1900 14630 28059 8642 4814 7968 4563 3726 7695 13542 48
18 16443 4050 15704 25172 4615 27744 8250

Process exited after 8.867 seconds with return value 0
Press any key to continue . . .


C:\Users\Zahid's PC\OneDrive
displaying file names:
iifile1.txt iifile2.txt iifile3.txt iifile4.txt iifile5.txt iifile6.txt iifile7.txt iifile8.txt iifile9.txt iifile10.txt
enter file 1 to choose inputs for x: iifile2.txt
enter file 2 to choose inputs for y: iifile7.txt
15965 5040 2256 12555 19939 3140 15633 13034 19182 25050 15872 3591 768 12626 14070 16728 15416 14382 26373 22869 26878
30200 32550 10547 10241 1298 6460 25875 34680 29250 7089 784 6417 7396 23049 12040 1552 8547 3689 12805 20274 12474 3438
0 21504 4848 18522 3706 17214 13689 4553 14442 8949 3328 18180 4375

Process exited after 10.26 seconds with return value 0
Press any key to continue . . .


C:\Users\Zahid's PC\OneDrive
displaying file names:
iifile1.txt iifile2.txt iifile3.txt iifile4.txt iifile5.txt iifile6.txt iifile7.txt iifile8.txt iifile9.txt iifile10.txt
enter file 1 to choose inputs for x: iifile3.txt
enter file 2 to choose inputs for y: iifile9.txt
2204 7600 2028 14852 22010 21008 12180 14848 8750 1809 30560 35000 3297 11115 15496 18093 22990 31714 42849 7503 8112 59
76 10206 25280 3658 7154 13442 5200 16732 350 2046 27946 5580 24360 4488 2550 924 8232 5439 2880 11640 11431 27632 11716
3675 3844 5440 18582 7592 1443 20294 16585 4116

Process exited after 6.927 seconds with return value 0
Press any key to continue . . .

Question 4:

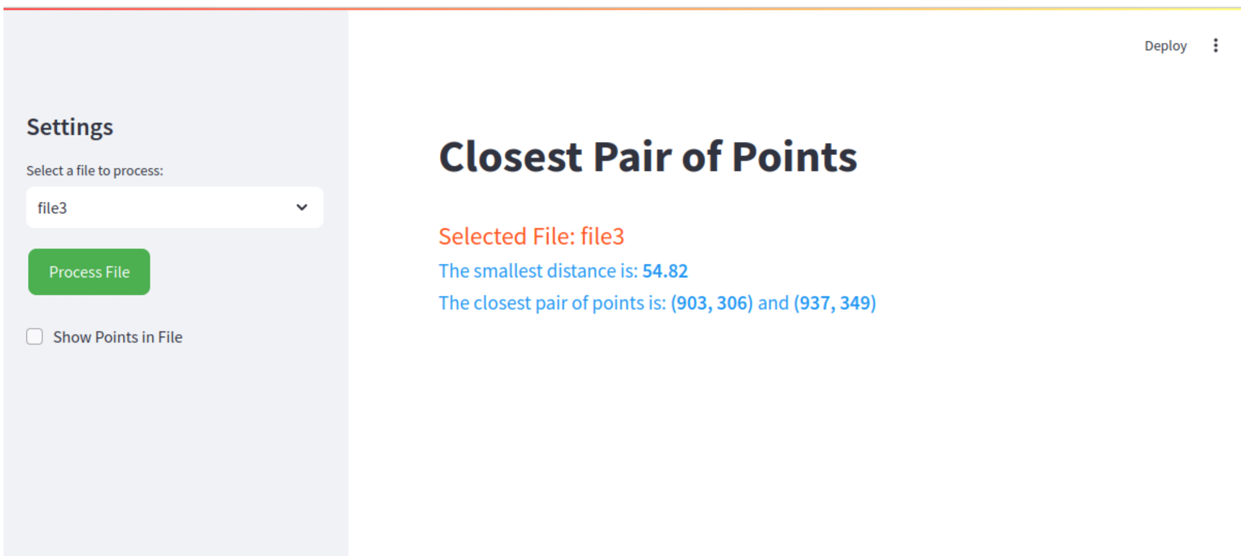
i

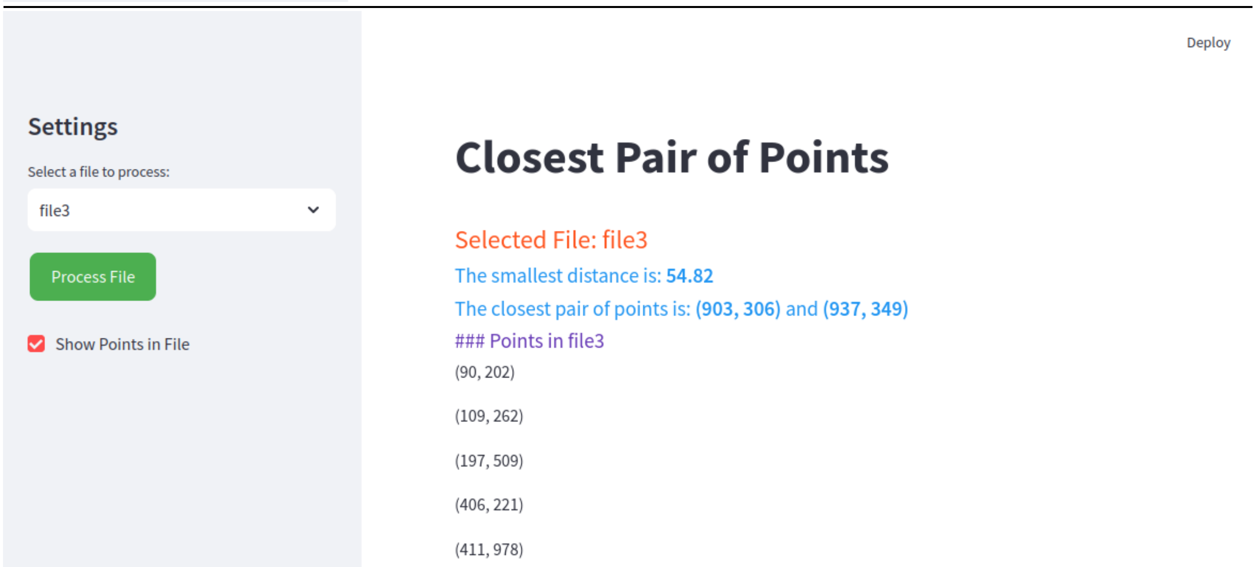
```
import streamlit as st
import math
import random
from typing import List

class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

closest_pair = [None, None]

def calculate_distance(p1: Point, p2: Point) -> float:
    return math.sqrt((p1.x - p2.x) ** 2 + (p1.y - p2.y) ** 2)

def brute_force_closest(points: List[Point]) -> float:
    global closest_pair
    min_distance = float('inf')
    n = len(points)
    for i in range(n):
        for j in range(i + 1, n):
            dist = calculate_distance(points[i], points[j])
            if dist < min_distance:
                min_distance = dist
                closest_pair[0], closest_pair[1] = points[i], points[j]
    return min_distance

def closest_in_strip(strip: List[Point], d: float) -> float:
    global closest_pair
    strip.sort(key=lambda p: p.y)
    min_distance = d
    for i in range(len(strip)):
        for j in range(i + 1, len(strip)):
            if (strip[j].y - strip[i].y) < min_distance:
                dist = calculate_distance(strip[i], strip[j])
                if dist < min_distance:
                    min_distance = dist
                    closest_pair[0], closest_pair[1] = strip[i], strip[j]
    return min_distance

def closest_pair_recursive(points: List[Point]) -> float:
    if len(points) <= 3:
```

```
        return brute_force_closest(points)

    mid = len(points) // 2
    mid_point = points[mid]

    dl = closest_pair_recursive(points[:mid])
    dr = closest_pair_recursive(points[mid:])
    d = min(dl, dr)

    strip = [p for p in points if abs(p.x - mid_point.x) < d]
    return min(d, closest_in_strip(strip, d))

def find_closest(points: List[Point]) -> float:
    points.sort(key=lambda p: p.x)
    return closest_pair_recursive(points)

st.title("Closest Pair of Points")
st.sidebar.title("Settings")
# Add custom CSS for colors and styling
st.markdown(
    """
    <style>
    .main-title {
        font-size: 2em;
        color: #4CAF50;
        text-align: center;
        margin-bottom: 20px;
    }
    .file-title {
        font-size: 1.5em;
        color: #FF5722;
    }
    .distance-text {
        font-size: 1.2em;
        color: #2196F3;
    }
    .points-text {
        font-size: 1.2em;
        color: #673AB7;
    }
    .custom-button {
        background-color: #4CAF50; /* Green */
        border: none;
        color: white;
        padding: 10px 20px;
        text-align: center;
```

```
        text-decoration: none;
        display: inline-block;
        font-size: 16px;
        margin: 4px 2px;
        cursor: pointer;
        border-radius: 8px;
    }
    .custom-checkbox {
        font-size: 1.2em;
        color: #FF9800;
    }
    </style>
    """,
    unsafe_allow_html=True,
)

# Generate random files
if 'generated_files' not in st.session_state:
    st.session_state.generated_files = []
    for i in range(10):
        num_points = random.randint(10, 50)
        points = [Point(random.randint(0, 1000), random.randint(0, 1000)) for _ in range(num_points)]
        st.session_state.generated_files.append(points)

file_index = st.sidebar.selectbox(
    "Select a file to process:",
    [f"file{i}" for i in range(len(st.session_state.generated_files))]
)

# Add a custom button for file processing
if st.sidebar.markdown('<button class="custom-button">Process File</button>', unsafe_allow_html=True):
    selected_points = st.session_state.generated_files[int(file_index[-1])]
    smallest_distance = find_closest(selected_points)
    p1, p2 = closest_pair

    st.markdown(f'<div class="file-title">Selected File: {file_index}</div>', unsafe_allow_html=True)
    st.markdown(f'<div class="distance-text">The smallest distance is: <b>{smallest_distance:.2f}</b></div>',
unsafe_allow_html=True)
    st.markdown(
        f'<div class="distance-text">The closest pair of points is: <b>({p1.x}, {p1.y})</b> and <b>({p2.x},
{p2.y})</b></div>',
        unsafe_allow_html=True,
    )

# Checkbox to show points
if st.sidebar.checkbox("Show Points in File"):
```

```
st.markdown(f'<div class="points-text">### Points in {file_index}</div>', unsafe_allow_html=True)
for point in st.session_state.generated_files[int(file_index[-1])]:
    st.write(f"({point.x}, {point.y})")
```



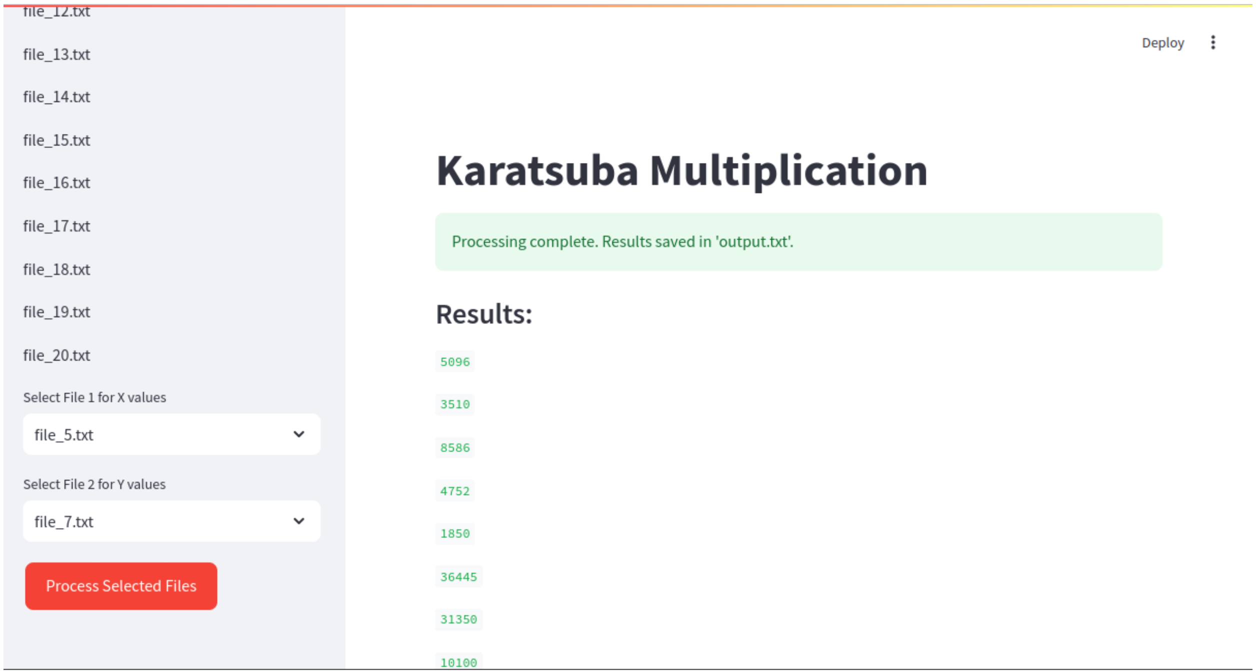


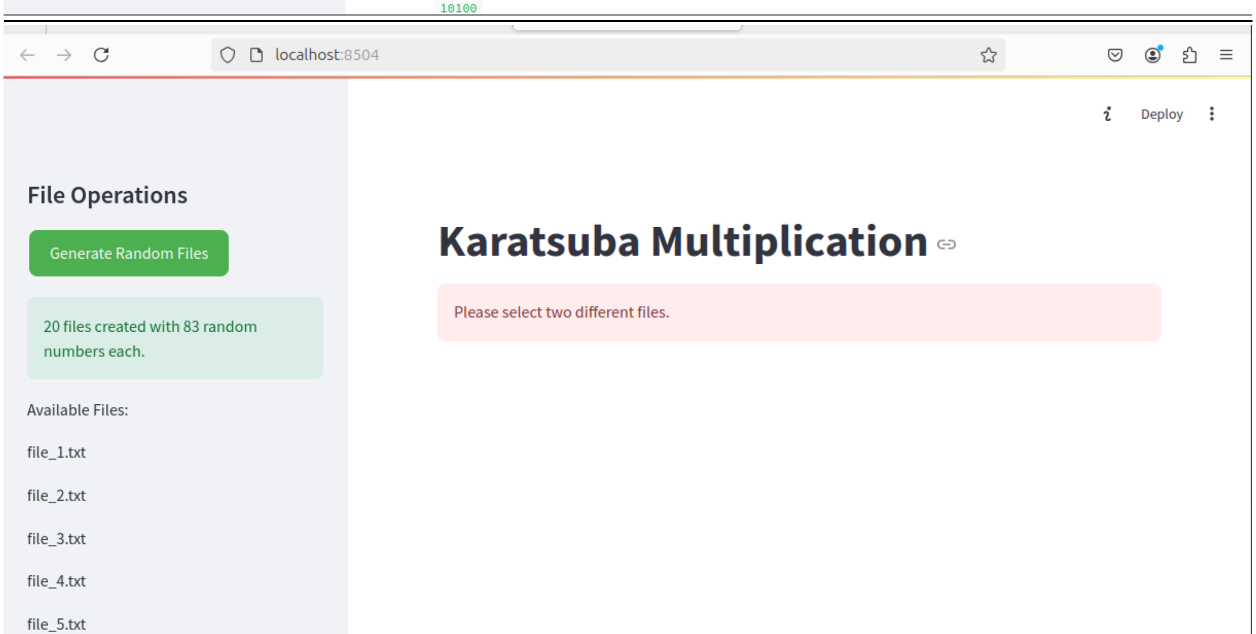
Ii

```
import os
import random
import math
import streamlit as st
# Custom CSS for button styling
st.markdown(
    """
    <style>
    .custom-button {
        background-color: #4CAF50; /* Green */
        border: none;
        color: white;
        padding: 10px 20px;
        text-align: center;
        text-decoration: none;
        display: inline-block;
        font-size: 16px;
        margin: 4px 2px;
        cursor: pointer;
        border-radius: 8px;
    }
    .custom-button-red{
        background-color: #f44336;
        border: none;
        color: white;
        padding: 10px 20px;
        text-align: center;
        text-decoration: none;
        display: inline-block;
        font-size: 16px;
        margin: 4px 2px;
        cursor: pointer;
        border-radius: 8px;
    }
    </style>
    """,
    unsafe_allow_html=True
)

def gen_file_name(idx):
    return f"file_{idx + 1}.txt"

def create_files():
    random.seed()
```

```
    n_nums = random.randint(50, 150)
    for i in range(20):
        fname = gen_file_name(i)
        with open(fname, "w") as f:
            for _ in range(n_nums):
                f.write(f"{random.randint(10, 209)}\n")
    return n_nums

def karatsuba(x, y):
    if x < 10 or y < 10:  # Base case
        return x * y

    n = max(len(str(x)), len(str(y)))
    half = n // 2
    high1, low1 = divmod(x, 10 ** half)
    high2, low2 = divmod(y, 10 ** half)

    z1 = karatsuba(high1, high2)
    z2 = karatsuba(low1, low2)
    z3 = karatsuba(high1 + low1, high2 + low2)

    return z1 * (10 ** (2 * half)) + (z3 - z1 - z2) * (10 ** half) + z2

st.title("Karatsuba Multiplication")
st.sidebar.title("File Operations")
# Add colored button for generating files
if st.sidebar.markdown('<button class="custom-button">Generate Random Files</button>',
unsafe_allow_html=True):
    n_nums = create_files()
    st.sidebar.success(f"20 files created with {n_nums} random numbers each.")

file_list = [gen_file_name(i) for i in range(20) if os.path.exists(gen_file_name(i))]
if file_list:
    st.sidebar.write("Available Files:")
    for f in file_list:
        st.sidebar.write(f)

file1 = st.sidebar.selectbox("Select File 1 for X values", file_list)
file2 = st.sidebar.selectbox("Select File 2 for Y values", file_list)

# Add colored button for processing files
if st.sidebar.markdown('<button class="custom-button-red">Process Selected Files</button>',
unsafe_allow_html=True):
    if not file1 or not file2:
        st.error("Please select two valid files.")
    elif file1 == file2:
```

```
    st.error("Please select two different files.")
else:
    x_vals = []
    y_vals = []
    results = []

    try:
        with open(file1, "r") as f1, open(file2, "r") as f2:
            x_lines = f1.readlines()
            y_lines = f2.readlines()

            for x_line, y_line in zip(x_lines, y_lines):
                x = int(x_line.strip())
                y = int(y_line.strip())
                x_vals.append(x)
                y_vals.append(y)

                prod = karatsuba(x, y)
                results.append(prod)

        with open("output.txt", "w") as out:
            for res in results:
                out.write(f"{res}\n")

        st.success("Processing complete. Results saved in 'output.txt'.")
        st.write("### Results:")
        for r in results:
            st.write(r)
    except ValueError as e:
        st.error(f"Error processing files: {e}")
```


file_12.txt
Deploy
file_13.txt
file_14.txt
file_15.txt
file_16.txt
file_17.txt
file_18.txt
file_19.txt
file_20.txt
Select File 1 for X values
file_5.txt
Select File 2 for Y values
file_7.txt
Process Selected Files
Karatsuba Multiplication
Processing complete. Results saved in 'output.txt'.
Results:
5096
3510
8586
4752
1850
36445
31350
10100


localhost:8504
Deploy
File Operations
Generate Random Files
20 files created with 83 random numbers each.
Available Files:
file_1.txt
file_2.txt
file_3.txt
file_4.txt
file_5.txt
Karatsuba Multiplication
Please select two different files.