heap\_sort.

#include<bits/stdc++.h>

using namespace std;

void scana(int a[], int n)

{

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

{

cin >> a[i];

}

}

void printa(int a[], int n)

{

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

{

cout << a[i] << " ";

}

cout << endl;

}

void heapify(int a[], int i, int n){

while(true){

int largest = i;

int l = (2 \* i) + 1;

int r = (2 \* i) + 2;

if(l < n && a[l] > a[largest]){

largest = l;

}

if(r < n && a[r] > a[largest]){

largest = r;

}

if(largest != i){

swap(a[largest], a[i]);

i = largest;

}

else{

break;

}

}

}

void heap\_sort(int a[], int n){

for(int i = (n / 2) - 1; i >= 0; i--){

heapify(a, i, n);

}

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

swap(a[0], a[i]);

heapify(a, 0, i);

}

}

int main(){

int n;

cin >> n;

int a[n];

scana(a, n);

heap\_sort(a, n);

printa(a, n);

return 0;

}

# insertion\_sort

#include<bits/stdc++.h>

using namespace std;

void scana(int a[], int n)

{

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

{

cin >> a[i];

}

}

void printa(int a[], int n)

{

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

{

cout << a[i] << " ";

}

cout << endl;

}

void insertion\_sort(int a[], int n){

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

int key = a[i];

int j = i - 1;

while(j >= 0 && a[j] > key){

a[j + 1] = a[j];

j--;

}

a[j + 1] = key;

}

}

int main()

{

int n;

cin >> n;

int a[n];

scana(a, n);

insertion\_sort(a, n);

printa(a, n);

return 0;

}

# merge\_sort

#include<bits/stdc++.h>

using namespace std;

void scana(int a[], int n)

{

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

{

cin >> a[i];

}

}

void printa(int a[], int n)

{

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

{

cout << a[i] << " ";

}

cout << endl;

}

void merge(int a[], int l, int m, int r){

int ls = m - l + 1;

int rs = r - m;

int la[ls], ra[rs];

for(int i = 0, j = l; i < ls; i++, j++){

la[i] = a[j];

}

for(int i = 0, j = m + 1; i < rs; i++, j++){

ra[i] = a[j];

}

// printa(la, ls);

// printa(ra, rs);

int i = 0, j = 0, k = l;

while(i < ls && j < rs){

if(la[i] <= ra[j]){

a[k] = la[i];

k++, i++;

}

else{

a[k] = ra[j];

k++, j++;

}

}

while(i < ls){

a[k] = la[i];

k++, i++;

}

while(j < rs){

a[k] = ra[j];

k++, j++;

}

}

void merge\_sort(int a[], int l, int r){

if(l < r){

int m = (l + r) / 2;

merge\_sort(a, l, m);

merge\_sort(a, m + 1, r);

merge(a, l, m, r);

}

}

int main()

{

int n;

cin >> n;

int a[n];

scana(a, n);

merge\_sort(a, 0, n - 1);

printa(a, n);

return 0;

}

# quick\_sort

#include<bits/stdc++.h>

using namespace std;

void scana(int a[], int n)

{

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

{

cin >> a[i];

}

}

void printa(int a[], int n)

{

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

{

cout << a[i] << " ";

}

cout << endl;

}

int partition(int a[], int l, int r){

int pivot = a[r];

int idx = l - 1;

for(int i = l; i < r; i++){

if(a[i] < pivot){

idx++;

swap(a[i], a[idx]);

}

}

idx++;

swap(a[idx], a[r]);

return idx;

}

void quick\_sort(int a[], int l, int r){

if(l < r){

int pivot\_idx = partition(a, l, r);

quick\_sort(a, l, pivot\_idx - 1);

quick\_sort(a, pivot\_idx + 1, r);

}

}

int main(){

int n;

cin >> n;

int a[n];

scana(a, n);

quick\_sort(a, 0, n - 1);

printa(a, n);

return 0;

}

**COUNTING**

#include <iostream>

using namespace std;

void countSort(int array[], int size) {

int output[10];

int count[10];

int max = array[0];

for (int i = 1; i < size; i++) {

if (array[i] > max)

max = array[i];

}

for (int i = 0; i <= max; ++i) {

count[i] = 0;

}

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

count[array[i]]++;

}

for (int i = 1; i <= max; i++) {

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

}

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

output[count[array[i]] - 1] = array[i];

count[array[i]]--;

}

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

array[i] = output[i];

}

}

void printArray(int array[], int size) {

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

cout << array[i] << " ";

cout << endl;

}

int main() {

int array[] = {4, 2, 2, 8, 3, 3, 1};

int n = sizeof(array) / sizeof(array[0]);

countSort(array, n);

printArray(array, n);

}

MCM

#include <bits/stdc++.h>

using namespace std;

int mc[50][50];

int DynamicProgramming(int\* c, int i, int j){

if (i == j) {

return 0;

}

if (mc[i][j] != -1) {

return

mc[i][j];

}

mc[i][j] = INT\_MAX;

for (int k = i; k < j; k++) {

mc[i][j] = min(mc[i][j], DynamicProgramming(c, i, k) + DynamicProgramming(c, k + 1, j) + c[i - 1] \* c[k] \* c[j]);

}

return mc[i][j];

}

int Matrix(int\* c, int n){

int i = 1, j = n - 1;

return DynamicProgramming(c, i, j);

}

int main(){

int arr[] = { 23, 26, 27, 20 };

int n = sizeof(arr) / sizeof(arr[0]);

memset(mc, -1, sizeof mc);

cout << "Minimum number of multiplications is: " << Matrix(arr, n);

}

LCS

// The longest common subsequence in C++

#include <iostream>

using namespace std;

void lcsAlgo(char \*S1, char \*S2, int m, int n) {

int LCS\_table[m + 1][n + 1];

// Building the mtrix in bottom-up way

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

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

if (i == 0 || j == 0)

LCS\_table[i][j] = 0;

else if (S1[i - 1] == S2[j - 1])

LCS\_table[i][j] = LCS\_table[i - 1][j - 1] + 1;

else

LCS\_table[i][j] = max(LCS\_table[i - 1][j], LCS\_table[i][j - 1]);

}

}

int index = LCS\_table[m][n];

char lcsAlgo[index + 1];

lcsAlgo[index] = '\0';

int i = m, j = n;

while (i > 0 && j > 0) {

if (S1[i - 1] == S2[j - 1]) {

lcsAlgo[index - 1] = S1[i - 1];

i--;

j--;

index--;

}

else if (LCS\_table[i - 1][j] > LCS\_table[i][j - 1])

i--;

else

j--;

}

// Printing the sub sequences

cout << "S1 : " << S1 << "\nS2 : " << S2 << "\nLCS: " << lcsAlgo << "\n";

}

int main() {

char S1[] = "ACADB";

char S2[] = "CBDA";

int m = strlen(S1);

int n = strlen(S2);

lcsAlgo(S1, S2, m, n);

}