**ASSESSMENT FORM**

**Course:** **COMP6049001 - Algorithm Design and Analysis**

**Method of Assessment:** **Case Study**

A logo for a university

Description automatically generated

**Name of Lecturer : MAULIN NASARI, S.T., M.Kom.**

**Date : 13 Desember 2024**

**Class : LT01**

**Topic : Review II**

**Group Name : IQ Terbang**

|  |  |
| --- | --- |
| **Group Members :** | **1. Agief Prakasa Nurdien - 2702344601**  **2. Alvin Nabil Thoriq - 2702349445**  **3. Johannes Jourdan Triadi - 2702348272** |

**Semester/Academic Year : 3/2024-2025**

1. Submit proof of your attendance at the INC 2024.

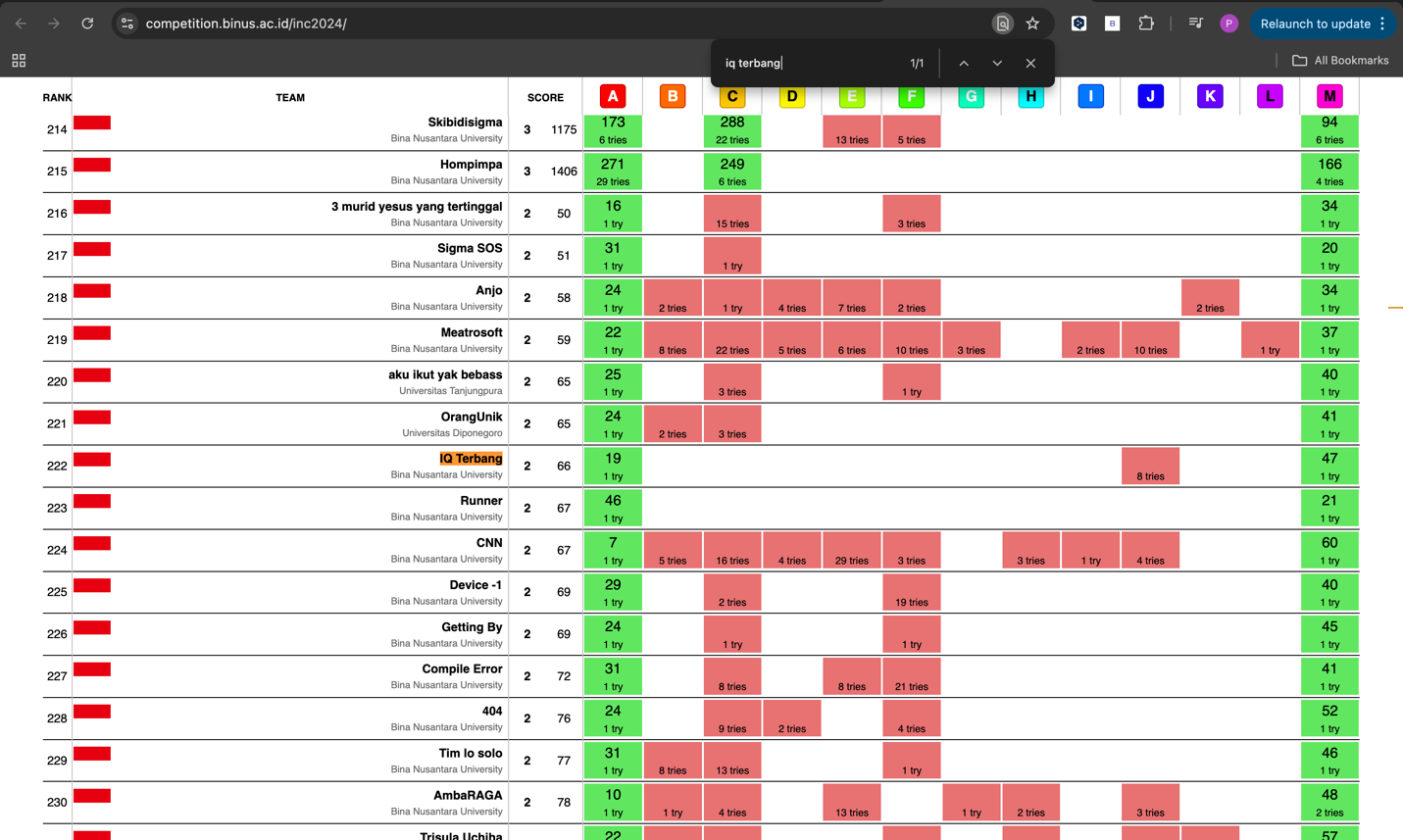
**Jawaban :**

Solved Problem :

* 1. Problem A - Problem C
  2. Problem M - ICPC Provincial

Unsolved Problem :

1. Problem J – Feng Shui

****

1. Write an analysis of the solutions used in INC 2024.

**Jawaban :**

1. *Problem A – Problem* *C*

A paper with text and images

Description automatically generated

A screenshot of a computer

Description automatically generated

Code :

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void simplify\_spelling(char\* word)

{

int length = strlen(word);

char result[101];

int j = 0;

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

{

if (word[i] == 'c') {

// Handle "ch" case

if (i < length - 1 && word[i + 1] == 'h')

{

result[j++] = 'c';

i++; // Skip the 'h'

}

// Handle 'c' followed by e, i, or y

else if (i < length - 1 && (word[i + 1] == 'e' || word[i + 1] == 'i' || word[i + 1] == 'y'))

{

result[j++] = 's';

}

// Handle 'c' followed by a, o, u or consonants

else

{

result[j++] = 'k';

}

}

else

{

result[j++] = word[i];

}

}

result[j] = '\0'; // Null-terminate the result

printf("%s\n", result);

}

int main()

{

int N;

scanf("%d", &N);

char word[101];

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

{

scanf("%s", word);

simplify\_spelling(word);

}

return 0;

}

Sample Input & Output #1 :

A black rectangle with white dots

Description automatically generated

Sample Input & Output #2 :

A black rectangle with white dots

Description automatically generated

Sample Input & Output #3 :

A black background with white text

Description automatically generated

**Analisis :**

Program diatas bertujuan untuk menyederhanakan ejaan (spelling simplification) kata-kata bahasa Inggris dengan mengubah aturan pengucapan huruf c dalam kata. Transformasi tersebut dilakukan berdasarkan aturan berikut:

1. Jika c diikuti oleh h, maka diubah menjadi c.
2. Jika c diikuti e, i, atau y, maka diubah menjadi s.
3. Jika c tidak memenuhi kedua aturan di atas, maka diubah menjadi k.
4. Selain itu, karakter lainnya tetap sama tanpa perubahan.

Program tersebut dimulai dengan membaca jumlah kata, N, menggunakan *scanf*. Kata-kata ini akan disimpan secara sementara dalam variabel *word*, sebuah array karakter berukuran maksimum 101 untuk setiap kata, termasuk null-terminator (\0). Kemudian program memproses setiap kata menggunakan loop *for* yang memanggil fungsi *simplify\_spelling* untuk menyederhanakan ejaannya satu per satu.

Pada fungsi *simplify\_spelling,* variabel *length* menyimpan hasil dari *strlen(word)* untuk mengetahui jumlah karakter yang akan diproses. Array *result* dideklarasikan untuk menyimpan versi ejaan baru dari kata. Indeks *j* diinisialisasi ke 0 sebagai penunjuk posisi pengisian dalam array *result*. Loop *for* digunakan untuk memproses setiap karakter pada kata, dengan indeks i sebagai penunjuk karakter saat ini.

* Jika karakter saat ini adalah *c*, maka dilakukan pemeriksaan :
  + Jika *c* diikuti *h*, *result[j]* diisi dengan *c*, lalu *i* dinaikkan dua kali untuk melewati h.
  + Jika *c* diikuti oleh *e*, *i*, atau *y*, *result[j]* diisi dengan *s*.
  + Jika tidak memenuhi kedua kondisi sebelumnya, *result[j]* diisi dengan *k*.
* Jika karakter saat ini bukan c, karakter langsung ditambahkan ke *result[j]*.

Setelah semua karakter diproses, array *result* diterminasi dengan *\0* untuk memastikan validitas string, lalu setiap kata hasil transformasi dicetak dengan *printf*  ke layar.

Untuk sebuah kata dengan panjang rata-rata *L*(Jumlah karakter dalam satu kata), fungsi *simplify\_spelling* memiliki time complexity *O(L)*, karena setiap karakter diproses satu kali. Sehingga dengan *N* kata, total Time Complexitynya adalah ***O(N × L)***. Kemudian untuk Space Complexitynya adalah ***O(L)*** karena array *result* berukuran maksimum *L + 1* yang digunakan untuk menyimpan hasil transformasi sementara.

1. *Problem M - ICPC Provincial*

A paper with text on it

Description automatically generated

A screenshot of a computer

Description automatically generated

Code :

#include <stdio.h>

#include <stdlib.h>

// Fungsi compare untuk qsort

int compare(const void\* a, const void\* b)

{

return (\*(int\*)a - \*(int\*)b);

}

// Fungsi untuk mengecek apakah mungkin membuat N tim dengan kekuatan minimal x

int isPossible(int\* arr, int N, int x, int total)

{

int teams = 0;

int i = 0;

// Mencoba membentuk tim dari array yang sudah disorting

while (i <= total - 3)

{

// Jika median >= x, maka tim valid

if (arr[i + 1] >= x)

{

teams++;

i += 2;

}

else

{

i++;

}

}

return teams >= N;

}

int main()

{

int N;

scanf("%d", &N);

int total = 3 \* N;

int arr[300000];

// Input array

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

{

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

}

// Sort array

qsort(arr, total, sizeof(int), compare);

// Binary search

int left = 0;

int right = 4000;

int ans = 0;

while (left <= right)

{

int mid = (left + right) / 2;

if (isPossible(arr, N, mid, total))

{

ans = mid;

left = mid + 1;

}

else

{

right = mid - 1;

}

}

printf("%d\n", ans);

return 0;

}

Sample Input & Output #1 :



Sample Input & Output #2 :



Sample Input & Output #3 :



**Analisis :**

Program diatas bertujuan untuk menyelesaikan masalah pembentukan tim dengan kekuatan minimum tertinggi. Tujuan dari program tersebut adalah menentukan nilai maksimum dari kekuatan minimal tim terlemah yang dapat dibentuk dari *3N* siswa, di mana setiap tim terdiri dari tiga anggota. Median kemampuan dari setiap tim menjadi indikator kekuatan tim tersebut.

Pertama-tama program membaca input *N* (jumlah tim) dan *3N* kemampuan siswa yang dimasukkan ke dalam array *arr*. Setelah itu, array diurutkan menggunakan fungsi *qsort* untuk memastikan bahwa data terstruktur, sehingga tim dapat dibentuk secara efisien dengan memanfaatkan elemen median. Langkah berikutnya adalah melakukan binary search untuk mencari nilai median maksimum (*x*) yang memungkinkan pembentukan *N* tim valid. Untuk memastikan apakah *N* tim valid dapat dibentuk dengan median minimal *x*, dan disinilah fungsi *isPossible* digunakan. Fungsi ini bekerja dengan memeriksa array yang sudah diurutkan, mencoba membentuk tim dari elemen-elemen yang ada. Median dari tiga elemen yang berurutan ditentukan sebagai elemen kedua (*arr[i+1]*). Jika elemen tersebut memenuhi *x*, tim dianggap valid, dan pointer *i* maju dua langkah untuk melanjutkan ke tim berikutnya. Proses ini memastikan bahwa setiap kali tim dibentuk, elemen yang digunakan dihapus dari pertimbangan untuk tim berikutnya. Jika fungsi berhasil membentuk *N* tim atau lebih dengan median minimal *x*, maka fungsi mengembalikan *true*. Jika tidak, fungsi mengembalikan *false*.

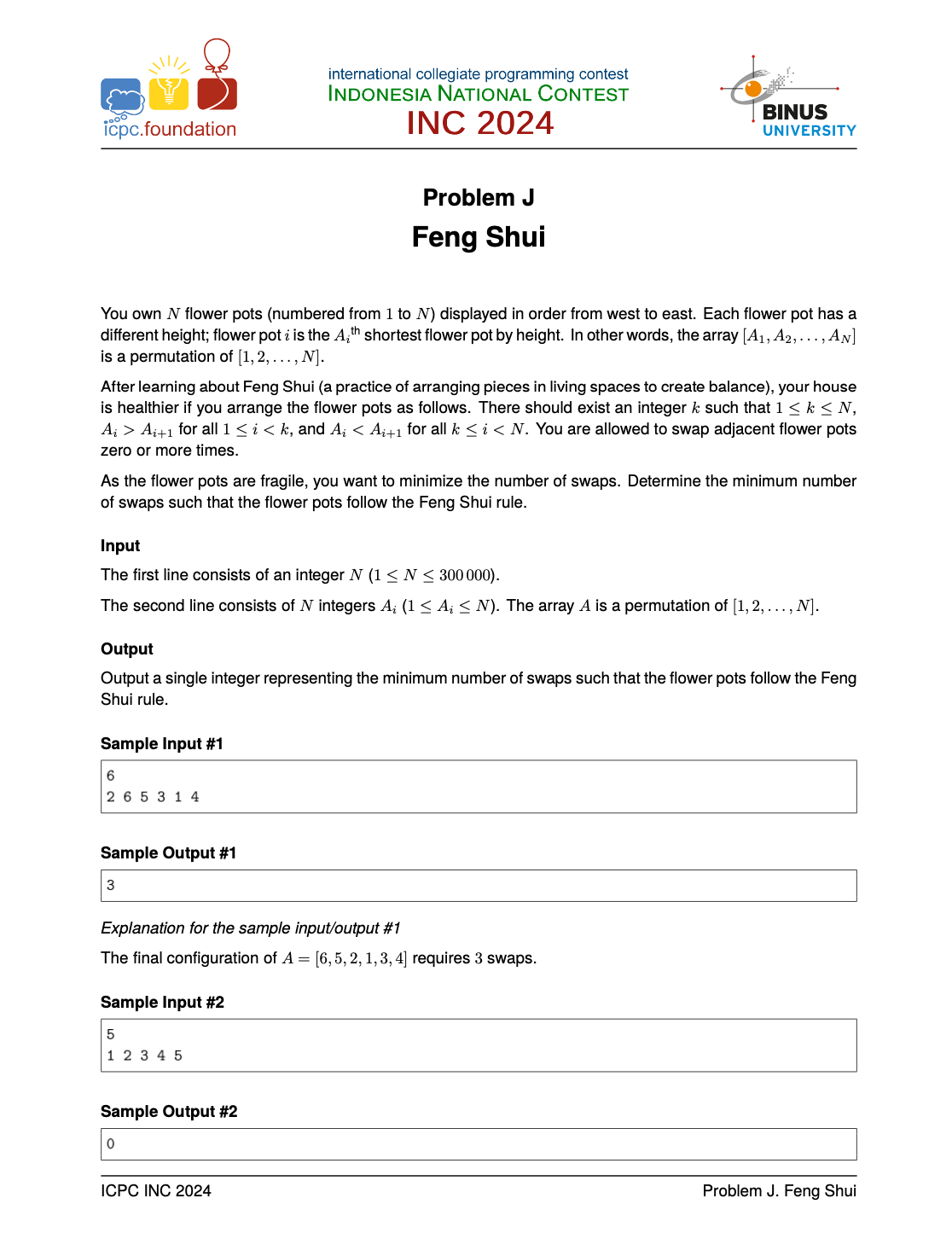
Binary search disini bekerja dengan menetapkan batas bawah (*left = 0*) dan batas atas (*right = 4000*) untuk nilai median. Nilai tengah (*mid*) dihitung, dan fungsi *isPossible* dipanggil untuk mengecek apakah median *x = mid* memungkinkan pembentukan *N* tim. Jika memungkinkan, nilai *mid* disimpan sebagai hasil sementara, dan pencarian dilanjutkan ke nilai median yang lebih tinggi (*left = mid + 1*). Sebaliknya, jika tidak memungkinkan, pencarian berlanjut ke median yang lebih rendah (*right = mid − 1*). Proses ini terus berulang hingga batas atas dan bawah bertemu, dan nilai median tertinggi yang valid disimpan dalam variabel *ans*, yang kemudian dicetak sebagai hasil.

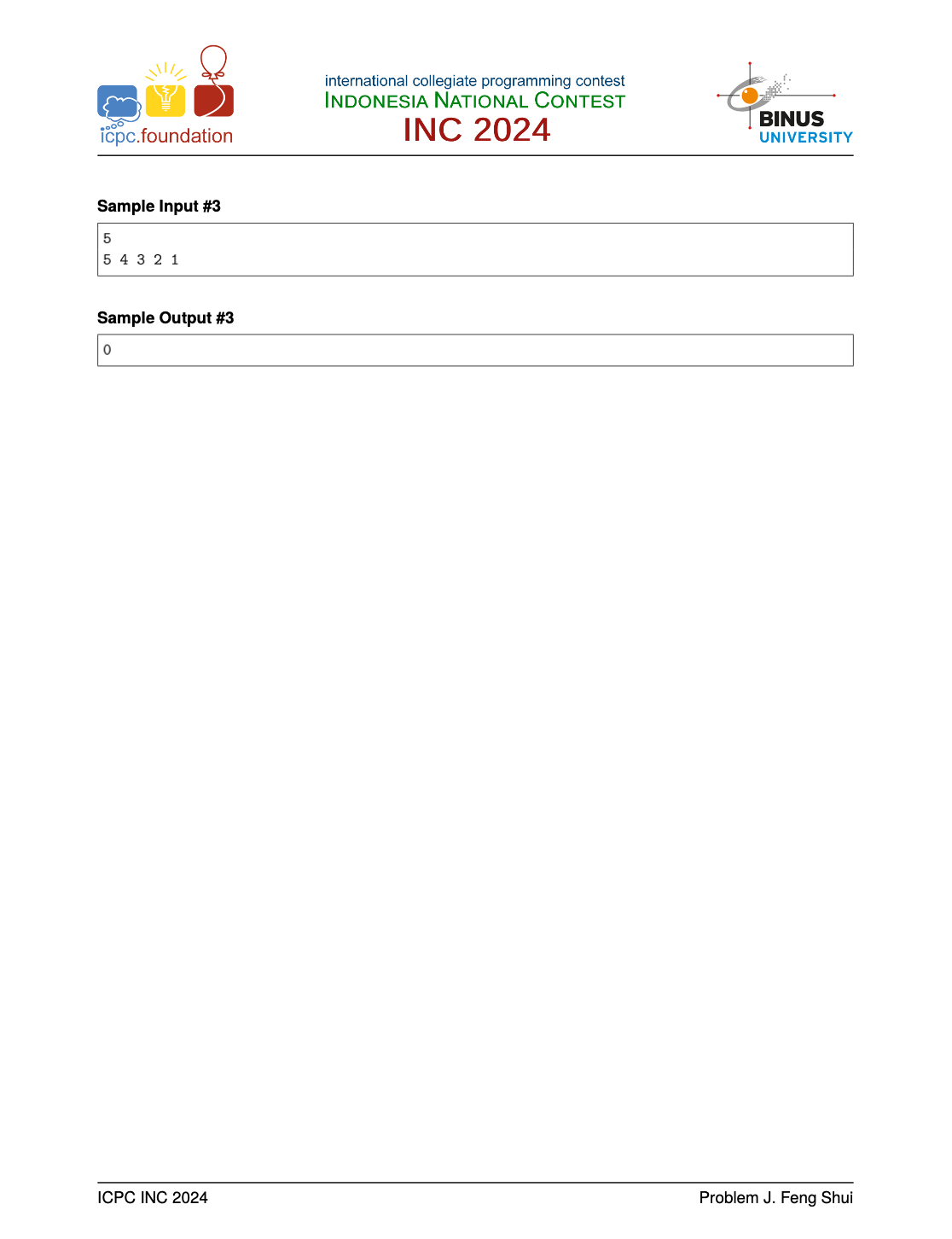
Time Complexity utama dari kode ini berasal dari pengurutan array *arr* dengan kompleksitas *O(3N log(3N))*, di mana 3N adalah jumlah total siswa. Lalu Binary search memerlukan *O(log(4000)) = O(12)* iterasi, dan setiap iterasi memanggil fungsi *isPossible*, yang berjalan dalam *O(3N)* karena memproses seluruh array. Dengan demikian, total Time Complexitynya adalah ***O(3N log(3N))***, yang mendominasi. Sedangkan untuk Space Complexitynya terdapat *O(3N)*, yang digunakan untuk menyimpan array *arr* serta Fungsi *isPossible* dan variabel tambahan seperti *left*, *right*, *mid*, dan *ans* yang menggunakan ruang konstan *O(1)*. Dengan demikian total Space Complexitynya adalah ***O(3N)****.*

1. Upsolve the unsolved problems in INC 2024 after the event.

**Jawaban :**

1. *Problem J – Feng Shui*





Unsolved code :

#include <stdio.h>

#include <stdlib.h>

#define MAX\_N 300005

// Function untuk menghitung jumlah swap minimum

long long countMinSwaps(int arr[], int n)

{

long long minSwaps = 1e18;

// Coba setiap kemungkinan titik k

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

{

int temp[MAX\_N];

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

{

temp[i] = arr[i];

}

long long swaps = 0;

// Sort descending untuk bagian pertama (1 to k-1)

for(int i = 0; i < k-1; i++)

{

for(int j = 0; j < k-1-i; j++)

{

if(temp[j] < temp[j+1])

{

int t = temp[j];

temp[j] = temp[j+1];

temp[j+1] = t;

swaps++;

}

}

}

// Sort ascending untuk bagian kedua (k to n-1)

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

{

for(int j = k; j < n-1-(i-k); j++)

{

if(temp[j] > temp[j+1])

{

int t = temp[j];

temp[j] = temp[j+1];

temp[j+1] = t;

swaps++;

}

}

}

// Cek apakah susunan sudah memenuhi aturan Feng Shui

int valid = 1;

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

{

if(temp[i-1] <= temp[i])

{

valid = 0;

break;

}

}

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

{

if(temp[i-1] >= temp[i])

{

valid = 0;

break;

}

}

if(valid && swaps < minSwaps)

{

minSwaps = swaps;

}

}

return minSwaps;

}

int main()

{

int n;

scanf("%d", &n);

int arr[MAX\_N];

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

{

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

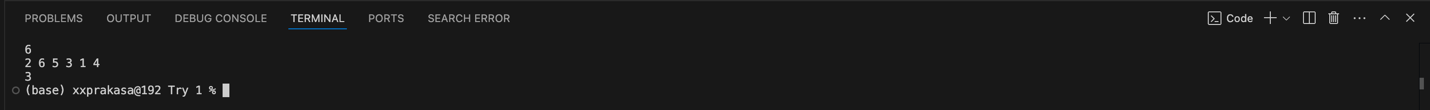
}

printf("%lld\n", countMinSwaps(arr, n));

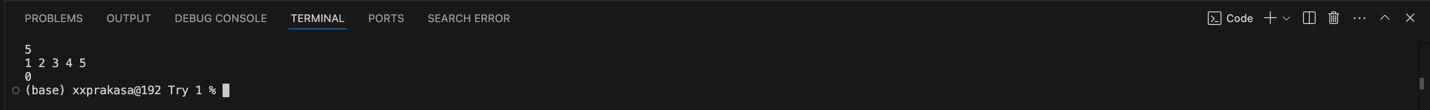
return 0;

}

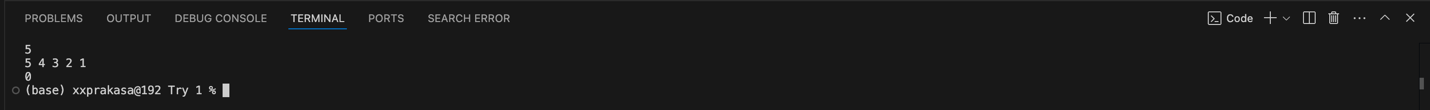
Sample Input & Output #1 :



Sample Input & Output #2 :



Sample Input & Output #3 :



**Analisis Unsolve :**

Dalam program yang kami buat, sebenarnya program telah mengeluarkan output yang sesuai dengan problem Feng Shui nya, namun sepertinya dalam program ini running time yang kami dapatkan adalah lebih dari 1 detik karena kami malah menggunakan algoritma bubble sort yang memiliki time complexity *O(N^2)*. Sebagai gantinya, kami akan merevisinya dengan menggunakan algoritma sorting yang lebih efisien, seperti quicksort atau mergesort, yang memiliki kompleksitas *O(N log N)*.

Upsolve code :

#include <stdio.h>

#include <stdlib.h>

#define MAX\_N 300005

// Function untuk menghitung jumlah swap minimum

long long countMinSwaps(int arr[], int n);

void mergeSortDescending(int arr[], int left, int right, long long \*swaps);

void mergeDescending(int arr[], int left, int mid, int right, long long \*swaps);

void mergeSortAscending(int arr[], int left, int right, long long \*swaps);

void mergeAscending(int arr[], int left, int mid, int right, long long \*swaps);

long long countMinSwaps(int arr[], int n)

{

long long minSwaps = 1e18;

// Coba setiap kemungkinan titik k

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

{

int temp[MAX\_N];

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

{

temp[i] = arr[i];

}

long long swaps = 0;

// Sort descending untuk bagian pertama (1 to k-1)

if (k > 1)

{

mergeSortDescending(temp, 0, k - 2, &swaps);

}

// Sort ascending untuk bagian kedua (k to n-1)

if (k <= n)

{

mergeSortAscending(temp, k - 1, n - 1, &swaps);

}

// Cek apakah susunan sudah memenuhi aturan Feng Shui

int valid = 1;

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

{

if(temp[i-1] <= temp[i])

{

valid = 0;

break;

}

}

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

{

if(temp[i-1] >= temp[i])

{

valid = 0;

break;

}

}

if(valid)

{

if(swaps < minSwaps)

{

minSwaps = swaps;

}

}

}

return minSwaps == 1e18 ? -1 : minSwaps; // Return -1 jika tidak ada konfigurasi yang valid

}

void mergeSortDescending(int arr[], int left, int right, long long \*swaps)

{

if (left >= right) return;

int mid = (left + right) / 2;

mergeSortDescending(arr, left, mid, swaps);

mergeSortDescending(arr, mid + 1, right, swaps);

mergeDescending(arr, left, mid, right, swaps);

}

void mergeDescending(int arr[], int left, int mid, int right, long long \*swaps)

{

int temp[MAX\_N];

int i = left, j = mid + 1, k = left;

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

{

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

{

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

}

else

{

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

\*swaps += (mid - i + 1); // Hitung swaps

}

}

while (i <= mid) temp[k++] = arr[i++];

while (j <= right) temp[k++] = arr[j++];

for (i = left; i <= right; i++) arr[i] = temp[i];

}

void mergeSortAscending(int arr[], int left, int right, long long \*swaps)

{

if (left >= right) return;

int mid = (left + right) / 2;

mergeSortAscending(arr, left, mid, swaps);

mergeSortAscending(arr, mid + 1, right, swaps);

mergeAscending(arr, left, mid, right, swaps);

}

void mergeAscending(int arr[], int left, int mid, int right, long long \*swaps)

{

int temp[MAX\_N];

int i = left, j = mid + 1, k = left;

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

{

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

{

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

}

else

{

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

\*swaps += (mid - i + 1); // Hitung swaps

}

}

while (i <= mid) temp[k++] = arr[i++];

while (j <= right) temp[k++] = arr[j++];

for (i = left; i <= right; i++) arr[i] = temp[i];

}

int main()

{

int n;

scanf("%d", &n);

int arr[MAX\_N];

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

{

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

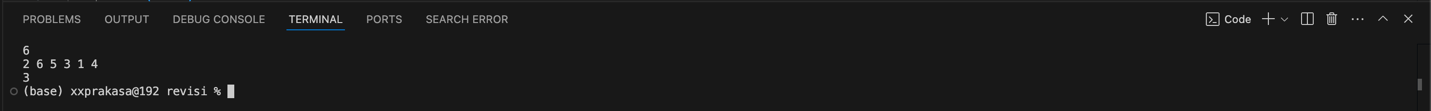
}

printf("%lld\n", countMinSwaps(arr, n));

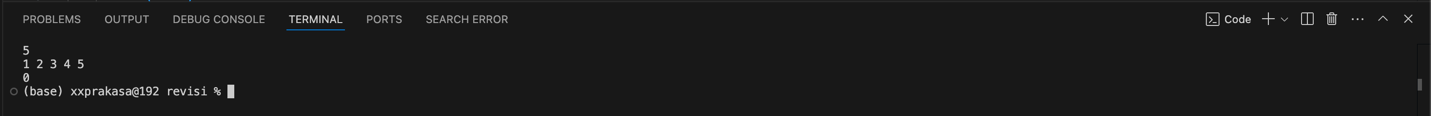
return 0;

}

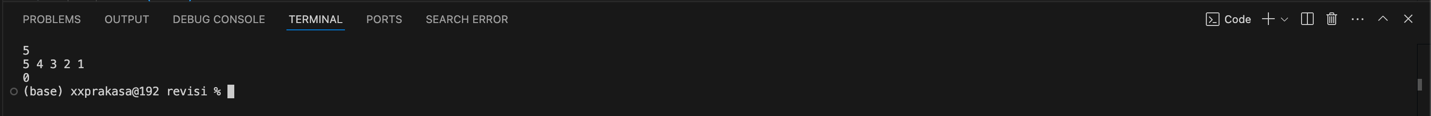
Sample Input & Output #1 :



Sample Input & Output #2 :



Sample Input & Output #3 :



**Analisis Upsolve :**

Jadi, program diatas bertujuan untuk menghitung jumlah swap minimum yang diperlukan untuk mengatur elemen *arr* menjadi dua bagian yang memenuhi aturan Feng Shui: elemen-elemen sebelum titik *k* (bagian pertama) harus diurutkan secara menurun (descending), sedangkan elemen-elemen setelah titik *k* (bagian kedua) harus diurutkan secara menaik (ascending).

Fungsi *countMinSwaps* mencoba setiap kemungkinan titik *k* dari 1 hingga n untuk membagi array menjadi dua bagian. Variabel *minSwaps* diinisialisasi dengan nilai yang sangat besar (10^18) sebagai pembanding untuk memastikan nilai swap terkecil ditemukan selama iterasi. Array sementara *temp* digunakan untuk menyimpan salinan array asli *arr* di setiap percobaan *k*, sehingga tidak mengubah data asli selama proses pengurutan dan validasi. Untuk setiap *k*, array dibagi menjadi dua bagian:

* Elemen dari indeks 0 hingga *k−1* diurutkan menurun menggunakan fungsi *mergeSortDescending*
* Elemen dari *k* hingga *n-1* diurutkan menaik menggunakan fungsi *mergeSortAscending*.

Variabel *swaps* mencatat jumlah swap yang diperlukan untuk mengurutkan kedua bagian tersebut, yang dihitung selama penggabungan subarray oleh fungsi *mergeDescending* dan *mergeAscending*. Setelah array terurut sesuai aturan, validasi dilakukan untuk memastikan bahwa bagian pertama benar-benar menurun dan bagian kedua benar-benar menaik. Jika validasi gagal, konfigurasi titik *k* dianggap tidak valid, dan iterasi dilanjutkan ke nilai *k* berikutnya. Jika validasi berhasil, jumlah swap yang diperlukan dibandingkan dengan nilai *minSwaps*, dan nilai terkecil diperbarui.

Time Complexity utama berasal dari penggunaan merge sort untuk mengurutkan dua bagian array pada setiap titik *k*. Merge sort memiliki kompleksitas waktu *O(n log n)* untuk mengurutkan n elemen, sehingga untuk setiap *k*, kompleksitasnya adalah *O(k log k + (n - k) log(n - k))*. Jika proses ini dilakukan untuk semua *k*, total Time Complexitynya mendekati ***O(n^2 log n)***, yang mendominasi langkah validasi dengan kompleksitas *O(n)* per titik *k*. Sedangkan untuk Space Complexity program adalah ***O(n)***, yang berasal dari penggunaan array tambahan *temp* untuk menyimpan salinan sementara elemen-elemen array selama proses penggabungan merge sort.