Assignment 2

**Question 1:** Write a program to implement radix sort

**Algorithm:**

Radix-Sort(A, d):

//It works same as counting sort for d number of passes.

//Each key in A[1..n] is a d-digit integer.

//(Digits are numbered 1 to d from right to left.)

for j = 1 to d do

//A[]-- Initial Array to Sort

int count[10] = {0};

//Store the count of "keys" in count[]

//key- it is number at digit place j

for i = 0 to n do

count[key of(A[i]) in pass j]++

for k = 1 to 10 do

count[k] = count[k] + count[k-1]

//Build the resulting array by checking

//new position of A[i] from count[k]

for i = n-1 downto 0 do

result[ count[key of(A[i])] ] = A[j]

count[key of(A[i])]--

//Now main array A[] contains sorted numbers

//according to current digit place

for i=0 to n do

A[i] = result[i]

end for(j)

end func

**Code:**

#include <iostream>

#include <vector>

void countSort(std::vector<int> &arr, int n, int exp)

{

int output[n];

int i, count[10] = {0};

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

count[ (arr[i]/exp)%10 ]++;

for (i = 1; i < 10; i++)

count[i] += count[i - 1];

for (i = n - 1; i >= 0; i--)

{

output[count[ (arr[i]/exp)%10 ] - 1] = arr[i];

count[ (arr[i]/exp)%10 ]--;

}

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

arr[i] = output[i];

}

void radixsort(std::vector<int> &arr, int n)

{

int m = arr[0];

for(int i = 1; i < n; i++)

if(arr[i] > m)

m = arr[i];

for (int exp = 1; m/exp > 0; exp \*= 10)

countSort(arr, n, exp);

}

int main()

{

int n; std::cin>>n;

std::vector<int> arr(n);

for(auto& i : arr)

std::cin>>i;

radixsort(arr, n);

for(const auto& i : arr)

std::cout<<i<<' ';

std::cout<<'\n';

return 0;

}

**Input:** 7

1 7 2 6 3 5 4

**Output:** 1 2 3 4 5 6 7

**Question 2:** Apply Merge Sort to count the number of inversion pairs in an array.

**Algorithm:**

Start

Algorithm for method main():

Start

Step 1: Accept an integer and store it in an integer variable n.

Step 2: Initialise an integer array arr of size n and take elements in arr.

Step 3: Display the result returned from calling subroutine count() with arr, arr, 0, n-1 as parameters

Stop

Algorithm for subroutine count():

Start

Step 1: Accept an integer array reference arr, an array temp and two integer variables left and right as parameters.

Step 2: Declare an integer variable mid and initialise an integer variable inv\_count with 0

Step 3: if right > left

1. mid <-- (right + left ) / 2
2. inv\_count <-- count(arr, temp, left, mid)
3. inv\_count -- inv\_count + count(arr, temp, mid+1, right)
4. inv\_count <-- inv\_count + merge(arr, temp, mid+1, right)

Step 4: Return inv\_count

Stop

Algorithm for subroutine merge():

Start

Step 1: Accept an integer array reference arr, an array temp, and three integer variables left, mid, right as parameters

Step 2: Initialise integer variables i, j, k and inv\_count with left, mid, left, 0

Step 3:

1. if arr[i] <= arr[j]
   1. temp[k++] <-- arr[j++]
2. else
   1. temp[k++] <-- arr[j++]
   2. inv\_count <-- inv\_count + mid – i

Step 4: while i <= mid – 1

temp[k++] <-- arr[i++]

Step 5: while j <= right

temp[k++] <-- arr[j++]

Step 6: for i = left to right

1. arr[i] <-- temp[i]

Step 7: return inv\_count

Stop

Stop

**Code:**

#include <iostream>

#include <vector>

int merge(std::vector<int> &arr, std::vector<int> temp, int left, int mid, int right)

{

int i, j, k;

int inv\_count = 0;

i = left;

j = mid;

k = left;

while ((i <= mid - 1) && (j <= right))

{

if (arr[i] <= arr[j])

{

temp[k++] = arr[i++];

}

else

{

temp[k++] = arr[j++];

inv\_count = inv\_count + (mid - i);

}

}

while (i <= mid - 1)

temp[k++] = arr[i++];

while (j <= right)

temp[k++] = arr[j++];

for (i=left; i <= right; i++)

arr[i] = temp[i];

return inv\_count;

}

int count(std::vector<int> &arr, std::vector<int> temp, int left, int right)

{

int mid, inv\_count = 0;

if (right > left)

{

mid = (right + left)/2;

inv\_count = count(arr, temp, left, mid);

inv\_count += count(arr, temp, mid+1, right);

inv\_count += merge(arr, temp, left, mid+1, right);

}

return inv\_count;

}

int main()

{

int n; std::cin>>n;

std::vector<int> arr(n);

for(auto& i : arr)

std::cin>>i;

std::cout<<count(arr, arr, 0, n-1)<<'\n';

return 0;

}

**Input:** 6

1 6 2 5 3 4

**Output:** 6

**Question 3:** Find the kth element of the combined array of two sorted arrays

**Algorithm:**

Start

Algorithm for method main():

Start

Step 1: Accept an integer and store it in an integer variable m

Step 2: Declare an integer array arr1 of size m an take input in arr1

Step 3: Accept an integer and store it in an integer variable n

Step 4: Declare an integer array arr2 of size n an take input in arr2

Step 5: Accept an integer and store it in an integer k

Step 6: Call subroutine kth with arr1, m, arr2, n, k as parameters and store the returned result in an integer variable ans

Step 7: Print the value of ans

Stop

Algorithm for subroutine kth(arr1, m, arr2, n, k):

Start

Step 1: if k > (m+n) || k < 1

return nothing;

Step 2: if (m > n)

return kth(arr2, n, arr1, m, k);

Step 3: if (m == 0)

return arr2[k - 1];

Step 4: if (k == 1)

return MIN(arr1[0], arr2[0]);

Step 5: i = MIN(m, k / 2)

Step 6: j = MIN(n, k / 2);

Step 7: if (arr1[i - 1] > arr2[j - 1] )

return kth(arr1, m, arr2 + j, n - j, k – j);

else

return kth(arr1 + i, m - i, arr2, n, k - i);

Stop

Stop

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

#include <optional>

std::optional<int> kth(int \*arr1, int m, int \*arr2, int n, int k)

{

if (k > (m+n) || k < 1)

return {};

if (m > n)

return kth(arr2, n, arr1, m, k);

if (m == 0)

return arr2[k - 1];

if (k == 1)

return std::min(arr1[0], arr2[0]);

int i = std::min(m, k / 2), j = std::min(n, k / 2);

if (arr1[i - 1] > arr2[j - 1] )

return kth(arr1, m, arr2 + j, n - j, k - j);

else

return kth(arr1 + i, m - i, arr2, n, k - i);

}

int main()

{

int m, n, k;

std::cin>>m;

std::vector<int> arr1(m);

for(auto& i : arr1)

std::cin>>i;

std::cin>>n;

std::vector<int> arr2(n);

for(auto& i : arr2)

std::cin>>i;

std::cin>>k;

std::optional<int> ans = kth(arr1.data(),m,arr2.data(), n, k);

if(!ans.has\_value())

std::cout<<"Invalid query\n";

else

std::cout<<ans.value()<<'\n';

return 0;

}

**Input:** 3

1 6 9

4

2 3 5 7

5

**Output:** 6

**Question 4:** Find neighbors of the median element in an array using partitioning strategy of QuickSorting method.

**Algorithm:**

Start

Algorithm for method main():

Start

Step 1: Accept an integer and store it in an integer variable n.

Step 2: Declare an integer array arr of size n and accept input in it

Step 3: Accept an integer and store it in an integer k

Step 4: Display the result obtained on calling subroutine kthSmallest( arr, 0, n-1, k )

Stop

Algorithm for subroutine kthSmallest( arr, l, r, k ):

Start

Step 1: if(k > 0 && k <= r - l + 1)

int pos <-- partition(arr, l, r);

if (pos-l = k-1)

return arr[pos];

if (pos-l > k-1)

return kthSmallest(arr, l, pos-1, k);

return kthSmallest(arr, pos+1, r, k-pos+l-1);

Step 2: return ∞;

Stop

Algorithm for subroutine partition(arr, l, r):

Start

Step 1: x <-- arr[r], i <-- l;

for j = l to r - 1

if arr[j] <= x

std::swap(arr[i], arr[j]);

i++;

Step 2: swap(arr[i], arr[r]);

Step 3: return i;

Stop

Stop

**Code:**

#include <iostream>

#include <climits>

#include <vector>

#include <algorithm>

int partition(std::vector<int> &arr, int l, int r)

{

int x = arr[r], i = l;

for (int j = l; j <= r - 1; j++)

{

if(arr[j] <= x)

{

std::swap(arr[i], arr[j]);

i++;

}

}

std::swap(arr[i], arr[r]);

return i;

}

int kthSmallest(std::vector<int> &arr, int l, int r, int k)

{

if(k > 0 && k <= r - l + 1)

{

int pos = partition(arr, l, r);

if (pos-l == k-1)

return arr[pos];

if (pos-l > k-1)

return kthSmallest(arr, l, pos-1, k);

return kthSmallest(arr, pos+1, r, k-pos+l-1);

}

return INT\_MAX;

}

int main()

{

int n, k; std::cin>>n;

std::vector<int> arr(n);

for(auto& i : arr)

std::cin>>i;

std::cin>>k;

std::cout << "K'th smallest element is " << kthSmallest(arr, 0, n-1, k)<<'\n';

return 0;

}

**Input:** 7

1 6 2 7 3 5 2 4

**Output:** K'th smallest element is 3

**Question 5:** Apply min-heap building strategy to find k-th smallest element in an array.

**Algorithm:**

Start

Step 1: Accept an integer and store it in an integer variable n.

Step 2: Declare an integer array arr of size n and accept elements in it

Step 3: Accept an integer and store it in an integer variable k.

Step 4: Construct a min heap from arr[0] to arr[k-1]

Step 5: for i = k to n-1

1. if arr[i] > arr[0]
   1. arr[0] <-- arr[i]
   2. Heapify from arr[0] to arr[k-1]

Step 6: Display the value of arr[0]

Stop

**Code:**

#include <iostream>

#include <algorithm>

#include <vector>

int main()

{

int n; std::cin>>n;

std::vector<int> arr(n);

for(auto& i : arr)

std::cin>>i;

int k; std::cin>>k;

std::make\_heap(arr.begin(), arr.begin()+k, [](const int &x, const int &y){return x>y;});

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

if(arr[i]>arr[0])

{

arr[0] = arr[i];

std::make\_heap(arr.begin(), arr.begin()+k, [](const int &x, const int &y){return x>y;});

}

std::cout<<arr[0]<<'\n';

}

**Input:** 5

1 5 2 4 3

3

**Output:** 3

**Question 6:** Apply quick sort on a 2D NxM matrix of numbers so that the numbers are sorted in rowmajor fashion in the 2D matrix (without auxiliary array).

**Algorithm:**

Start

Step 1: Accept two integers and store them in integer variables m and n.

Step 2: Create a vector of vectors arr of size m \* n

Step 3: Run a loop from i = 0 to m-1 and pass the reference of arr[i] to the quicksort subroutine (shown earlier). The reference ensures that no memory is wasted in creating an extra copy of the row.

Step 4: Display the sorted 2D array arr

Stop

**Code:**

#include <iostream>

#include <vector>

#include <iterator>

#include <algorithm>

template<typename T, typename Comp>

int partition(std::vector<T> &arr, int low, int high, Comp cmp)

{

auto [pivot, i] = std::make\_pair(arr[high], low-1);

for(int j = low; j < high; j++)

if(cmp(arr[j], pivot))

{

i++;

std::swap(arr[i], arr[j]);

}

std::swap(arr[i+1], arr[high]);

return i+1;

}

template<typename T, typename Comp>

void sort(std::vector<T> &arr, int low, int high, Comp cmp)

{

if(low < high)

{

int x = partition(arr, low, high, cmp);

sort(arr, low, x-1, cmp);

sort(arr, x+1, high, cmp);

}

}

int main()

{

int m, n; std::cin>>m>>n;

std::vector<std::vector<int>> arr(m, std::vector<int>(n));

for(auto& i : arr)

for(auto& j : i)

std::cin>>j;

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

sort(arr[i], 0, arr[i].size()-1, std::less<int>());

std::cout<<"The sorted 2D array is:\n";

for(auto& i : arr)

{

for(auto& j : i)

std::cout<<j<<' ';

std::cout<<'\n';

}

}

**Input:** 3 4

1 8 3 6

2 6 -1 7

0 3 1 7

**Output:** The sorted 2D array is:

1 3 6 8

-1 2 6 7

0 1 3 7

**Question 7:** Arrange a list of words (each of equal length) using dictionary sorting strategy.

**Algorithm:**

Start

Algorithm for method main():

Start

Step 1: Accept an integer and store it in an integer variable n

Step 2: Create a vector of strings data of length n and accept elements in it

Step 3: Call subroutine radixSort( arr, n )

Step 4: Print array arr

Stop

Algorithm for subroutine radixSort( b, r ):

Start

Step 1: Find the length of the largest string in b and store it in an integer max

Step 2: Run a loop for digit = max downto 0

countSort( b, r, digit - 1 )

Stop

Algorithm for subroutine countSort( a, size, k ):

Start

Step 1: b [ size ], c[ 257 ]

Step 2: Initialise c with 0

Step 3: for j = 0 to size – 1

1. if k < a[j].size()
   1. c[ a[ j ][ k ] + 1 ] <-- c[ a[ j ][ k ] + 1 ] + 1
2. else
   1. c[ 0 ] <-- c[ 0 ] + 1

Step 4: for f = 1 to 256

1. c[ f ] <-- c[ f ] + c[ f - 1 ]

Step 5: for r = size – 1 downto 0

1. if k < a[ r ].size()
   1. b[ a[ r ][ k ] + 1 ] <-- a[ r ]
   2. c[ a[ j ][ k ] + 1 ] <-- c[ a[ j ][ k ] + 1 ] - 1

Step 6: for l = 0 to size – 1

1. a[ l ] <-- b[ l ]

Stop

Stop

**Code:**

#include <string>

#include <vector>

#include <iostream>

size\_t getMax(std::vector<std::string> &arr, int n)

{

size\_t max = arr[0].size();

for (int i = 1; i < n; i++)

{

if (arr[i].size()>max)

max = arr[i].size();

}

return max;

}

void countSort(std::vector<std::string> &a, int size, size\_t k)

{

std::vector<std::string> b(size);

std::vector<int> c(257, 0);

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

c[k < a[j].size() ? (int)(unsigned char)a[j][k] + 1 : 0]++;

for (int f = 1; f <257; f++)

c[f] += c[f - 1];

for (int r = size - 1; r >= 0; r--){

b[c[k < a[r].size() ? (int)(unsigned char)a[r][k] + 1 : 0] - 1] = a[r];

c[k < a[r].size() ? (int)(unsigned char)a[r][k] + 1 : 0]--;

}

for (int l = 0; l < size; l++){

a[l] = b[l];

}

}

void radixSort(std::vector<std::string> &b, int r)

{

size\_t max = getMax(b, r);

for (size\_t digit = max; digit > 0; digit--)

countSort(b, r, digit - 1);

}

int main()

{

int n; std::cin>>n;

std::vector<std::string> data(n);

for(auto& i : data)

std::cin>>i;

radixSort(data, n);

for(auto& i : data)

std::cout<<i<<'\n';

return 0;

}

**Input:** 5

abc ram sam jodu modhu

**Output:** abc

jodu

modhu

ram

sam

**Question 8:** Implement Strassen’s Matrix multiplication strategy

**Algorithm:**

Start

Algorithm for method main():

Start

Step 1: Accept an integer and store it in an integer variable n

Step 2: Declare two matrices mtx1 and mtx2 of order n \* n

Step 3: Input elements in mtx1 and mtx2.

Step 4: Find the product of mtx1 and mtx2 and store it in res

Step 5: Display the matrix res

Stop

Algorithm for subroutine product( X, Y ):

Start

Step 1: n <-- X.size()

Step 2: if n = 2

1. bound( 2, 2 )
2. bound[0][0] <-- (X[0][0] \* Y[0][0]) + (X[0][1] \* Ydata[1][0]);
3. bound[0][1] <-- (X[0][0] \* Y[0][1]) + (X[0][1] \* Ydata[1][1]);
4. bound[1][0] <-- (X[1][0] \* Y[0][0]) + (X[1][1] \* Ydata[1][0]);
5. bound[1][1] <-- (X[1][0] \* Y[0][1]) + (X[1][1] \* Ydata[1][1]);
6. return bound

Step 3: if n % 2 = 1

1. for i = 0 to X.size() - 1
   1. X[i].insert(0)
   2. Y[i].insert(0)
2. X.insert( { 0, 0, 0 ... X.size() elements } )
3. Y.insert( { 0, 0, 0 ... Y.size() elements } )
4. Y[Y.size() - 1 ][ Y.size() - 1 ] <-- 1
5. X[X.size() - 1 ][ X.size() - 1 ] <-- 1

Step 4: [ A, B, C, D ] <-- slice(X)

Step 5: [ E, F, G, H ] <-- slice(Y)

Step 6: P1 <-- A \* ( F - H )

Step 7: P2 <-- (A + B) \* H

Step 8: P3 <-- (C + D) \* E

Step 9: P4 <-- D \* (G – E)

Step 10: P5 <-- (A + D) \* (E + H)

Step 11: P6 <-- (B – D) \* (G + H)

Step 12: P7 <-- (A – C) \* (E + F)

Step 13: temp <-- merge( P6 + P5 + P4 – P2, P1 + P2, P3 + P4, P1 + P5 – (P3 + P7) )

Step 14: if n % 2 = 1

1. temp.pop()
2. for i = 0 to temp.size() - 1
   1. temp[i].pop()
3. return temp

Stop

Algorithm for subroutine slice(mat):

Start

Step 1: n <-- mat.size()

Step 2: ans1(n/2, n/2), ans2(n/2, n/2), ans3(n/2, n/2), ans4(n/2, n/2)

Step 3: for i = 0 to n/2 – 1

for j = 0 to n/2 – 1

ans1[i][j] <-- mat[i][j]

Step 4: for x = 0, i = 0 to n/2 – 1, x++

for y = 0, j = n/2 to n – 1, y++

ans2[x][y] <-- mat[i][j]

Step 5: for x = 0, i = n/2 to n – 1, x++

for y = 0, j = 0 to n/2 – 1, y++

ans3[x][y] <-- mat[i][j]

Step 6: for x = 0, i = n/2 to n – 1, x++

for y = 0, j = n/2 to n – 1, y++

ans4[x][y] <-- mat[i][j]

Step 7: return ans1, ans2, ans3, ans4

Stop

Stop

**Code:**

#include <iostream>

#include <vector>

#include <tuple>

template<typename T>

struct Mat

{

std::vector<std::vector<T>> data;

Mat(int rows, int cols)

{

data = std::vector<std::vector<T>>(rows, std::vector<T>(cols));

}

Mat(int rows, int cols, int val)

{

data = std::vector<std::vector<T>>(rows, std::vector<T>(cols, val));

}

Mat<T> merge(const Mat<T> &a, const Mat<T> &b, const Mat<T> &c, const Mat<T> &d)

{

int n = a.data.size()\*2;

Mat<T> res(n, n, T(0));

for(int i = 0, x = 0; i < n/2; i++, x++)

for(int j = 0, y = 0; j < n/2; j++, y++)

res.data[i][j] = a.data[x][y];

for(int i = 0, x = 0; i < n/2; i++, x++)

for(int j = n/2, y = 0; j < n; j++, y++)

res.data[i][j] = b.data[x][y];

for(int i = n/2, x = 0; i < n; i++, x++)

for(int j = 0, y = 0; j < n/2; j++, y++)

res.data[i][j] = c.data[x][y];

for(int i = n/2, x = 0; i < n; i++, x++)

for(int j = n/2, y = 0; j < n; j++, y++)

res.data[i][j] = d.data[x][y];

return res;

}

std::tuple<Mat<T>, Mat<T>, Mat<T>, Mat<T>> slice(const Mat<T> &mat)

{

int n = mat.data.size();

Mat<T> ans1(n/2, n/2);

Mat<T> ans2(n/2, n/2);

Mat<T> ans3(n/2, n/2);

Mat<T> ans4(n/2, n/2);

for(int i = 0; i < n/2; i++)

for(int j = 0; j < n/2; j++)

ans1.data[i][j] = mat.data[i][j];

for(int i = 0, x = 0; i < n/2; i++, x++)

for(int j = n/2, y = 0; j < n; j++, y++)

ans2.data[x][y] = mat.data[i][j];

for(int i = n/2, x = 0; i < n; i++, x++)

for(int j = 0, y = 0; j < n/2; j++, y++)

ans3.data[x][y] = mat.data[i][j];

for(int i = n/2, x = 0; i < n; i++, x++)

for(int j = n/2, y = 0; j < n; j++, y++)

ans4.data[x][y] = mat.data[i][j];

return std::make\_tuple(ans1, ans2, ans3, ans4);

}

Mat<T> operator-(const Mat<T>& b)

{

Mat<T> c(b.data.size(), b.data.size());

for(int i=0;i<b.data.size();i++)

for(int j=0;j<b.data.size();j++)

c.data[i][j]=data[i][j]-b.data[i][j];

return c;

}

Mat<T> operator+(const Mat<T>& b)

{

Mat<T> c(b.data.size(), b.data.size());

for(int i=0;i<b.data.size();i++)

for(int j=0;j<b.data.size();j++)

c.data[i][j]=data[i][j]+b.data[i][j];

return c;

}

Mat<T> operator\*(Mat<T> Y)

{

Mat<T> X = \*this;

int n = X.data.size();

if(n == 2)

{

Mat<T> bound(2, 2);

bound.data[0][0] = (X.data[0][0] \* Y.data[0][0]) + (X.data[0][1] \* Y.data[1][0]);

bound.data[0][1] = (X.data[0][0] \* Y.data[0][1]) + (X.data[0][1] \* Y.data[1][1]);

bound.data[1][0] = (X.data[1][0] \* Y.data[0][0]) + (X.data[1][1] \* Y.data[1][0]);

bound.data[1][1] = (X.data[1][0] \* Y.data[0][1]) + (X.data[1][1] \* Y.data[1][1]);

return bound;

}

if(n % 2 == 1)

{

for(int i=0;i<X.data.size();i++)

{

X.data[i].push\_back(T(0));

Y.data[i].push\_back(T(0));

}

X.data.push\_back(std::vector<T>(X.data.size(),T(0)));

Y.data.push\_back(std::vector<T>(Y.data.size(),T(0)));

Y.data[Y.data.size()-1][Y.data.size()-1]=T(1);

X.data[X.data.size()-1][X.data.size()-1]=T(1);

}

auto [A, B, C, D] = slice(X);

auto [E, F, G, H] = slice(Y);

auto P1 = A \* (F-H);

auto P2 = (A+B) \* H;

auto P3 = (C+D) \* E;

auto P4 = D \* (G-E);

auto P5 = (A+D) \* (E+H);

auto P6 = (B-D) \* (G+H);

auto P7 = (A-C) \* (E+F);

auto temp = merge((P6+P5)+(P4-P2), P1+P2, P3+P4, (P1+P5)-(P3+P7));

if(n%2==1)

{

temp.data.pop\_back();

for(auto& i: temp.data)

i.pop\_back();

}

return temp;

}

};

int main()

{

int n; std::cout<<"Enter n\n";

std::cin>>n;

Mat<int> mtx1(n, n), mtx2(n, n);

std::cout<<"Enter the 1st matrix:\n";

for(auto& i : mtx1.data)

for(auto& j : i)

std::cin>>j;

std::cout<<"Enter the 2nd matrix:\n";

for(auto& i : mtx2.data)

for(auto& j : i)

std::cin>>j;

std::cout<<"The resultant matrix:\n";

auto res = mtx1 \* mtx2;

for(auto& i : res.data)

{

for(auto& j : i)

std::cout<<j<<", ";

std::cout<<"\n";

}

std::cout<<std::endl;

return 0;

}

**Input/Output:**

Enter n

3

Enter the 1st matrix:

1 5 3

2 7 4

1 2 4

Enter the 2nd matrix:

9 3 7

2 8 4

1 0 9

The resultant matrix:

22, 43, 54,

36, 62, 78,

17, 19, 51,