**Institute of Engineering & Management**

**Department of Computer Science & Engineering**

**Design & Analysis of Algorithm Lab for 3rd year 5th semester 2018**

**Code: CS 591**

**Date:** 01/08/18

**WEEK-2**

**Source code:**

#include <stdio.h>

void radixsort(int arr[], int n){

int max=arr[0], i, j, temp[n];

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

if(max<arr[i])

max = arr[i];

for(i=1;max/i>0;i\*=10){

int count[10]={0};

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

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

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

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

for(j=n-1;j>=0;j--){

temp[count[(arr[j]/i)%10]-1]=arr[j];

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

}

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

arr[j]=temp[j];

}

}

int main(){

int n, i;

printf("Enter the size of array: ");

scanf("%d", &n);

int arr[n];

printf("Enter the array: ");

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

scanf("%d",&arr[i]);

radixsort(arr, n);

printf("The sorted array: ");

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

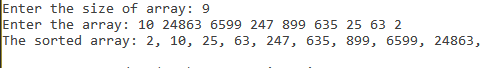
printf("%d, ",arr[i]);

printf("\n");

return 0;

}

**Screen-Shot:**

****

**Time Complexity:**

**Source code:**

#include <stdio.h>

#include <stdlib.h>

int inver\_count(int \*arr, int low, int high){

int mid=(low+high)/2, i, lp=low, rp=(low+high)/2, count\_inver=0, temp[high-low];

if(low>=high-1)

return 0;

else{

count\_inver+=inver\_count(arr, low, mid);

count\_inver+=inver\_count(arr, mid, high);

for(i=0;i<high-low;i++){

if(lp==mid)

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

else if(rp==high)

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

else if(arr[lp]>arr[rp]){

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

count\_inver+=mid-lp;

}

else if(arr[lp]<=arr[rp])

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

}

for(i=0;i<high-low;i++)

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

return count\_inver;

}

}

int main(){

int n, i, \*arr;

printf("Enter the size of array: ");

scanf("%d", &n);

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

printf("Enter the array: ");

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

scanf("%d", &arr[i]);

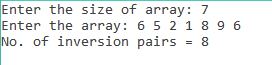
printf("No. of inversion pairs = %d\n", inver\_count(arr, 0, n));

free(arr);

return 0;

}

**Screen-Shot:**

****

**Time Complexity:**

**Source code:**

#include <stdio.h>

int kth(int \*arr1, int \*arr2, int \*end1, int \*end2, int k){

if (arr1 == end1)

return arr2[k];

if (arr2 == end2)

return arr1[k];

int mid1 = (end1 - arr1) / 2;

int mid2 = (end2 - arr2) / 2;

if (mid1 + mid2 < k){

if (arr1[mid1] > arr2[mid2])

return kth(arr1, arr2 + mid2 + 1, end1, end2, k-mid2-1);

else return kth(arr1 + mid1 + 1, arr2, end1, end2, k-mid1-1);

}

else{

if (arr1[mid1] > arr2[mid2])

return kth(arr1, arr2, arr1 + mid1, end2, k);

else return kth(arr1, arr2, end1, arr2 + mid2, k);

}

}

int main(){

int n, m, k, i;

printf("Enter the size of 1st & 2nd array: ");

scanf("%d%d", &n, &m);

int arr1[n], arr2[m];

printf("Enter the 1st array: ");

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

scanf("%d",&arr1[i]);

printf("Enter the 2nd array: ");

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

scanf("%d",&arr2[i]);

printf("Enter the postion: ");

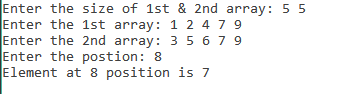
scanf("%d", &k);

printf("Element at %d position is %d\n", kth(arr1, arr2, arr1+n, arr2+m, k-1));

return 0;

}

**Screen-Shot:**

****

**Time Complexity:**

**Source code:**

#include <stdio.h>

int kth(int n, int \*arr, int pos)

{

int left = -1, right = n, pivot = arr[0], i=0, temp, res=0;

if(n<=1)

return arr[0];

while(left!=right-1)

{

if(i%2 == 0)

{

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

left++;

else{

temp = arr[left+1];

arr[left+1] = arr[right-1];

arr[right-1] = temp;

right--;

}

}

else{

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

right--;

else{

temp = arr[left+1];

arr[left+1] = arr[right-1];

arr[right-1] = temp;

left++;

}

}

i++;

}

if(left!=-1)

{

arr[0] = arr[left];

arr[left] = pivot;

}

if(left>pos)

res=kth(left, arr, pos);

else if(left<pos)

res=kth(n-right, &arr[right], pos-left-1);

else return arr[left];

return res;

}

int main()

{

int n, i;

printf("Enter the size(>1) of array: ");

scanf("%d", &n);

int arr[n];

printf("Enter the array: ");

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

scanf("%d",&arr[i]);

printf("The neighbor elements are %d & %d\n", kth(n, arr, ((n/2)-1)), kth(n, arr, ((n+1)/2)));

return 0;

}

**Screen-Shot:**

****

**Time Complexity:**

**Source code:**

#include <iostream>

#include <vector>

#include <tuple>

typedef std::vector<std::vector<int>> data;

data merge(const data &a, const data &b, const data &c, const data &d)

{

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

data res(n, std::vector<int>(n,0));

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

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

res[i][j] = a[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[i][j] = b[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[i][j] = c[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[i][j] = d[x][y];

return res;

}

std::tuple<data, data, data, data> slice(const std::vector<std::vector<int>> &mat)

{

int n = mat.size();

std::vector<std::vector<int>> ans1(n/2, std::vector<int>(n/2));

std::vector<std::vector<int>> ans2(n/2, std::vector<int>(n/2));

std::vector<std::vector<int>> ans3(n/2, std::vector<int>(n/2));

std::vector<std::vector<int>> ans4(n/2, std::vector<int>(n/2));

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

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

ans1[i][j] = mat[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[x][y] = mat[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[x][y] = mat[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[x][y] = mat[i][j];

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

}

data operator-(const data& a, const data& b)

{

std::vector<std::vector<int>> c(a.size(), std::vector<int>(a.size()));

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

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

c[i][j]=a[i][j]-b[i][j];

return c;

}

data operator+(const data& a, const data& b)

{

std::vector<std::vector<int>> c(a.size(), std::vector<int>(a.size()));

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

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

c[i][j]=a[i][j]+b[i][j];

return c;

}

data product(data X, data Y)

{

int n = X.size();

if(n==2)

{

data bound(2,std::vector<int>(2));

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

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

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

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

return bound;

}

if(n%2==1)

{

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

{

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

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

}

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

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

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

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

}

data A, B, C, D, E, F, G, H;

std::tie(A, B, C, D) = slice(X);

std::tie(E, F, G, H) = slice(Y);

data P1 = product(A, F-H);

data P2 = product(A+B, H);

data P3 = product(C+D, E);

data P4 = product(D, G-E);

data P5 = product(A+D, E+H);

data P6 = product(B-D, G+H);

data P7 = product(A-C, E+F);

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

if(n%2==1)

{

temp.pop\_back();

for(auto& i: temp)

i.pop\_back();

}

return temp;

}

int main()

{

int n;

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

std::cin>>n;

data mtx1(n, std::vector<int>(n)), mtx2(n, std::vector<int>(n));

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

for(auto& i : mtx1)

for(auto& j : i)

std::cin>>j;

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

for(auto& i : mtx2)

for(auto& j : i)

std::cin>>j;

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

auto res = product(mtx1,mtx2);

for(auto& i : res)

{

for(auto& j : i)

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

std::cout<<"\n";

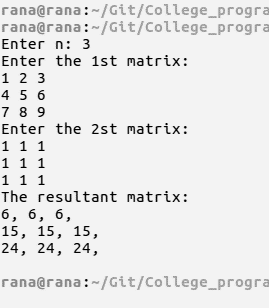
}

std::cout<<std::endl;

return 0;

}

**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;

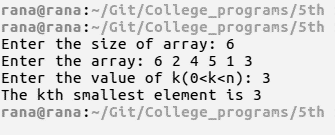
std::cout<<"Enter the value of k(0<k<n): ";

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");

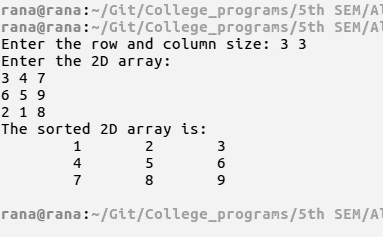
}

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);

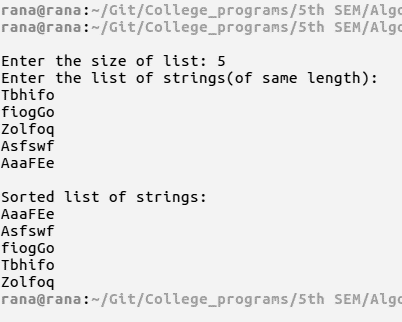
std::cout<<"\nSorted list of strings:\n";

for(auto &i: list)

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

}

**Screen-shot:**

****

**Time Complexity:**