学号：　201618013229011
姓名：　李坚松

Assignment 1

Problem-1

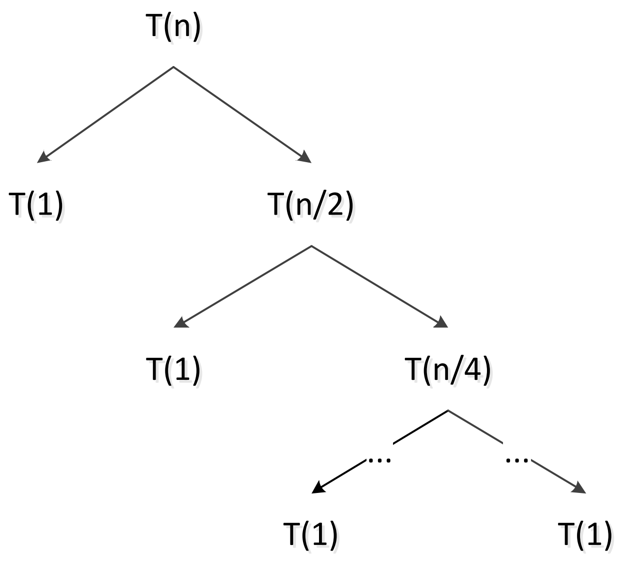
Solution

The problem is to let us to find the n-th smallest value of the two databases, we can take use of the idea of divide and conquer to solve it, which is similar to binary-search. To begin with, we define the function query(database &d, int k) to get the k-th smallest value of database d. We mark the two databases by A and B. We can get the medians of A and B which equal query(A,n/2) and query(B,n/2), separately marked by m1 and m2. Then database A and B is divided into two subsets. Compare m1 and m2, if m1 > m2, then the median of A and B must in the range of (n/2)-th smallest to the minimum of A, and (n/2)-th smallest to the maximum of B, otherwise, the median of A and B must in the range of (n/2)-th smallest to the minimum of B, and (n/2)-th smallest to the maximum of A. We can iteratively find the median of the range until they only have one element, then the minimum of medians of two subsets is the answer. The pseudo code of the algorithm is shown below:

```
Algorithm: GetMedian ( database &A, database &B, int n )
Input: database A, database B, both A and B contains n values
Output: the n-th smallest of A and B
Begin
    int la = lb = 1, ha = hb = n, mida = ( la + ha ) / 2, midb = ( lb + hb ) / 2;
    while( ha >= la ) //until search range is reduced to 1
    {
        mida = ( la + ha ) / 2;
        midb = ( lb + hb ) / 2;
        if( query( A, mida ) > query( B, midb ) )
        {
            ha = mida;
            lb = midb;
        }
        else
        {
            la = mida;
            hb = midb;
        }
    }
    return min( query( A, mida ), query( B, midb ) );
End
```

Sub-problem Reduction Graph

The sub-problem reduction graph is shown below:



Proof

We can use loop invariant technique to prove its correctness.

Initialization: la and lb are pointers that point to the lower bound of database A and B, ha and hb are pointers that point to the higher bound of A and B, mida and midb are pointers that point to the median position of A and B.

Maintenance: suppose query(A, mida) > query(B, midb), the global median should be in the smaller part of A and bigger part of B. Then the search range reduced by half, so the boundary pointer should be updated accordingly. Otherwise, the search range also should be updated accordingly. During the while loop, loop invariant holds, and the algorithm searches in the potential range.

Termination: At termination, pointer ha and la points to the same position, which means the search range varies from n to n/2, n/4,…1, after log(n) rounds, the search range is reduced to 1, query(A, mida) and query(B, midb) is the n-th or (n+1)-th smallest of A and B. According to definition, we choose the smaller one to be final median.

Time Complexity

For each step, the search range is reduced by half, and we make two queries, one comparison and update two pointers, T(n)=0+T(n/2)+c, therefore, the time complexity is O(logn).

Problem-2

Solution

The problem is to find out the k-th largest element in an unsorted array. We can use the idea of divide and conquer to solve it, which is similar to quick sort. Suppose the array contains n elements, which is marked by A[0..n-1], where k <= n. We randomly choose one element as the pivot, for example the first one, scan A, for current element, if it is greater than pivot, then move it to left part of A, else move it to the right part of A. After the first pass, the left part is greater than pivot, the right part is no more than pivot. We get the position of pivot in A, if it equals k-1, then pivot is the answer. Else if the pivot's position is less than k-1, we iteratively use the idea of partition for the right part, else we apply the idea to the left part, until the pivot's position is k-1, then return A[k-1]. The pseudo code of the algorithm is shown below:

```
Algorithm: GetKMax ( ElementType A[], int k )
Input: an unsorted array A and k
Output: the k-th largest element of A
Begin
    int start = 0, end = A.length()-1;
    int index = Partition( A, start, end );
    while( index != k-1 )
    {
        if( index > k-1 )
        {
            end = index - 1;
            index = Partition( A, start, end );
        }
        else
        {
            start = index + 1;
            index = Partition( A, start, end );
        }
    }
    return A[ index ];
End
```

The algorithm Partition is similar to quick sort, it returns the real position of pivot in the array B, its pseudo code is shown below:

Algorithm: Partition (ElementType B[], int start, int end)

Input: an unsorted array B, start and end position of B

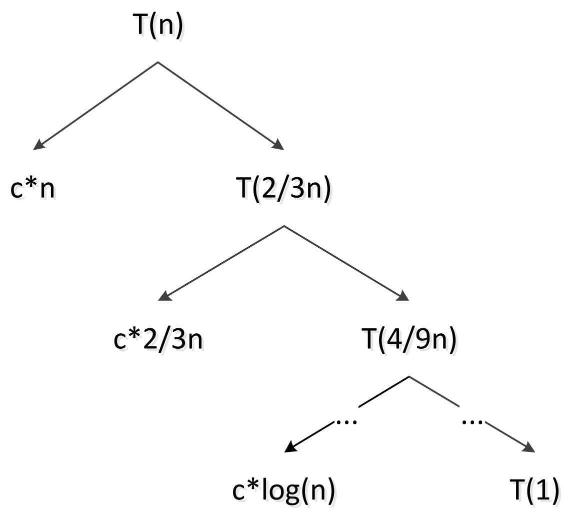
Output: the real sorted position of pivot in the array B

```
Begin
    int i = start, j = end;
    int temp;
    if(start < end )
    {
        temp = B[start];
        while( i != j )
        {
            while( j > i && B[j] < temp)
                j--;
            B[i] = B[j];
            while( j > i && B[i] >= temp)
                i++;
            B[j] = B[i];
        }
        B[i] = temp;
    }
    return i;
End
```

Sub-problem Reduction Graph

Just like modified quick sort, we modified the algorithm GetKMax. At every recursive step, we take some policy to discard fixed length, then the scale of the problem will be reduced by a constant. For example, every step, we discard 1/3, then recursively call the Partition method for the left 2/3, then the whole time would be cn+(2/3)*cn+(2/3)^2*cn+…+(2/3)^j*cn+…,

The sub-problem reduction graph is shown below:



Proof

We can use loop invariant technique to prove its correctness.

Initialization: After the first pass, A will be divided into two parts, where the pivot is in the real position.

Maintenance: Suppose the pivot's position is i, if i > k, then the k-th largest element must be in the left parts, otherwise, it is in the right part. During the while loop, loop invariant holds, and the algorithm searches in the potential part.

Termination: At termination, i = k-1, which means we get the k-th largest element of the whole array.

Time Complexity

Just like modified quick sort, we modified the algorithm GetKMax. At every recursive step, we take some policy to discard fixed length, then the scale of the problem will be reduced by a constant. For example, every step, we discard 1/3, then recursively call the Partition method for the left 2/3, then the whole time would be cn+(2/3)*cn+(2/3)^2*cn+…+(2/3)^j*cn+…, it is a geometrical ratio, so the average time complexity is O(n).

Problem-3

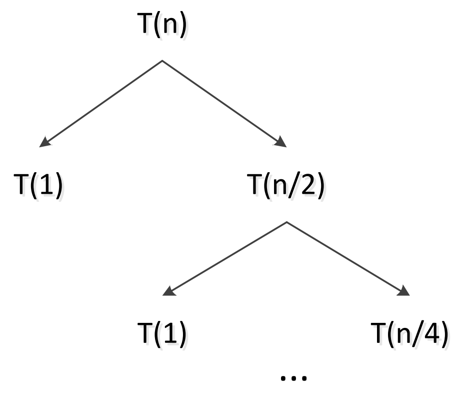
Solution

To find a local minimum of T, we can probe from the root of T. If T has no children nodes, then root's value is the local minimum. If the root's value is smaller than its children, then the root's value is the local minimum. Otherwise, we choose the child node whose value is smaller, then we can recursively find the local minimum in the children sub-tree. Eventually, we would find a local minimum. The pseudo code of the algorithm is shown below:

```
Algorithm: FindLocalMinimum ( T *root )
Input: complete binary tree whose root node points to root.
Output: a local minimum of the complete binary tree
Begin
    //root has no children, return its value.
    if ( root->lchild == null && root-> rchild == null )
        return root->value;
    if (root->value < root->lchild->value && root->value < root->rchild->value )
        return root->value;
    else
    {
        if( root->lchild->value < root->rchild->value)
            return FindLocalMinimum( root->lchild );
        else
            return FindLocalMinimum( root->rchild );
    }
End
```

Sub-problem Reduction Graph

If the root's value is not the local minimum, then we find it at its children sub-tree, problem size reduces by half.The sub-problem reduction graph is shown below:



Proof

Initialization: The complete binary tree has only one node, which meets the definition of local minimum, so its value is the local minimum.

Maintenance: If the root’s value is smaller than its children, then according to the definition, the root’s value is the local minimum. Otherwise, we choose the child node whose value is smaller, then we recursively find the local minimum in the children sub-tree. During the recursion, loop invariant holds, and the algorithm searches in the potential range.

Termination: At the node v, if v is not the leaf node, and v’s value is smaller than its child nodes and parent, then its value is a local minimum. Otherwise, we will find at the leaf node, one of the leaf node would be a local minimum.

Time Complexity

T(n) = T(n/2) + c, so the time complexity is O(logn).

Problem-9

Input is n points in a plane, first line is n, which is the number of points, next n lines is the x and y coordinates of the point.

Output is the pair with the least Euclidean distance.

Implement

Following is the source code.

```
/*
	Name: Closest Pair
	Copyright:
	Author: Json Lee
	Date: 27/09/16 21:49
	Description: divide and conquer
*/
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
using namespace std;
const double INF = 1e20;//INF is large enough
const int N = 100005;//max points numbers
//define the point struct, x and y is the value of point’s coordinate
```

```
struct Point
{
        double x;
        double y;
}point[N];
int n;//number of points
int tmpt[N];//temporary array to store the points' order

//bool function, for sorting by x-coordinate
bool cmpxy(const Point& a, const Point& b)
{
        if(a.x != b.x)
                return a.x < b.x;
        return a.y < b.y;
}

//bool function, for sorting by y-coordinate
bool cmpy(const int& a, const int& b)
{
        return point[a].y < point[b].y;
}

//return the minimum of a and b
double min(double a, double b)
{
        return a < b ? a : b;
}

//return the Euclidean distance of point[i] and point[j]
double dis(int i, int j)
{
        return sqrt((point[i].x-point[j].x)*(point[i].x-point[j].x)
                                + (point[i].y-point[j].y)*(point[i].y-point[j].y));
}

//return distance of    closest pair
double Closest_Pair(int left, int right)
{
        //initilize d to large enough
        double d = INF;
        if(left==right)
                return d;
        if(left + 1 == right)
                return dis(left, right);
        int mid = (left+right)>>1;//(left+right)/2
        //get the left part closest distance
        double d1 = Closest_Pair(left,mid);
        //get the right part closest distance
        double d2 = Closest_Pair(mid+1,right);
        d = min(d1,d2);
        //conquer to get the closest distance that between left and right
        int i,j,k=0;
        //get the section whose width is d
        for(i = left; i <= right; i++)
        {
                if(fabs(point[mid].x-point[i].x) <= d)
                        tmpt[k++] = i;
        }
        //sort by y-coordinate
        sort(tmpt,tmpt+k,cmpy);
        //linear scan
        for(i = 0; i < k; i++)
        {
                for(j = i+1; j < k && point[tmpt[j]].y-point[tmpt[i]].y<d; j++)
                {
                        double d3 = dis(tmpt[i],tmpt[j]);
                        if(d > d3)
                                d = d3;
                }
        }
        return d;
}

int main()
{
        freopen("p9-in.txt","r",stdin);
        while(scanf("%d",&n)!=EOF)
        {
                for(int i = 0; i < n; i++)
                        scanf("%lf %lf",&point[i].x,&point[i].y);
                //sort by x-coordinate
                sort(point,point+n,cmpxy);
                printf("closest distance: %.2lf\n",Closest_Pair(0,n-1));
        }
        fclose(stdin);
        return 0;
}
```

Problem-10

Pseudo Code of Strassen

Algorithm: Strassen (int n , Matrix &A, Matrix &B, Matrix &Result)

Input: the dimension of Result is n, matrix A, matrix B, and the Result matrix
Output: the result of A * B

```
Begin
    // splitting A and B into 4 submatrices
    for i = 0 to n/2
    {
        for j = 0 to n/2
        {
            A11[ i ][ j ] = A [ i ][ j ];
            A12[ i ][ j ] = A[ i ][ j + n/2 ];
            A21[ i ][ j ] = A[ i + n/2 ][ j ];
            A22[ i ][ j ] = A[ i + n/2 ][ j + n/2 ];

            B11[ i ][ j ] = B[ i ][ j ];
            B12[ i ][ j ] = B[ i ][ j + n/2 ];
            B11[ i ][ j ] = B[ i + n/2 ][ j ];
            B11[ i ][ j ] = B[ i + n/2 ][ j + n/2 ];
        }
    }

    Matrix TA,TB;
    int half;
    //recursively calculate P1 P2 P3 P4 P5 P6 P7
    half = n/2;
    TB = B12 – B22;
    Strassen( half, A11, TB, P1 );

    TA = A11 + A12;
    Strassen( half, TA, B22, P2 );

    TA = A21 + A22;
    Strassen( half, TA, B11, P3 );

    TB = B21 – B11;
    Strassen( half, A22, TB, P4 );
    TA = A11 + A22;
    TB = B11 + B22;
    Strassen( half, TA, TB, P5 );

    TA = A12 – A 22;
    TB = B21 + B22;
    Strassen( half, TA, TB, P6 );

    TA = A11 – A21;
    TB = B11 + B12;
    Strassen( half, TA, TB, P7 );

    //calculate the submatrices of the Result matrix
    C11 = P4 + P5 +P6 – P2;
    C12 = P1 + P2;
    C21 = P3 + P4;
    C22 = P1 + P5 – P3 – P7;
End
```

Implement of Strassen

The source code of Strassen in cpp is shown below:

```
/*
    Name: Strassen
    Copyright:
    Author: Json Lee
    Date: 26/09/16 16:10
    Description:
    Divide and Conquer,
    P1=A11*(B12-B22)
    P2=(A11+A12)*B22
    P3=(A21+A22)*B11
    P4=A22*(B21-B11)
    P5=(A11+A22)*(B11+B22)
    P6=(A12-A22)*(B21+B22)
    P7=(A11-A21)*(B11+B12)

    C11=P4+P5+P6-P2
    C12=P1+P2
    C21=P3+P4
    C22=P1+P5-P3-P7
*/
```

```
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;

//two matrices add operation
void add(int** MatrixA, int** MatrixB, int** MatrixResult, int MatrixSize)
{
	for (int i = 0; i < MatrixSize; i++)
	{
		for (int j = 0; j < MatrixSize; j++)
		{
			MatrixResult[i][j] = MatrixA[i][j] + MatrixB[i][j];
		}
	}
}

//two matrices subtraction operation
void sub(int** MatrixA, int** MatrixB, int** MatrixResult, int MatrixSize)
{
	for (int i = 0; i < MatrixSize; i++)
	{
		for (int j = 0; j < MatrixSize; j++)
		{
			MatrixResult[i][j] = MatrixA[i][j] - MatrixB[i][j];
		}
	}
}

//two matrices multiply operation
void mul(int** MatrixA, int** MatrixB, int** MatrixResult, int MatrixSize)
{
	for (int i = 0; i<MatrixSize; i++)
	{
		for (int j = 0; j<MatrixSize; j++)
		{
			MatrixResult[i][j] = 0;
			for (int k = 0; k<MatrixSize; k++)
			{
				MatrixResult[i][j] = MatrixResult[i][j] + MatrixA[i][k] * MatrixB[k][j];
			}
		}
	}
```

```
}

void strassen(int N, int **MatrixA, int **MatrixB, int **MatrixC)
{
    int HalfSize = N / 2;
    int newSize = N / 2;

    if (N <= 64)     //n<=64 take normal method
    {
        mul(MatrixA, MatrixB, MatrixC, N);
    }
    else
    {
        int** A11;int** A12;int** A21;int** A22;
        int** B11;int** B12;int** B21;int** B22;
        int** C11;int** C12;int** C21;int** C22;
        int** M1;int** M2;int** M3;int** M4;int** M5;int** M6;int** M7;
        int** AResult;int** BResult;

        //making a 1 diminsional pointer based array.
        A11 = new int *[newSize];A12 = new int *[newSize];A21 = new int *[newSize];A22 =
new int *[newSize];
        B11 = new int *[newSize];B12 = new int *[newSize];B21 = new int *[newSize];B22 =
new int *[newSize];
        C11 = new int *[newSize];C12 = new int *[newSize];C21 = new int *[newSize];C22 =
new int *[newSize];
        M1 = new int *[newSize];M2 = new int *[newSize];M3 = new int *[newSize];M4 =
new int *[newSize];M5 = new int *[newSize];M6 = new int *[newSize];M7 = new int
*[newSize];
        AResult = new int *[newSize];BResult = new int *[newSize];

        int newLength = newSize;

        //making that 1 diminsional pointer based array , a 2D pointer based array
        for (int i = 0; i < newSize; i++)
        {
            A11[i] = new int[newLength];
            A12[i] = new int[newLength];
            A21[i] = new int[newLength];
            A22[i] = new int[newLength];

            B11[i] = new int[newLength];
            B12[i] = new int[newLength];
            B21[i] = new int[newLength];
            B22[i] = new int[newLength];

            C11[i] = new int[newLength];
            C12[i] = new int[newLength];
            C21[i] = new int[newLength];
            C22[i] = new int[newLength];

            M1[i] = new int[newLength];
            M2[i] = new int[newLength];
            M3[i] = new int[newLength];
            M4[i] = new int[newLength];
            M5[i] = new int[newLength];
            M6[i] = new int[newLength];
            M7[i] = new int[newLength];

            AResult[i] = new int[newLength];
            BResult[i] = new int[newLength];
        }
        //splitting input Matrixes, into 4 submatrices each.
        for (int i = 0; i < N / 2; i++)
        {
            for (int j = 0; j < N / 2; j++)
            {
                    A11[i][j] = MatrixA[i][j];
                    A12[i][j] = MatrixA[i][j + N / 2];
                    A21[i][j] = MatrixA[i + N / 2][j];
                    A22[i][j] = MatrixA[i + N / 2][j + N / 2];

                    B11[i][j] = MatrixB[i][j];
                    B12[i][j] = MatrixB[i][j + N / 2];
                    B21[i][j] = MatrixB[i + N / 2][j];
                    B22[i][j] = MatrixB[i + N / 2][j + N / 2];

            }
        }

        //P5[][]
        add(A11, A22, AResult, HalfSize);
        add(B11, B22, BResult, HalfSize);
        strassen(HalfSize, AResult, BResult, M1);

        //P3[][]
        add(A21, A22, AResult, HalfSize);
        strassen(HalfSize, AResult, B11, M2);
```

```
//P1[][]
sub(B12, B22, BResult, HalfSize);
strassen(HalfSize, A11, BResult, M3);

//P4[][]
sub(B21, B11, BResult, HalfSize);
strassen(HalfSize, A22, BResult, M4);

//P2[][]
add(A11, A12, AResult, HalfSize);
strassen(HalfSize, AResult, B22, M5);



//P7[][]
sub(A21, A11, AResult, HalfSize);
add(B11, B12, BResult, HalfSize);
strassen(HalfSize, AResult, BResult, M6);

//P6[][]
sub(A12, A22, AResult, HalfSize);
add(B21, B22, BResult, HalfSize);
strassen(HalfSize, AResult, BResult, M7);

//C11
add(M1, M4, AResult, HalfSize);
sub(M7, M5, BResult, HalfSize);
add(AResult, BResult, C11, HalfSize);

//C12
add(M3, M5, C12, HalfSize);

//C21
add(M2, M4, C21, HalfSize);

//C22
add(M1, M3, AResult, HalfSize);
sub(M6, M2, BResult, HalfSize);
add(AResult, BResult, C22, HalfSize);

//combine c11 c12 c21 c22 to get the result Matrix.
for (int i = 0; i < N / 2; i++)
{
        for (int j = 0; j < N / 2; j++)
```

```
                {
                        MatrixC[i][j] = C11[i][j];
                        MatrixC[i][j + N / 2] = C12[i][j];
                        MatrixC[i + N / 2][j] = C21[i][j];
                        MatrixC[i + N / 2][j + N / 2] = C22[i][j];
                }
        }

        // release
        for (int i = 0; i < newLength; i++)
        {
                delete[] A11[i]; delete[] A12[i]; delete[] A21[i];
                delete[] A22[i];

                delete[] B11[i]; delete[] B12[i]; delete[] B21[i];
                delete[] B22[i];
                delete[] C11[i]; delete[] C12[i]; delete[] C21[i];
                delete[] C22[i];
                delete[] M1[i]; delete[] M2[i]; delete[] M3[i]; delete[] M4[i];
                delete[] M5[i]; delete[] M6[i]; delete[] M7[i];
                delete[] AResult[i]; delete[] BResult[i];
        }
        delete[] A11; delete[] A12; delete[] A21; delete[] A22;
        delete[] B11; delete[] B12; delete[] B21; delete[] B22;
        delete[] C11; delete[] C12; delete[] C21; delete[] C22;
        delete[] M1; delete[] M2; delete[] M3; delete[] M4; delete[] M5;
        delete[] M6; delete[] M7;
        delete[] AResult;
        delete[] BResult;
    }
}

//generate test matrices
void fill_matrix(int** MatrixA, int** MatrixB, int length)
{
    for (int row = 0; row<length; row++)
    {
        for (int column = 0; column<length; column++)
        {

                MatrixB[row][column] = (MatrixA[row][column] = rand() % 5);
                //matrix2[row][column] = rand() % 2;
        }
    }
```

```
}

int main()
{
        freopen("p10-in.txt","r",stdin);
        freopen("p10-out.txt","w",stdout);
        //using system rand to generate test matrices
        int MatrixSize = 0;
        int** MatrixA;
        int** MatrixB;
        int** MatrixC;

        while(cin >> MatrixSize)
        {
                int N = MatrixSize;
                cout<<"matrix size: "<<N<<endl;
                MatrixA = new int *[MatrixSize];
                MatrixB = new int *[MatrixSize];
                MatrixC = new int *[MatrixSize];

                for (int i = 0; i < MatrixSize; i++)
                {
                        MatrixA[i] = new int[MatrixSize];
                        MatrixB[i] = new int[MatrixSize];
                        MatrixC[i] = new int[MatrixSize];
                }

                //generate test matrices
                fill_matrix(MatrixA, MatrixB, MatrixSize);

                //using system clock to measure time consumption
                clock_t t1, t2;
                double duration;//time duration
                t1 = clock();
                mul(MatrixA, MatrixB, MatrixC, MatrixSize);
                t2 = clock();
                duration = (double)(t2 - t1) / CLOCKS_PER_SEC;
                cout << "grade school method time consumption: " << duration << " s" << endl;

                t1 = clock();
                strassen(N, MatrixA, MatrixB, MatrixC);
                t2 = clock();
                duration = (double)(t2 - t1) / CLOCKS_PER_SEC;
                cout << "strassen method time consumption: " << duration << " s" << endl;
        }

        return 0;
}
```

Implement of grade-school method

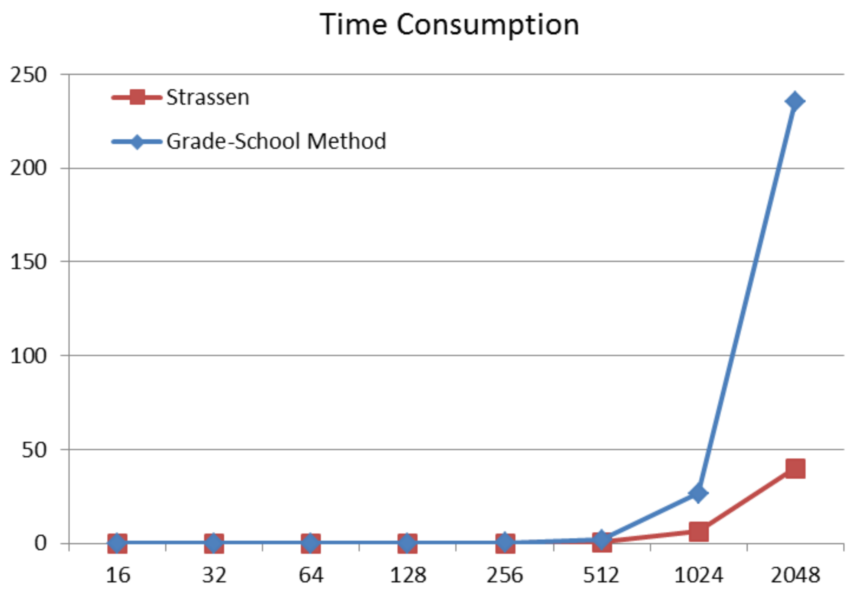
```
void multiply(int** MatrixA, int** MatrixB, int** MatrixResult, int MatrixSize)
{
        for (int i = 0; i<MatrixSize; i++)
        {
                for (int j = 0; j<MatrixSize; j++)
                {
                        MatrixResult[i][j] = 0;
                        for (int k = 0; k<MatrixSize; k++)
                        {
                                MatrixResult[i][j] = MatrixResult[i][j] + MatrixA[i][k] * MatrixB[k][j];
                        }
                }
        }
}
```

Comparison

Test environment: Processor: Intel Core i5-3230M, two cores, CPU: 2.6GHz, 8GB DDR3 Memory. After multiple tests, we can get the table below:

Matrix Size(n*n)	**Time Consumption (s)**	
	Grade-School Method	**Strassen**
16	0	0.001
32	0	0
64	0.002	0.003
128	0.019	0.022
256	0.166	0.098
512	1.82	0.704
1024	26.801	6.224
2048	235.117	40.02

According to the data above, we can get the following line chart:



As is shown in the picture, we can get the following conclusion:

When the matrix size is small, grade-school method would be better. However, when the matrix size is large enough, strassen method will outperform grade-school method.