**Assignment 4**

**Source code:**

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

template<typename T>

class Matrix

{ public:

int n, m;

std::vector<std::vector<T>> arr;

Matrix(int i)

{

n = i; m = i;

arr.resize(n, std::vector<T>(n));

}

Matrix(int i, int j, T x)

{

n = i; m = j;

arr.resize(n, std::vector<T>(m, x));

}

void size\_update()

{

n = arr.size(); m = arr[0].size();

}

Matrix operator+(Matrix &a)

{

Matrix<T> tmp(n, m, 0);

for(int i=0 ;i<n; i++)

for(int j=0; j<m; j++)

tmp.arr[i][j] = arr[i][j] + a.arr[i][j];

return tmp;

}

Matrix operator-(Matrix &a)

{

Matrix<T> tmp(n, m, 0);

for(int i=0 ;i<n; i++)

for(int j=0; j<m; j++)

tmp.arr[i][j] = arr[i][j] - a.arr[i][j];

return tmp;

}

Matrix operator()(int r1, int r2, int c1, int c2)

{

Matrix<T> tmp (r2-r1, c2-c1, 0);

for(int i = r1; i < r2; i++)

{

for(int j = c1; j < c2; j++)

tmp.arr[i-r1][j-c1] = arr[i][j];

}

return tmp;

}

void assign(Matrix &&a, int r1, int r2, int c1, int c2)

{

for(int i=0; i<a.n; i++)

for(int j=0; j<a.m; j++)

arr[i+r1][j+c1] = a.arr[i][j];

}

void assign(Matrix &&a)

{

for(int i=0; i<a.n; i++)

for(int j=0; j<a.m; j++)

arr[i][j] = a.arr[i][j];

}

void display()

{

for(auto i: arr)

{

for(auto j: i)

cout << j << "\t";

cout << "\n";

}

cout << "\n";

}

Matrix matmul(Matrix &a);

};

template<typename T>

Matrix<T> pad\_even(Matrix<T> &x, int t)

{

Matrix<T> tmp(x.n + t);

tmp.assign(move(x), t, x.n, t, x.n);

return tmp;

}

template<typename T>

Matrix<T> pad\_odd(Matrix<T> &x, int t)

{

auto tmp = x(t, x.n, t, x.n);

return tmp;

}

template<typename T>

Matrix<T> matmul(Matrix<T> &&X, Matrix<T> &&Y)

{

Matrix<T> Z (X.n);

int n = X.n;

if(X.n == 2 && Y.n == 2)

{

Z.arr[0][0] = X.arr[0][0]\*Y.arr[0][0] + X.arr[0][1]\*Y.arr[1][0];

Z.arr[0][1] = X.arr[0][0]\*Y.arr[0][1] + X.arr[0][1]\*Y.arr[1][1];

Z.arr[1][0] = X.arr[1][0]\*Y.arr[0][0] + X.arr[1][1]\*Y.arr[1][0];

Z.arr[1][1] = X.arr[1][0]\*Y.arr[0][1] + X.arr[1][1]\*Y.arr[1][1];

return Z;

}

Matrix<T> A(n/2), B(n/2), C(n/2), D(n/2), E(n/2), F(n/2), G(n/2), H(n/2);

A.assign(X(0,n/2,0,n/2)); B.assign(X(0,n/2,n/2,n));

C.assign(X(n/2,n,0,n/2)); D.assign(X(n/2,n,n/2,n));

E.assign(Y(0,n/2,0,n/2)); F.assign(Y(0,n/2,n/2,n));

G.assign(Y(n/2,n,0,n/2)); H.assign(Y(n/2,n,n/2,n));

auto p1 = matmul(move(A), F-H), p2 = matmul(A+B, move(H)), p3 = matmul(C+D, move(E)), p4 = matmul(move(D), G-E);

auto p5 = matmul(A+D, E+H), p6 = matmul(B-D, G+H), p7 = matmul(A-C, E+F);

Z.assign(p5+p4-p2+p6, 0, n/2, 0, n/2);

Z.assign(p1+p2, 0, n/2, n/2, n);

Z.assign(p3+p4, n/2, n, 0, n/2);

Z.assign(p1+p5-p3-p7, n/2, n, n/2, n);

return Z;

}

int main()

{

int n;

cin >> n;

Matrix<int> m1 (n, n, 0);

for(int i = 0; i < n\*n; i++)

cin >> m1.arr[i/n][i%n];

Matrix<int> m2 (n, n, 0);

for(int i = 0; i < n\*n; i++)

cin >> m2.arr[i/n][i%n];

int t = pow(2, ceil(log2(m1.n))) - m1.n;

m1 = pad\_even(m1, t);

m2 = pad\_even(m2, t);

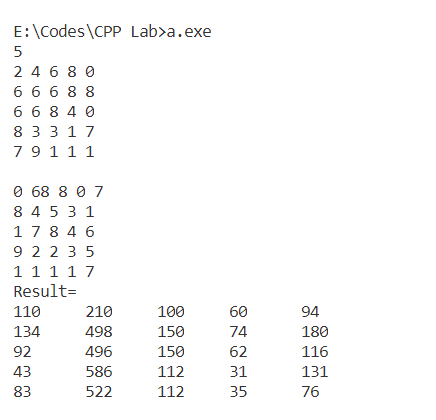
auto res = matmul(move(m1), move(m2));

res = pad\_odd(res, t);

cout << "Result=\n";res.display();

}

**Screen-Shot:**



**Time Complexity:**

**Source Code:**

#include <iostream>

#include <vector>

inline void swap(int &a, int &b)

{

    int temp = a;

    a = b;

    b = temp;

}

void heapify(std::vector<int> &heap, int pos)

{

    int large=heap[pos], i = pos;

    int left = (2\*pos)+1;

    int right = (2\*pos)+2;

    if(left<heap.size())

        if(large<heap[left])

        {

            large = heap[left];

            i = left;

        }

    if(right<heap.size())

        if(large<heap[right])

        {

            large = heap[right];

            i = right;

        }

    if(pos!=i)

    {

        swap(heap[pos], heap[i]);

        heapify(heap, i);

    }

}

int kth\_small(std::vector<int> &vect, int k)

{

    int n = vect.size();

    std::vector<int> kheap(vect.begin(), vect.begin()+(k+1));

    for(int i=(k-1)/2;i>=0;i--)

        heapify(kheap, i);

    for(int i=k+1;i<n;i++)

        if(kheap[0]>vect[i])

        {

            swap(kheap[0], vect[i]);

            heapify(kheap, 0);

        }

    return kheap[0];

}

int main()

{

    int n, k;

    std::cout<<"Enter the size of array: ";

    std::cin>>n;

    std::vector<int> arr(n);

    std::cout<<"Enter the array: ";

    for(auto &i: arr)

        std::cin>>i;

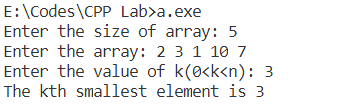
    std::cout<<"Enter the value of k(0<k<n): ";

    std::cin>>k;

    std::cout<<"The kth smallest element is "<<kth\_small(arr, k-1)<<std::endl;

}

**Screen-shot:**



**Time Complexity:**

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

void qsrt(int left, int right, int \*\*arr, int n, int m)

{

    if(left>=right-2)

        return;

    int pivot = arr[(left+1)/m][(left+1)%m], i=0, j, temp, end = right,                                             start=left;

    while(left!=right-1)

    {

        if(i%2 == 0)

        {

            if(pivot>=arr[(left+1)/m][(left+1)%m])

                left++;

            else

            {

                temp = arr[(left+1)/m][(left+1)%m];

                arr[(left+1)/m][(left+1)%m] = arr[(right-                                           1)/m][(right-1)%m];

                arr[(right-1)/m][(right-1)%m] = temp;

                right--;

            }

        }

        else

        {

            if(pivot<=arr[(right-1)/m][(right-1)%m])

                right--;

            else

            {

                temp = arr[(left+1)/m][(left+1)%m];

                arr[(left+1)/m][(left+1)%m] = arr[(right-                                           1)/m][(right-1)%m];

                arr[(right-1)/m][(right-1)%m] = temp;

                left++;

            }

        }

        i++;

    }

    if(left!=start)

    {

        arr[(start+1)/m][(start+1)%m] = arr[left/m][left%m];

        arr[left/m][left%m] = pivot;

    }

    qsrt(start, left, arr, n, m);

    qsrt(left, end, arr, n, m);

}

int main()

{

    int \*\*arr, i, j, n, m;

    printf("Enter the row and column size: ");

    scanf("%d%d",&n, &m);

    arr = (int \*\*)malloc(n\*sizeof(int \*));

    for(i=0;i<n;i++)

        arr[i] = (int \*)malloc(m\*sizeof(int));

    printf("Enter the 2D array:\n");

    for(i=0;i<n;i++)

        for(j=0;j<m;j++)

            scanf("%d", &arr[i][j]);

    qsrt(-1, n\*m, arr, n, m);

    printf("The sorted 2D array is:\n");

    for(i=0;i<n;i++)

    {

        for(j=0;j<m;j++)

            printf("\t%d", arr[i][j]);

        printf("\n");

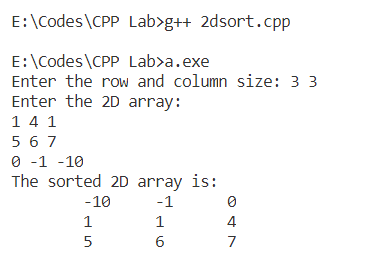
    }

    printf("\n");

    return 0;

}

**Screen-shot:**



**Time Complexity:**

**Source Code:**

#include <iostream>

#include <vector>

#include <tuple>

inline void upper\_case(std::tuple<std::string,std::string> &str)

{

    int size = std::get<1>(str).size();

    for(int i=0;i<size;i++)

        std::get<0>(str).push\_back(tolower(std::get<1>(str)[i]));

}

void radixsort(std::vector<std::tuple<std::string,std::string>> &list)

{

    int len = std::get<0>(list[0]).size(), n = list.size();

    std::vector<std::tuple<std::string,std::string>> temp(n);

    for(int i=len-1;i>=0;i--)

    {

        int count[26]={0};

        for(auto &j: list)

            count[std::get<0>(j)[i]-'a']++;

        for(int j=1;j<26;j++)

            count[j]+=count[j-1];

        for(int j=n-1;j>=0;j--)

        {

            temp[count[std::get<0>(list[j])[i]-'a']-1] = list[j];

            count[std::get<0>(list[j])[i]-'a']--;

        }

        for(int j=0;j<n;j++)

            list[j]=temp[j];

    }

}

int main()

{

    int n, i;

    std::cout<<"\nEnter the size of list: ";

    std::cin>>n;

    std::vector<std::tuple<std::string, std::string>> list(n);

    std::cout<<"Enter the list of strings(of same length):\n";

    for(auto &i: list)

    {

        std::cin>>std::get<1>(i);

        upper\_case(i);

    }

    radixsort(list);

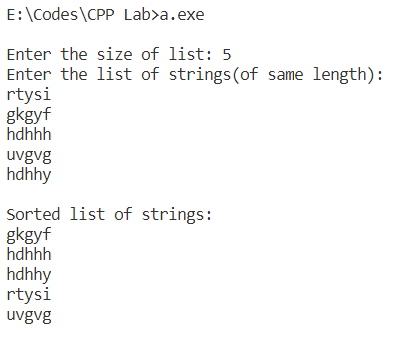
    std::cout<<"\nSorted list of strings:\n";

    for(auto &i: list)

        std::cout<<std::get<1>(i)<<std::endl;

}

**Screen-shot:**



**Time Complexity:**