Kelas : LB01

Kelompok : 17

Anggota Kelompok :

1. 2602089143 – Jonathan Alvindo Fernandi
2. 2602089793 – Kevyn Aprilyanto
3. 2602101653 – Andrew Alfonso Lie

Departemen : *Computer Science*

Kampus : Kemanggisan

***AoL (Assurance of Learning) Case Study***

**Mata Kuliah *COMP6049001 – Algorithm Design and Analysis***

1. *Write an analysis of the solutions used in INC 2023!*

Jawaban:

1. *Problem A – Golden Tickets*

*Code*:

#include <stdio.h>

#include <string.h>

int main() {

int n, m, k, x = 0;

char s[100][11], t[100][11], topIns[100][11], goldTeam[100][11];

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

getchar();

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

scanf("%s %[^\n]", s[i], t[i]);

getchar();

if(i < m) {

strcpy(topIns[i], t[i]);

}

}

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

if(k == 0) {

break;

} else {

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

if(!strcmp(t[j], t[i])) {

break;

} else if(j == (i - 1)) {

strcpy(goldTeam[x], s[i]);

x++;

k--;

if(k == 0) {

break;

}

}

}

}

}

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

if(x != 0) {

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

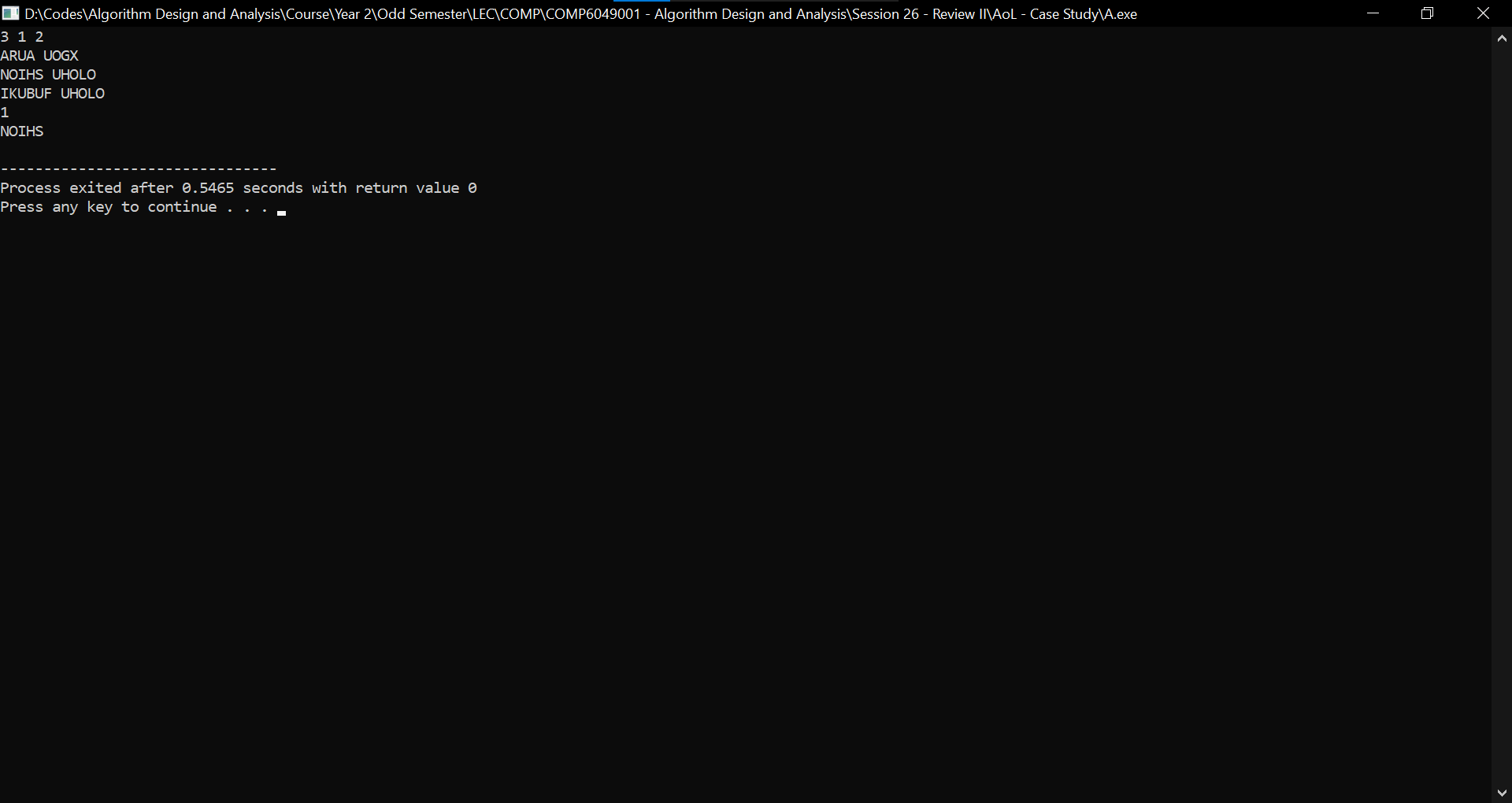
puts(goldTeam[i]);

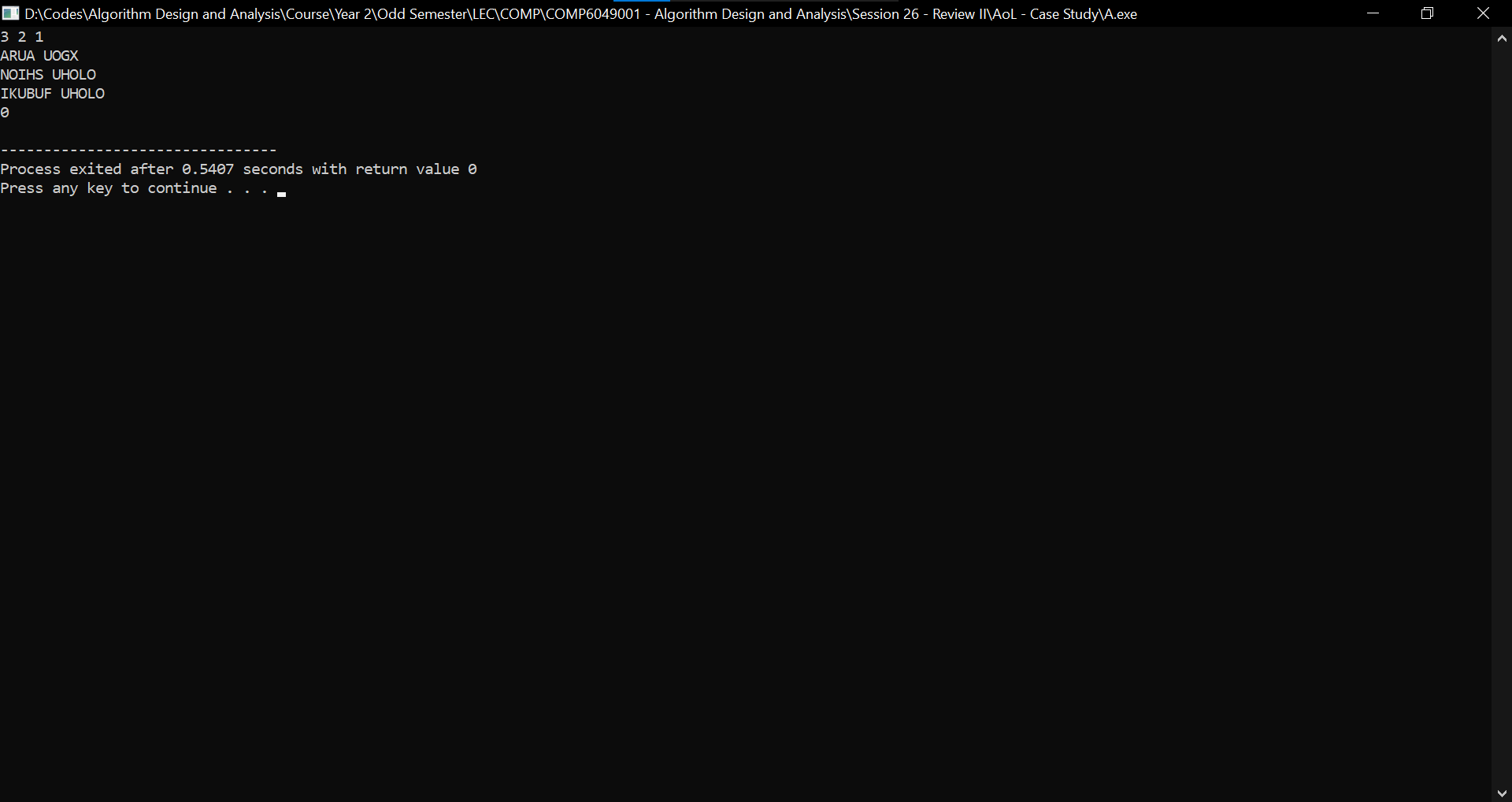
}

}

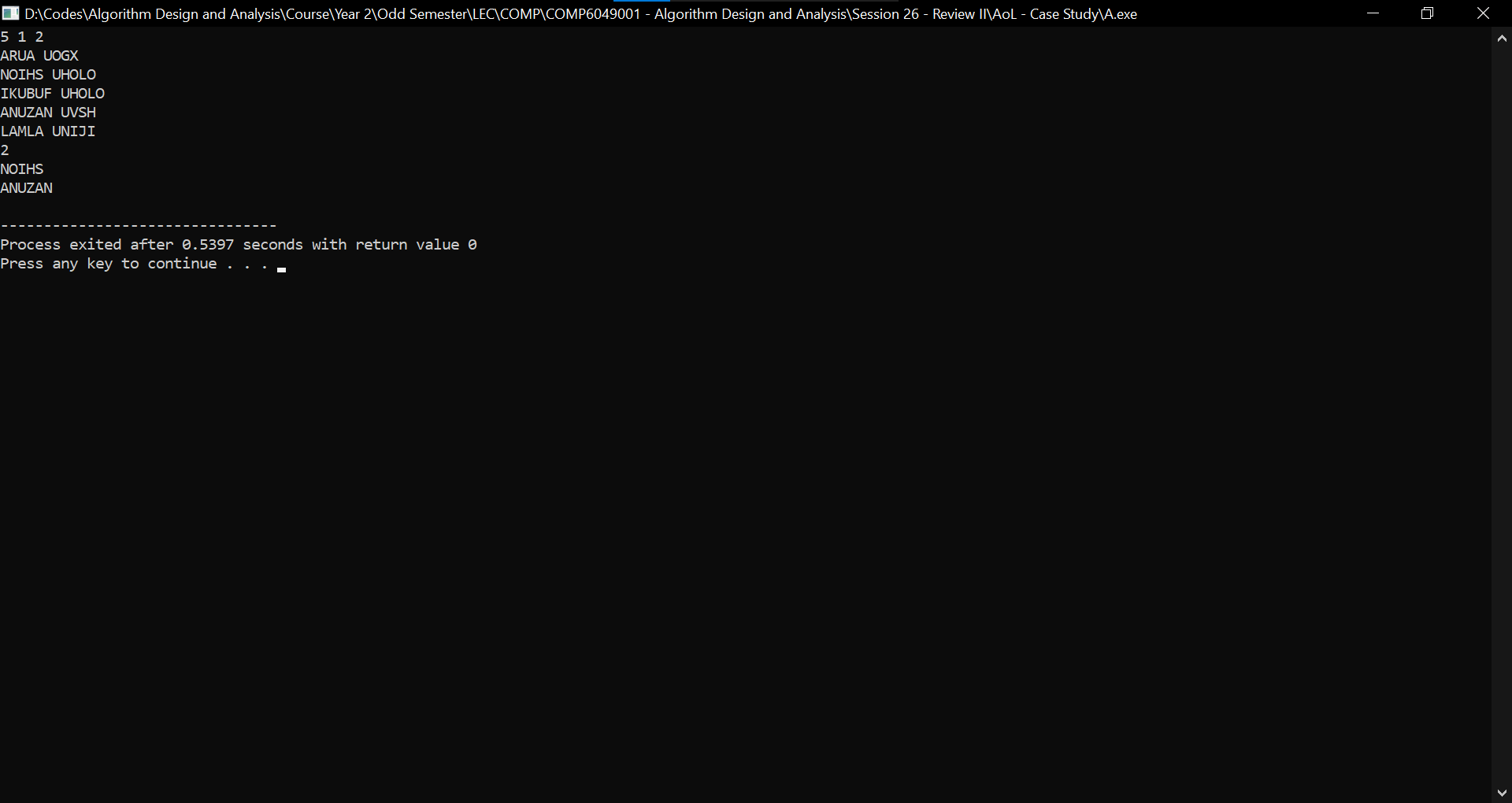
return 0;

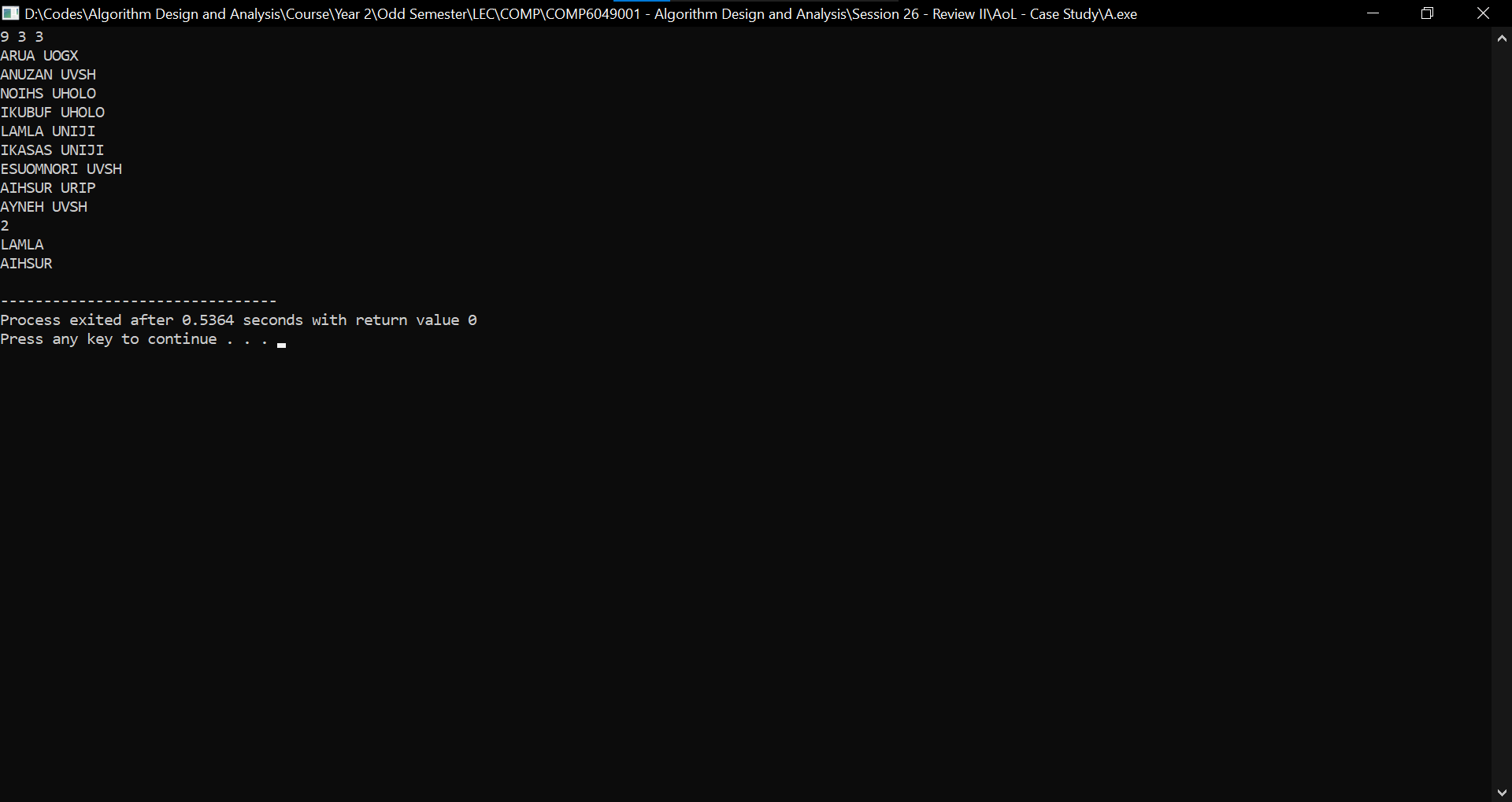
}

*Output #1*:

*Output #2*:

*Output #3*:



*Output #4*:

Analisis:

Program di atas pertama-tama membaca jumlah tim (*n*), jumlah tim teratas (*m*), serta jumlah *Golden Tickets* (*k*). Program tersebut juga membaca nama-nama tim (*s*) beserta institusinya (*t*). Lalu, nama-nama institusi dari *m* tim teratas disimpan ke dalam *topIns*. Tim yang tersisa dicek apakah institusinya tidak masuk dalam institusi *m* tim teratas. Jika tidak, tim tersebut akan ditambahkan ke daftar penerima *Golden Tickets* (*goldTeam*). Program tersebut kemudian akan mencetak jumlah tim penerima *Golden Tickets* (*x*) beserta nama dari timnya (*goldTeam*). Program tersebut memiliki *time complexity* sebesar *O(n2)* akibat stuktur *nested loop* yang digunakan untuk membandingkan institusi dari tim yang tersisa dengan daftar institusi *m* tim teratas. Program tersebut juga memiliki *space complexity* sebesar *O(n)* guna menyimpan nama-nama dari tim beserta institusinya.

1. *Problem B – Diet Plan*

*Code*:

#include <stdio.h>

#include <stdlib.h>

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

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

}

int main() {

int n, m, k;

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

getchar();

int p[n], maxDays = 0;

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

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

}

getchar();

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

int temp[n], count = 0;

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

count += p[j];

}

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

temp[j] = p[j];

}

qsort((temp + 1), i, sizeof(int), compare);

for(int j = 1; j <= k; j++) {

count -= temp[j];

}

if(count <= m) {

if(i > maxDays) {

maxDays = i;

}

}

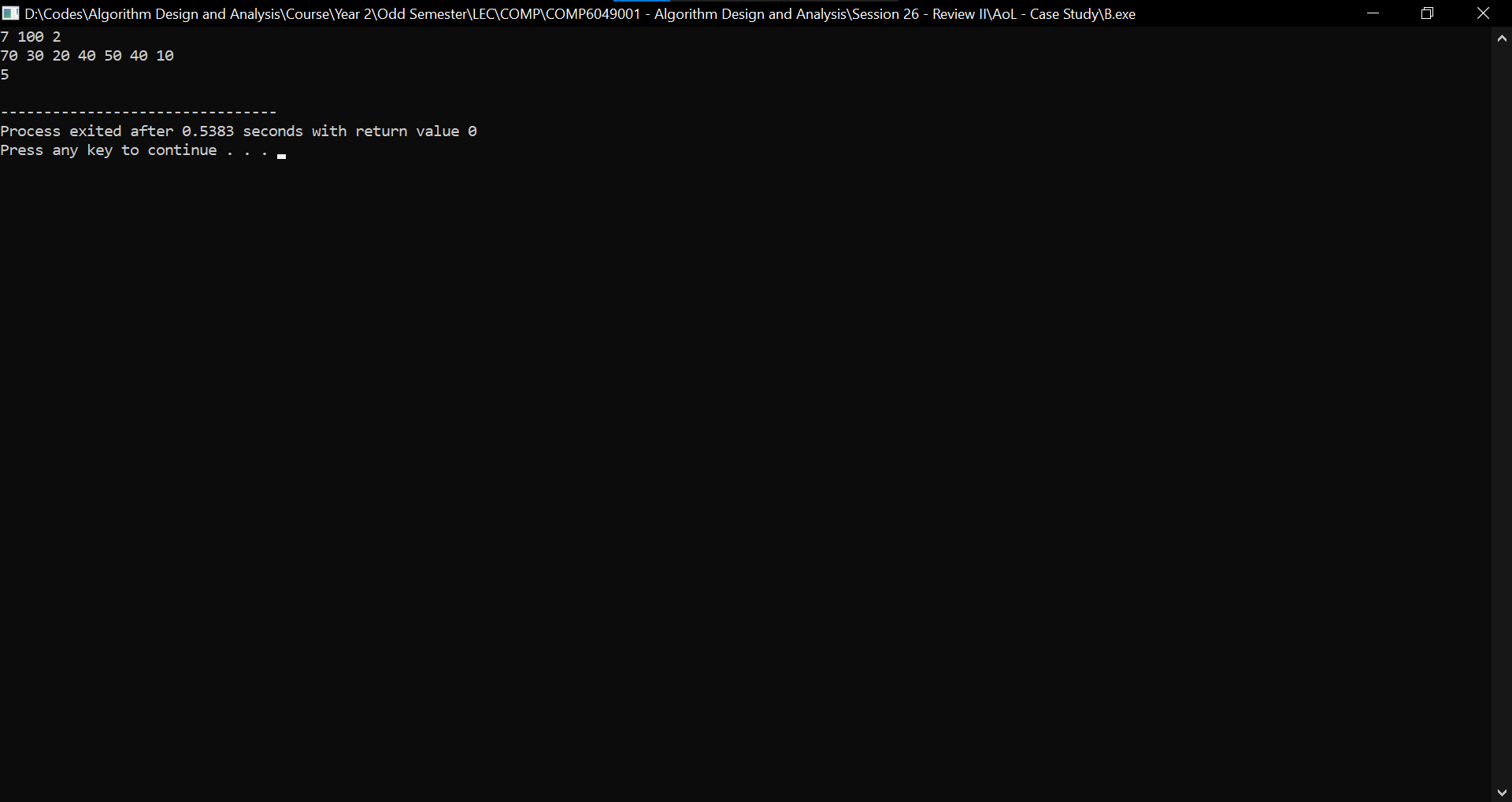
}

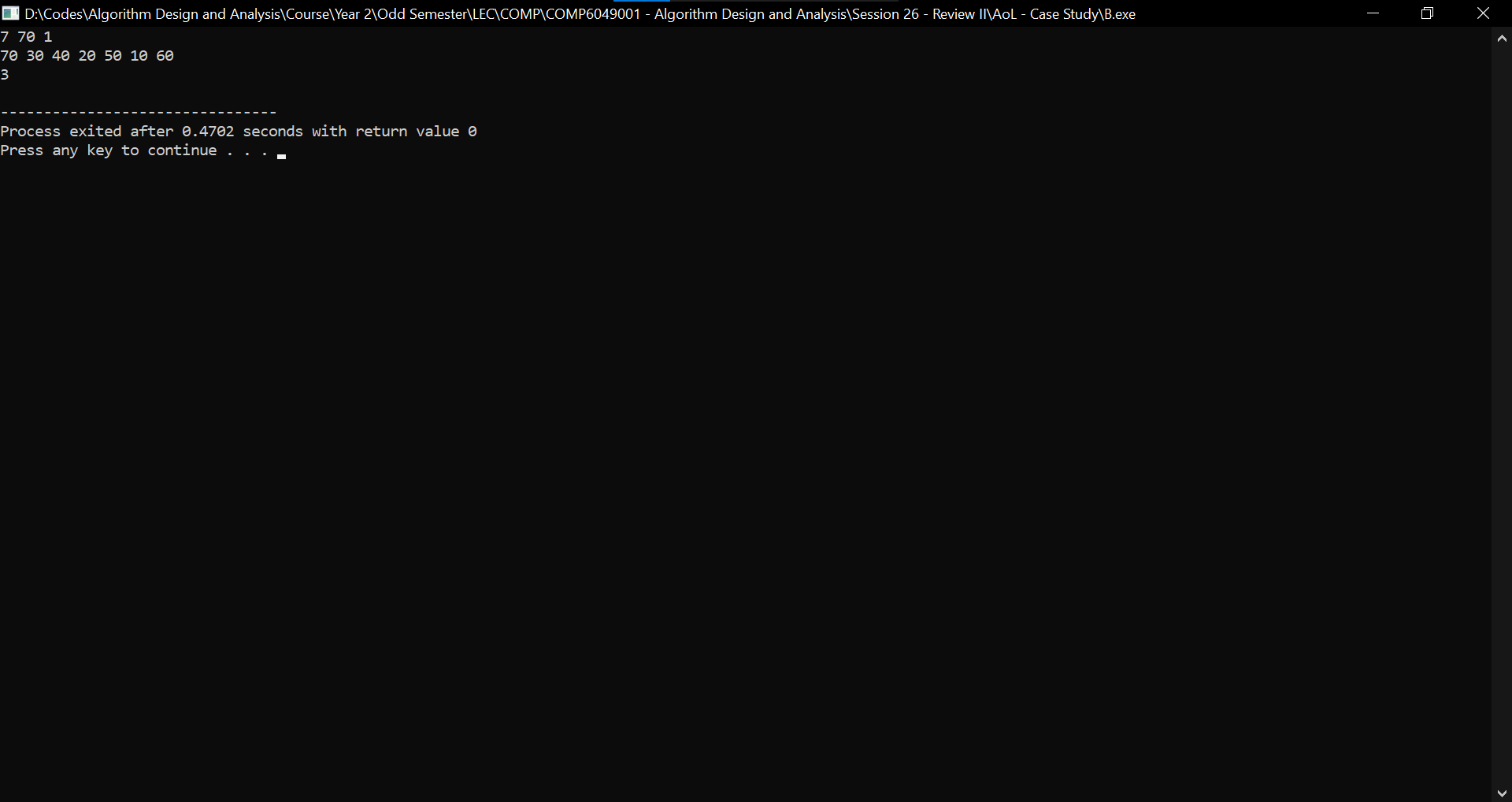
printf("%d\n", maxDays);

return 0;

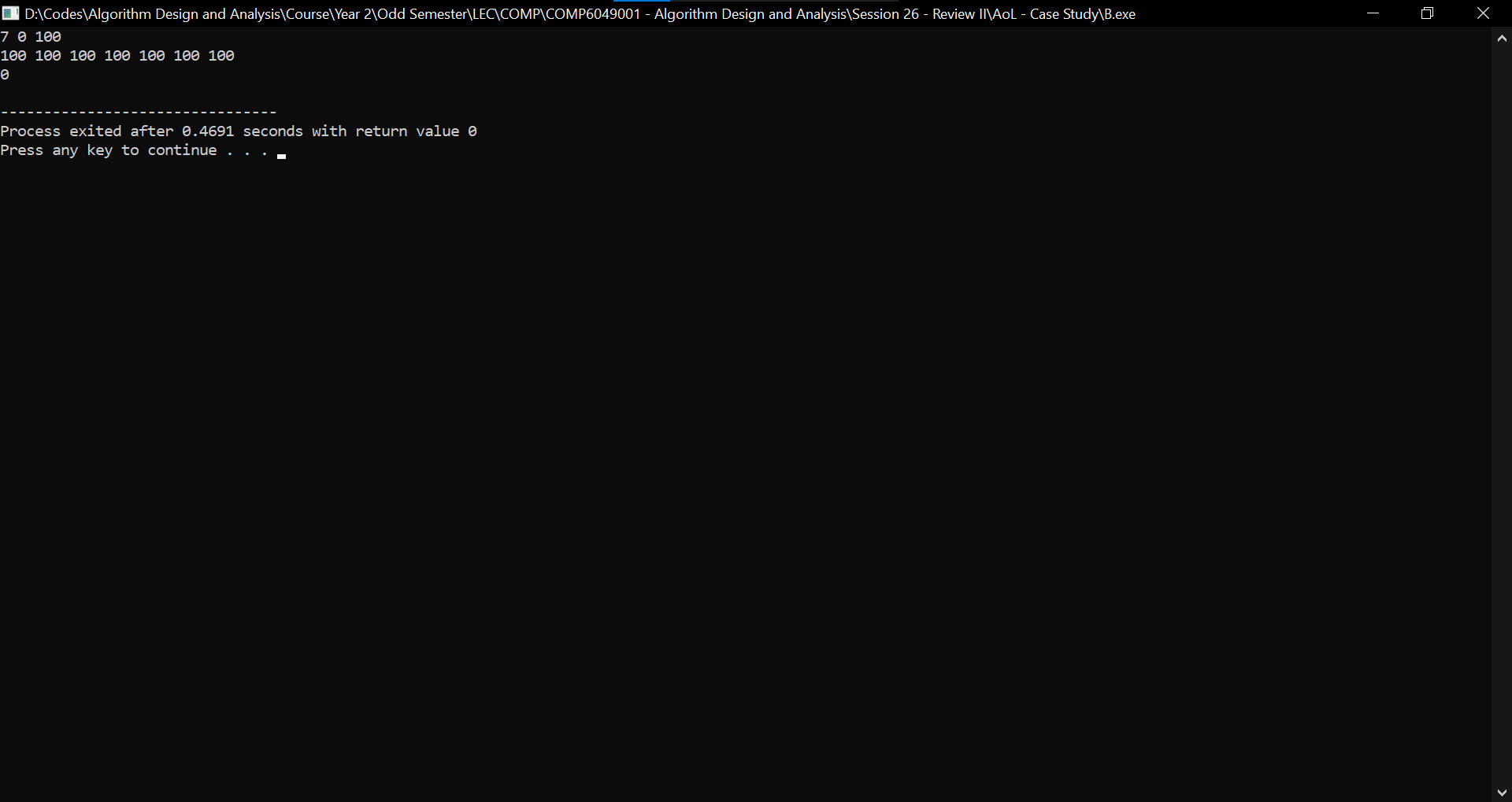
}

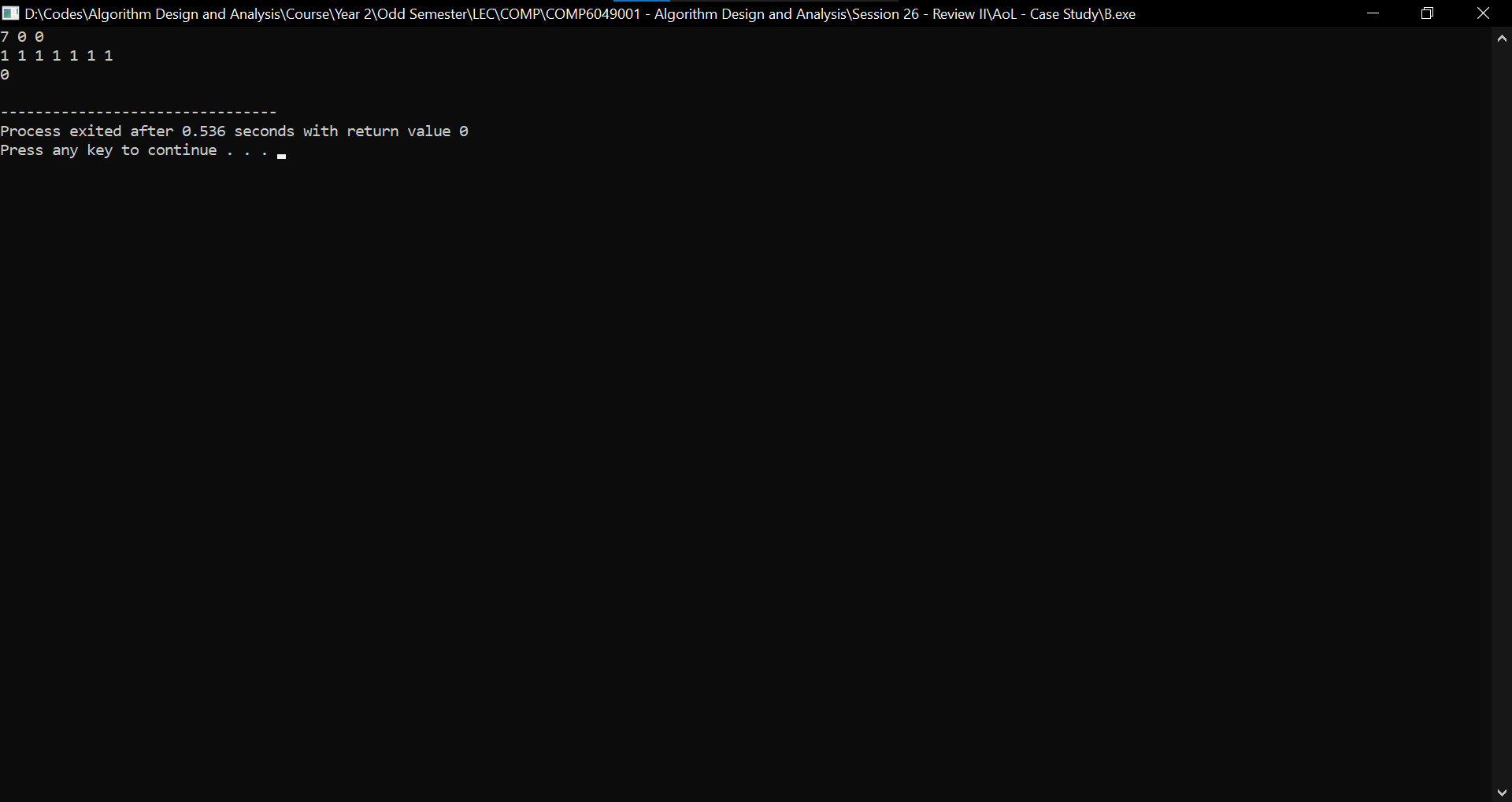
*Output #1*:



*Output #2*:

*Output #3*:



*Output #4*:

Analisis:

Program di atas pertama-tama membaca jumlah hari diet (*n*), jumlah susu (*m*), serta jumlah biskuit (*k*). Program tersebut juga membaca jumlah susu yang diperlukan setiap harinya (*p*). Untuk setiap harinya, jumlah total susu yang diperlukan hingga hari tersebut (*count*) dihitung dan jumlah susu yang diperlukan setiap harinya (*p*) disimpan ke dalam sebuah *array* temporer (*temp*). Fungsi *qsort* dipanggil guna mengurutkan a*rray* temporer (*temp*) tersebut dengan urutan menurun. Strategi *greedy* digunakan dalam mengurangi kebutuhan susu pada *k* hari tertinggi (*temp*) dari jumlah total susu yang diperlukan hingga hari tersebut (*count*). Hal tersebut dilakukan guna menggantikan konsumsi susu pada hari-hari tersebut menjadi biskuit. Program tersebut kemudian akan mencetak jumlah hari maksimum rencana diet tersebut dapat dipertahankan (*days*). Program tersebut memiliki *time complexity* sebesar *O(n2 log n)* akibat fungsi *qsort* yang dipanggil dalam setiap kali perulangan. Program tersebut juga memiliki *space complexity* sebesar *O(n)* guna menyimpan jumlah susu yang diperlukan.

1. *Problem E – Reverse Severer*

*Code*:

#include <stdio.h>

#include <string.h>

int main() {

int n, q;

scanf("%d", &n);

getchar();

char s[n + 1], t[n + 1], temp[n + 1];

scanf("%[^\n]", s);

getchar();

scanf("%d", &q);

getchar();

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

scanf("%[^\n]", t);

getchar();

int possibility = 1, idx = 0;

memset(temp, 0, sizeof(temp));

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

strncat(temp, &t[j], 1);

if(s[n - idx - 1] == t[j]) {

if(!strncmp(&s[n - j - 1], temp, (j - idx + 1))) {

idx = j + 1;

memset(temp, 0, sizeof(temp));

}

}

if(strlen(temp) && (j == (n - 1))) {

possibility--;

}

}

if(!possibility) {

puts("NO");

} else {

puts("YES");

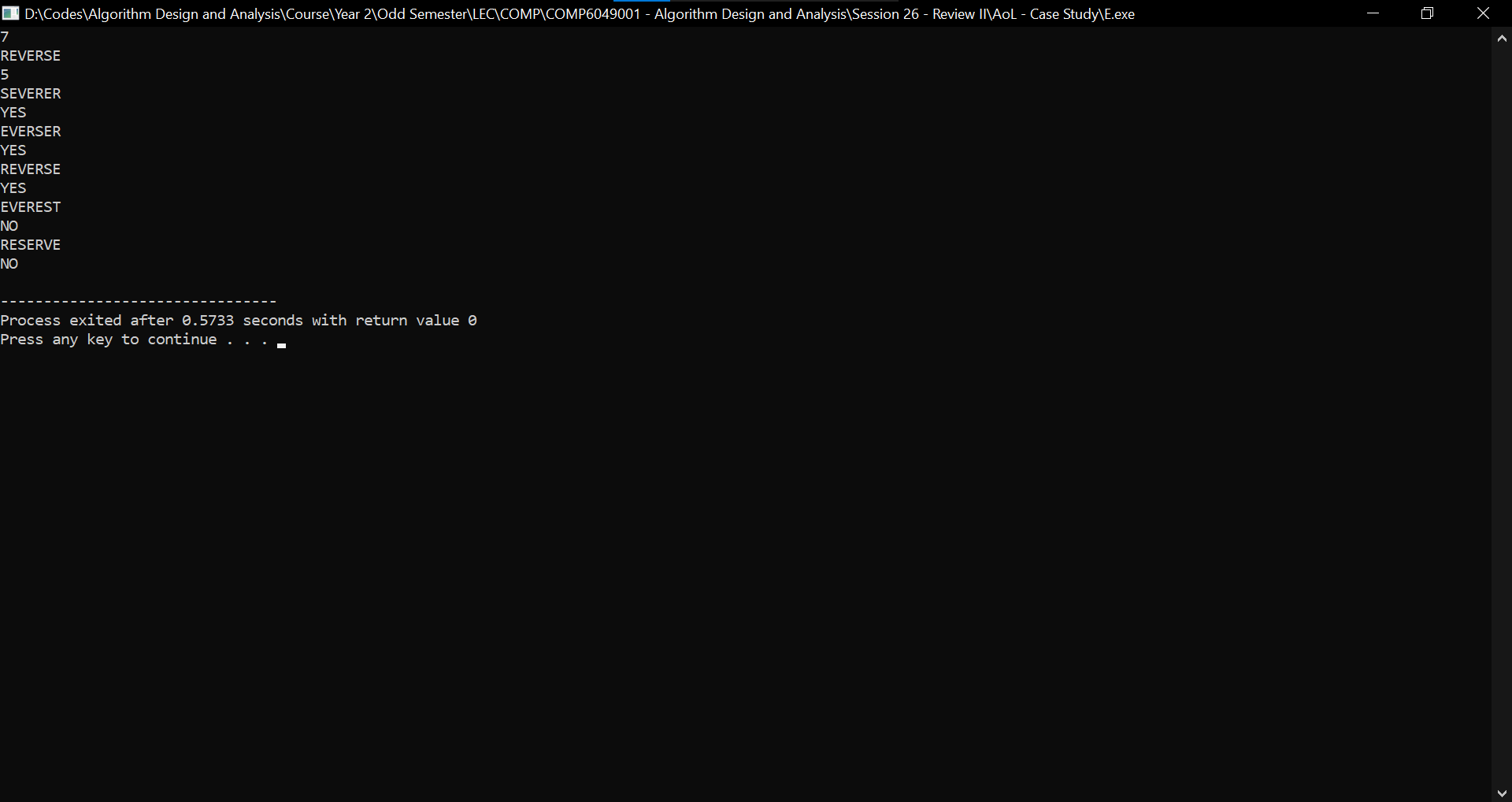
}

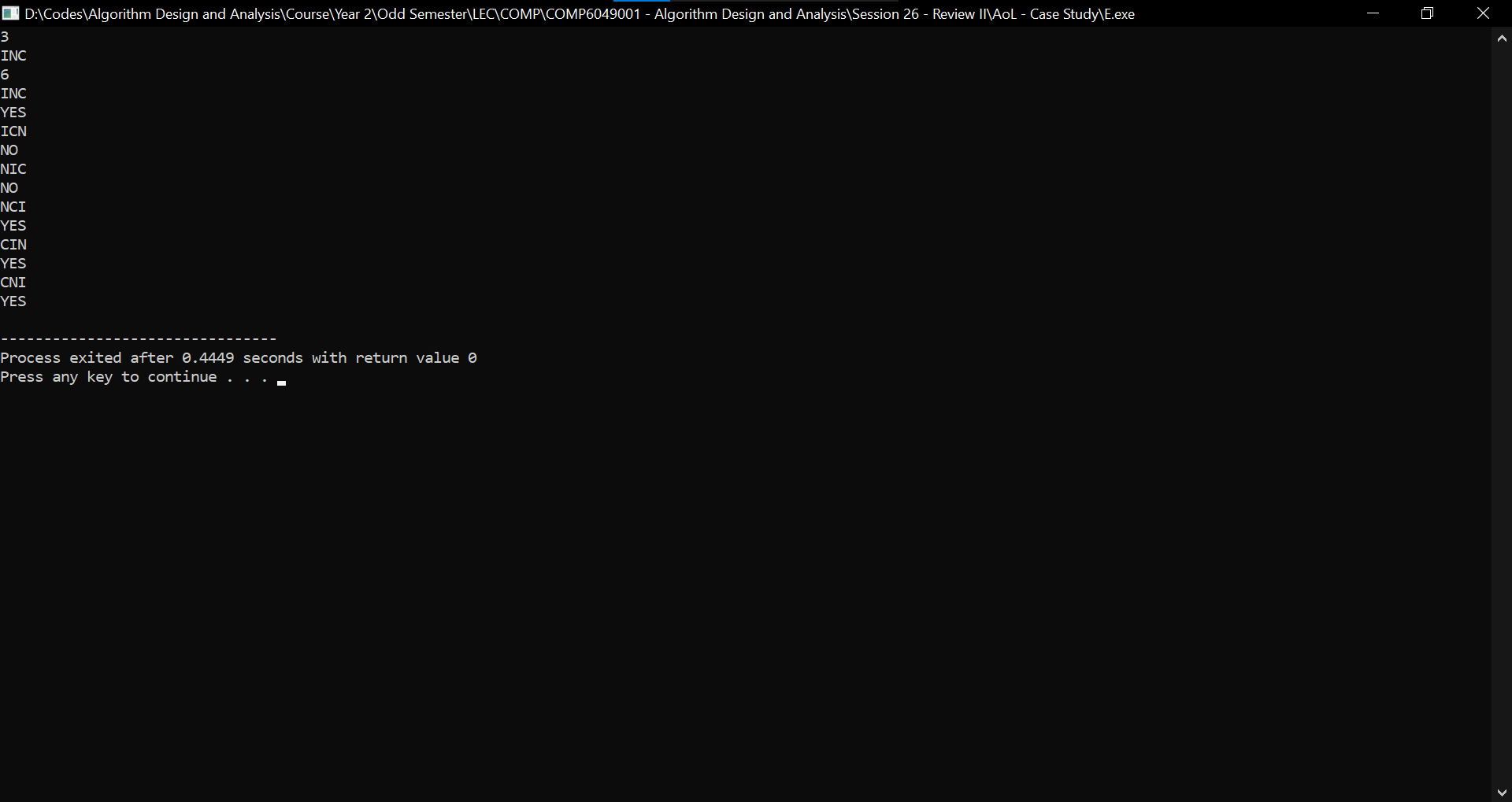
}

return 0;

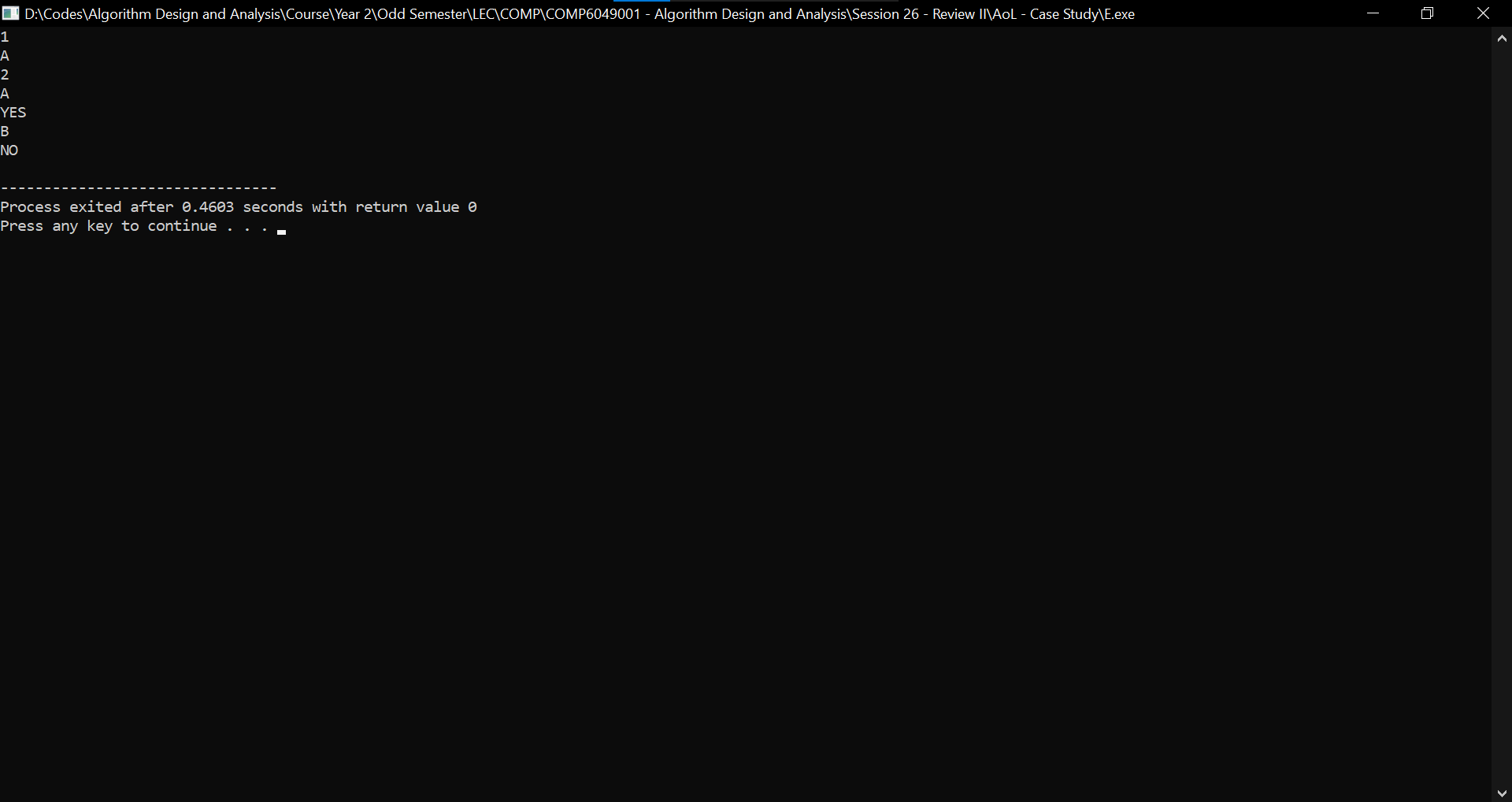
}

*Output #1*:



*Output #2*:

*Output #3*:



Analisis:

Program di atas pertama-tama membaca panjang *string* (*n*) dan *string* (*s*). Program tersebut juga membaca jumlah kueri (*q*) dan *string* target dari setiap kueri (*t*). Guna melacak karakter terakhir yang cocok dalam *string* (*s*), diinisialisasi sebuah *string* temporer (*temp*) dan sebuah indeks (*idx*). Setiap karakter dalam *string* target (*t*) dilakukan iterasi untuk menambahkan karakter tersebut ke dalam *string* temporer (*temp*). Jika karakter saat ini di *string* target (*t*) cocok dengan karakter pada posisi terkait dari akhir *string* (*s*), dilakukan pengecekan apakah *substring* dalam *string* temporer (*temp*) cocok dengan *substring* dalam *string* (s) dimulai dari posisi saat ini hingga akhir. Jika *substring* tersebut cocok, *string* temporer (*temp*) akan direset dan *idx* akan diperbarui menjadi posisi saat ini. Jika masih terdapat karakter yang tersisa di dalam *string* temporer (*temp*) pada akhir iterasi, dapat disimpulkan bahwa *string* target (*t*) tidak dapat diperoleh dari *substring* dalam *string* (*s*) menggunakan operasi tersebut. Dengan demikian, program tersebut akan mencetak “NO”.Jika seluruh karakter dalam *string* temporer (*t*) cocok dengan *substring* dalam *string* (*s*) menggunakan operasi tersebut, program akan mencetak “YES”. Program tersebut memiliki *time complexity* sebesar *O(n2)*, di mana *n* merupakan panjang dari *string*. Program tersebut juga memiliki *space complexity* sebesar *O(n)* guna menyimpan *string*, *string* target, dan *string* temporer.

1. *Problem H – Horse Carts*

*Code:*

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100000

typedef struct {

int w;

int v;

} Treasure;

typedef struct {

Treasure heap[MAX\_SIZE];

int size;

} PriorityQueue;

void push(PriorityQueue \*pq, Treasure val) {

int i = pq->size++;

while (i > 0) {

int j = ((i - 1) / 2);

if(pq->heap[j].v >= val.v) {

break;

}

pq->heap[i] = pq->heap[j];

i = j;

}

pq->heap[i] = val;

}

Treasure top(PriorityQueue \*pq) {

return pq->heap[0];

}

void pop(PriorityQueue \*pq) {

Treasure val = pq->heap[--pq->size];

int i = 0;

while((i \* 2 + 1) < pq->size) {

int left = i \* 2 + 1;

int right = i \* 2 + 2;

int j = left;

if((right < pq->size) && (pq->heap[right].v > pq->heap[left].v)) {

j = right;

}

if(pq->heap[j].v <= val.v) {

break;

}

pq->heap[i] = pq->heap[j];

i = j;

}

pq->heap[i] = val;

}

int empty(PriorityQueue \*pq) {

return (pq->size == 0);

}

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

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

}

int main() {

int x, y;

scanf("%d %d", &x, &y);

getchar();

Treasure treasures[x];

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

scanf("%d %d", &treasures[i].w, &treasures[i].v);

getchar();

}

int carts[y];

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

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

}

getchar();

qsort(treasures, x, sizeof(Treasure), compare);

qsort(carts, y, sizeof(int), compare);

PriorityQueue pq;

pq.size = 0;

int j = 0;

long long totalVal = 0;

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

while ((j < x) && (treasures[j].w <= carts[i])) {

push(&pq, treasures[j++]);

}

if(!empty(&pq)) {

totalVal += top(&pq).v;

pop(&pq);

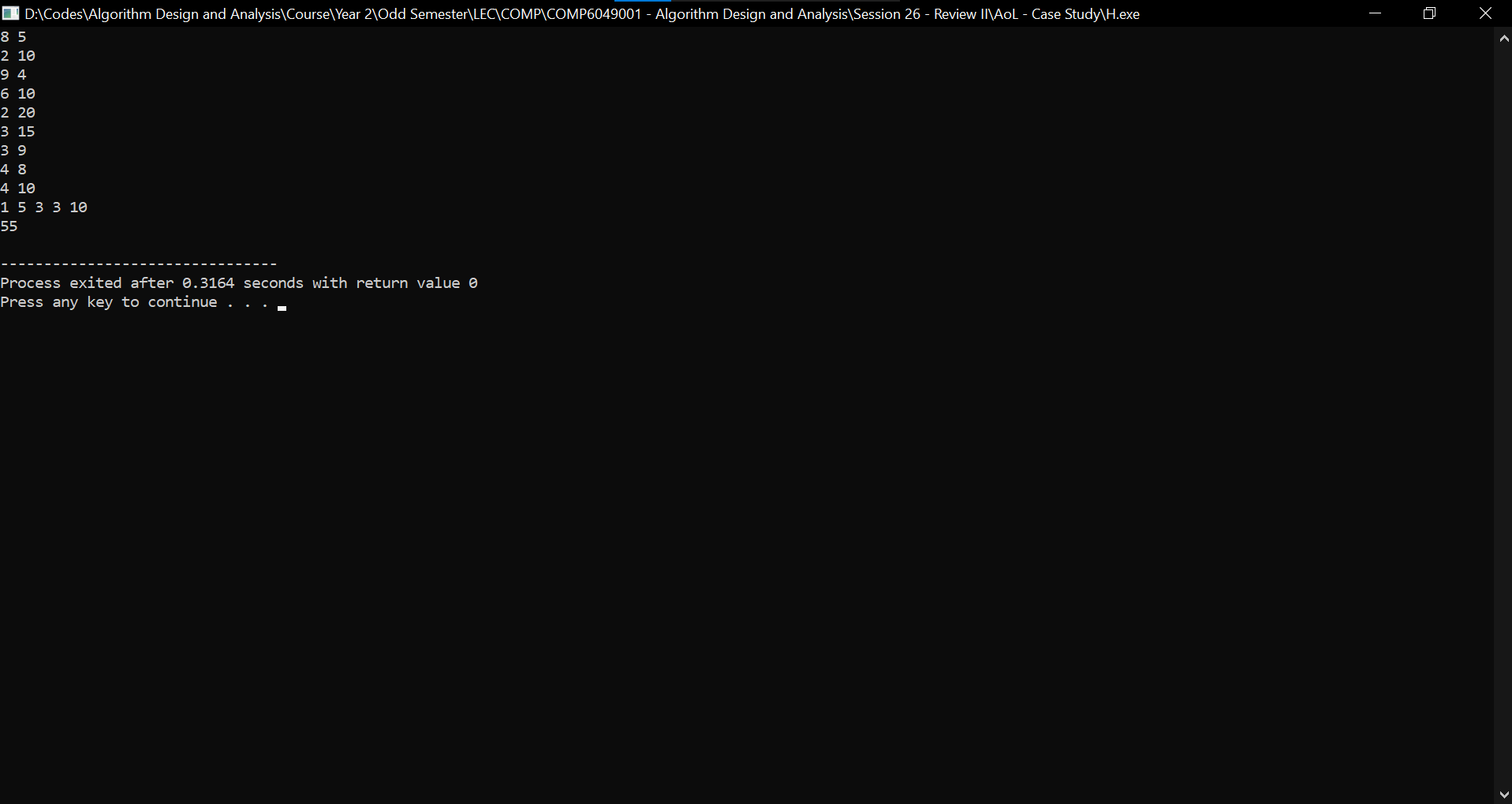
}

}

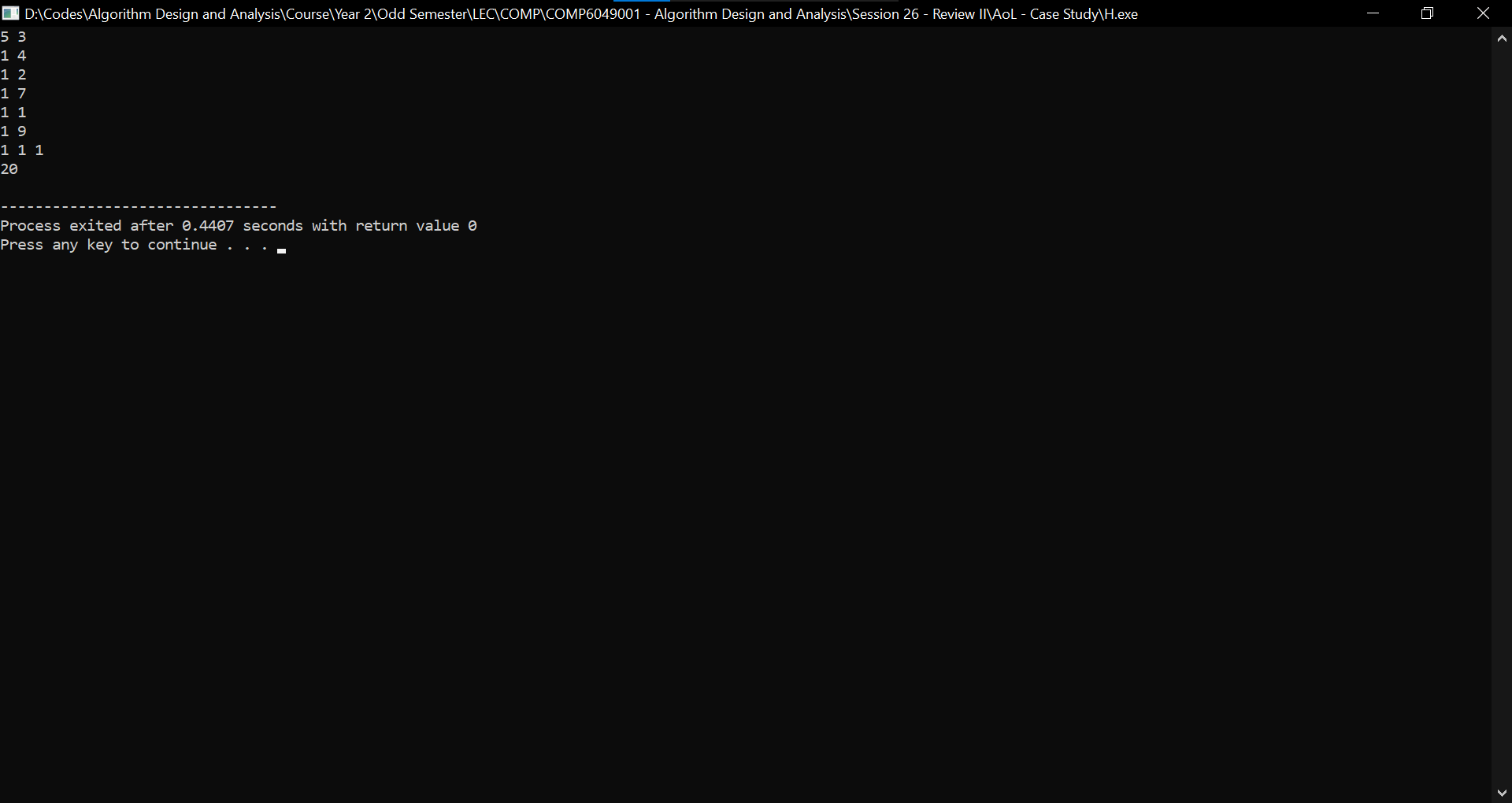
printf("%lld\n", totalVal);

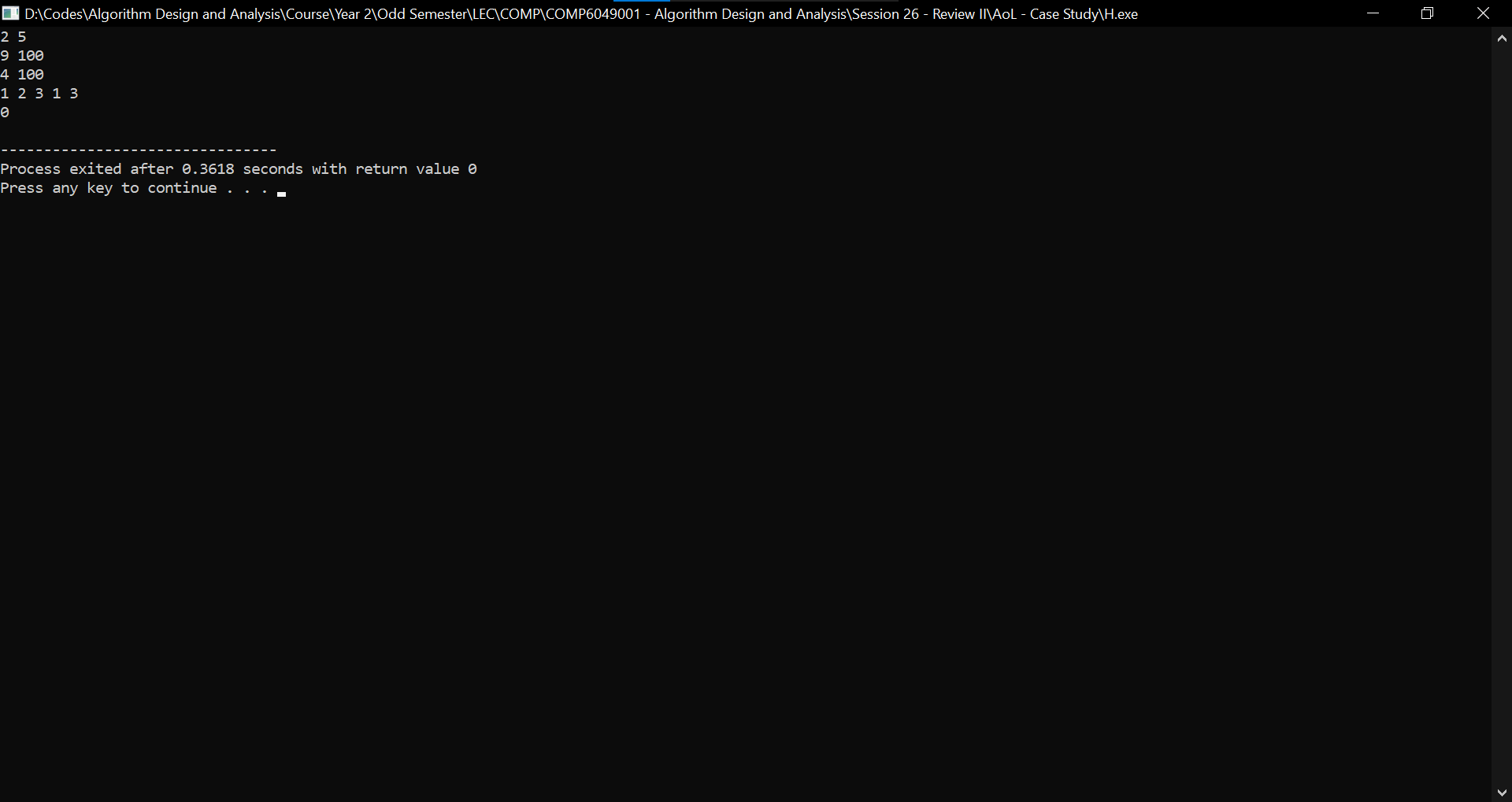
return 0;

}

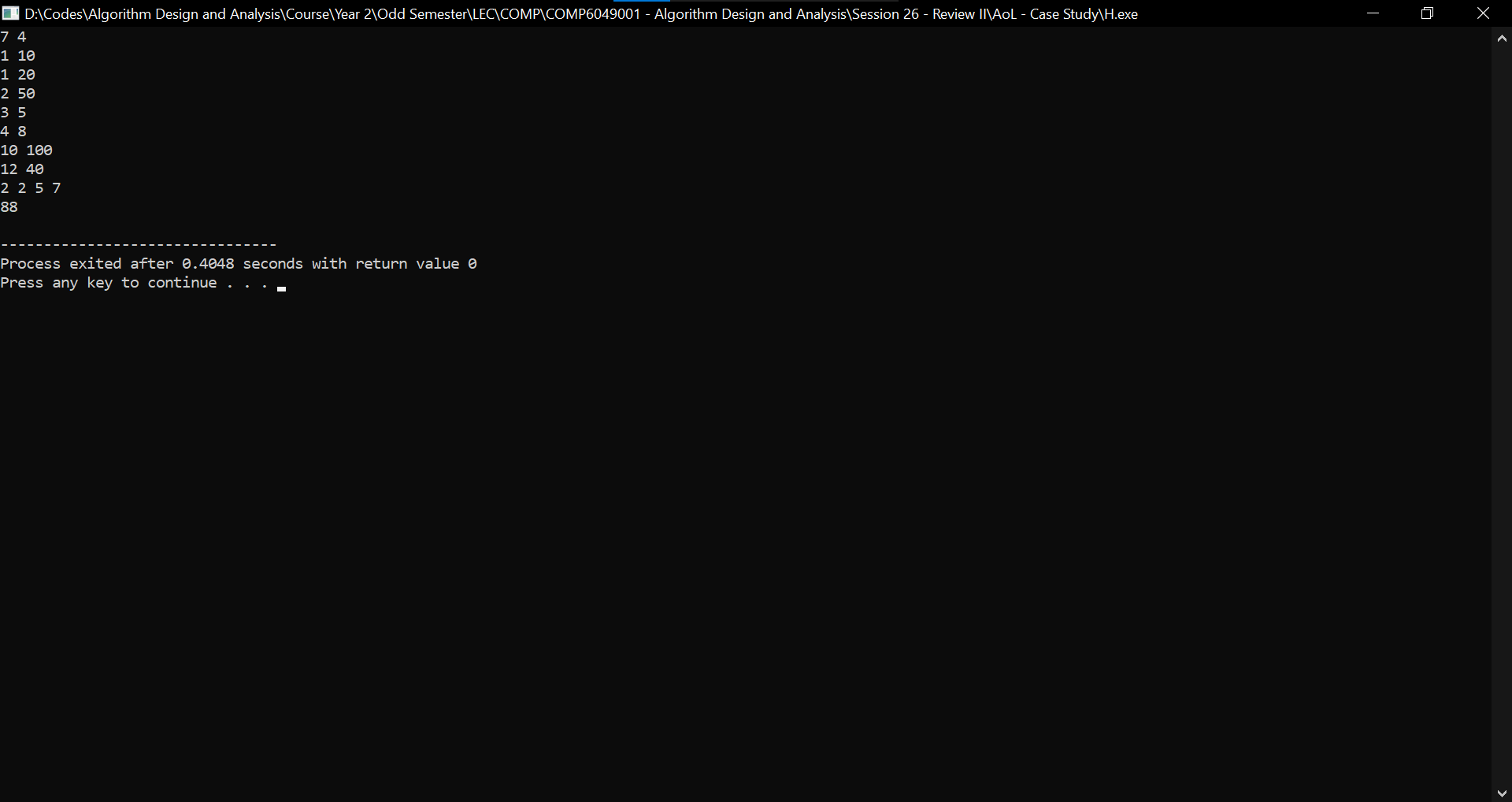
*Output #1*:

*Output #2*:



*Output #3*:

*Output #4*:



Analisis:

Program di atas pertama-tama membaca jumlah harta karun (*n*) dan jumlah kereta kuda (*m*). Program tersebut juga membaca berat (*w*) dan nilai dari setiap harta karun (*v*), serta berat maksimum yang dapat diangkut oleh setiap kereta kuda (*s*). Harta karun (*treasures*) dan kereta kuda tersebut (*s*) diurutkan terlebih dahulu secara menaik berdasarkan beratnya dengan menggunakan fungsi *qsort*. Harta karun tersebut (*treasures*) kemudian disimpan ke dalam *priority queue* (*pq*), di mana harta karun dengan nilai (*v*) tertinggi berada di *head*. Untuk setiap kereta kuda, dilakukan iterasi guna mengecek apakah ada harta karun yang dapat diangkut oleh kereta kuda tersebut, yaitu apabila berat dari harta karun tersebut (*w*) kurang dari atau sama dengan batas berat dari kereta kuda tersebut (*s*). Jika ada, harta karun tersebut akan di-*push* ke dalam *priority queue* (*pq*). Lalu, *priority queue* (*pq*) dicek apakah tidak kosong. Jika tidak kosong, harta karun dengan nilai maksimum (*head*) dijadikan sebagai nilai total maksimum (*maxTotalVal*)dan dihapus dari *priority queue* (*pq*). Program tersebut akan mencetak nilai total maksimum harta karun yang dapat dibawa oleh kereta-kereta kuda tersebut (*maxTotalVal*). Program tersebut memiliki *time complexity* sebesar *O(n log n)* akibat operasi *qsort*  dan *priority queue* yang dilakukan, di mana *n* merupakan jumlah harta karun. Program tersebut juga memiliki *space complexity* sebesar *O(n)* guna menyimpan harta karun dan *priority queue*.

1. *Upsolve the unsolved problems in INC 2023 after the competition!*

Jawaban:

1. *Problem C – Independent Set (Max)*

*Code*:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100000

typedef struct Node {

long long vertex;

struct Node \*next;

} Node;

Node \*createNode(long long v) {

Node \*newNode = (Node\*)malloc(sizeof(Node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

typedef struct Graph {

int numVertices;

Node \*\*adjLists;

} Graph;

Graph \*createGraph(long long vertices) {

Graph \*graph = (Graph\*)malloc(sizeof(Graph));

graph->numVertices = vertices;

graph->adjLists = (Node\*\*)malloc(vertices \* sizeof(Node));

for(long long i = 0; i < vertices; i++) {

graph->adjLists[i] = NULL;

}

return graph;

}

void addEdge(Graph \*graph, long long src, long long dest) {

Node \*newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

}

int dynamicProgramming[MAX\_SIZE][2];

void depthFirstSearch(Graph \*graph, long long curr, long long prev) {

dynamicProgramming[curr][0] = 0;

dynamicProgramming[curr][1] = 1;

Node \*temp = graph->adjLists[curr];

while(temp) {

int next = temp->vertex;

if(next != prev) {

depthFirstSearch(graph, next, curr);

if(dynamicProgramming[next][0] > dynamicProgramming[next][1]) {

dynamicProgramming[curr][0] += dynamicProgramming[next][0];

} else {

dynamicProgramming[curr][0] += dynamicProgramming[next][1];

}

dynamicProgramming[curr][1] += dynamicProgramming[next][0];

}

temp = temp->next;

}

}

int main() {

long long n, u, v;

scanf("%lld", &n);

getchar();

Graph \*graph = createGraph(n+1);

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

scanf("%lld %lld", &u, &v);

getchar();

addEdge(graph, u, v);

addEdge(graph, v, u);

}

if(n == 2) {

puts("0");

} else {

depthFirstSearch(graph, 1, 0);

if(dynamicProgramming[1][0] > dynamicProgramming[1][1]) {

printf("%lld\n", ((n - 1) \* dynamicProgramming[1][0]));

} else {

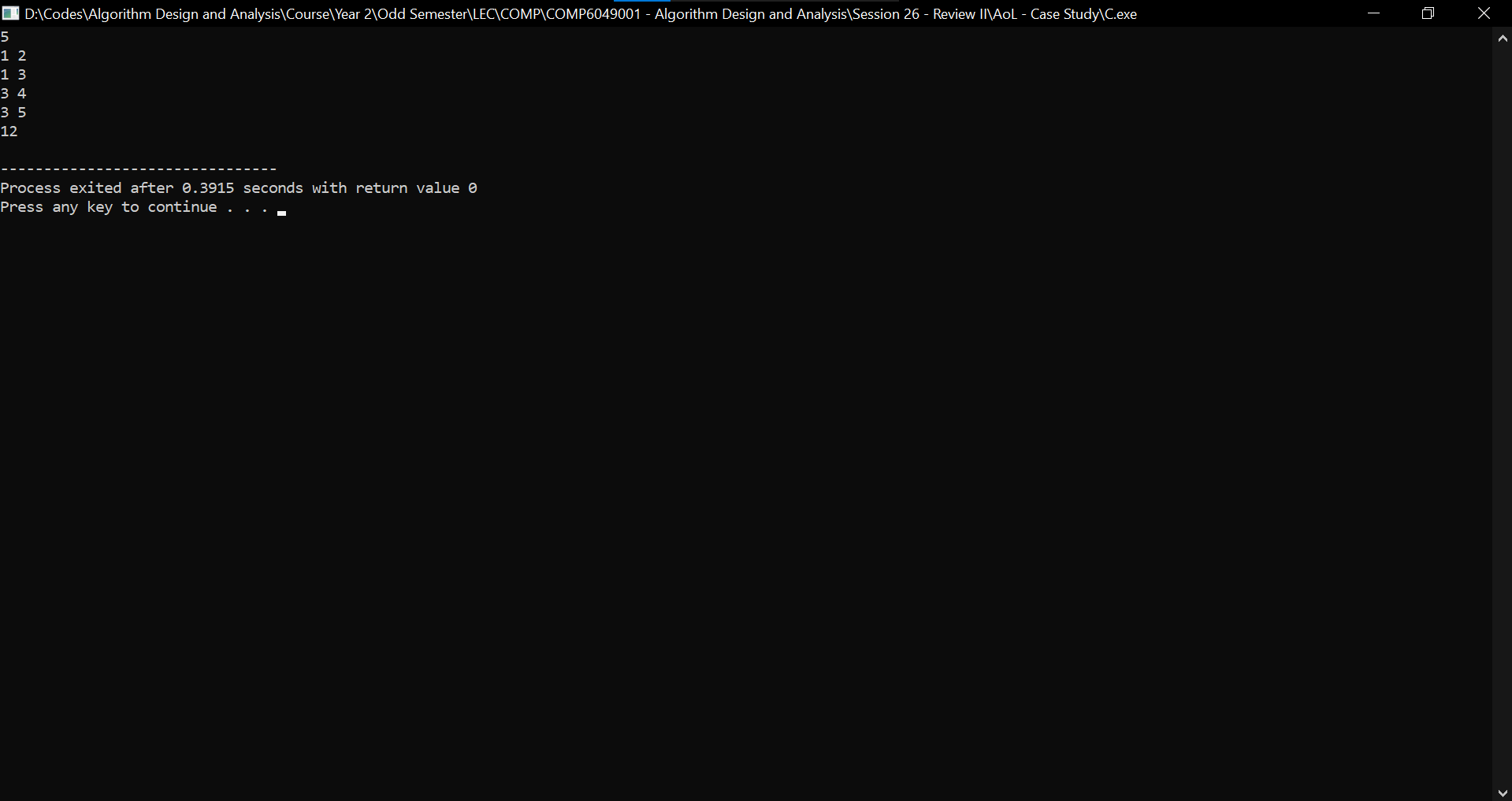
printf("%lld\n", ((n - 1) \* dynamicProgramming[1][1]));

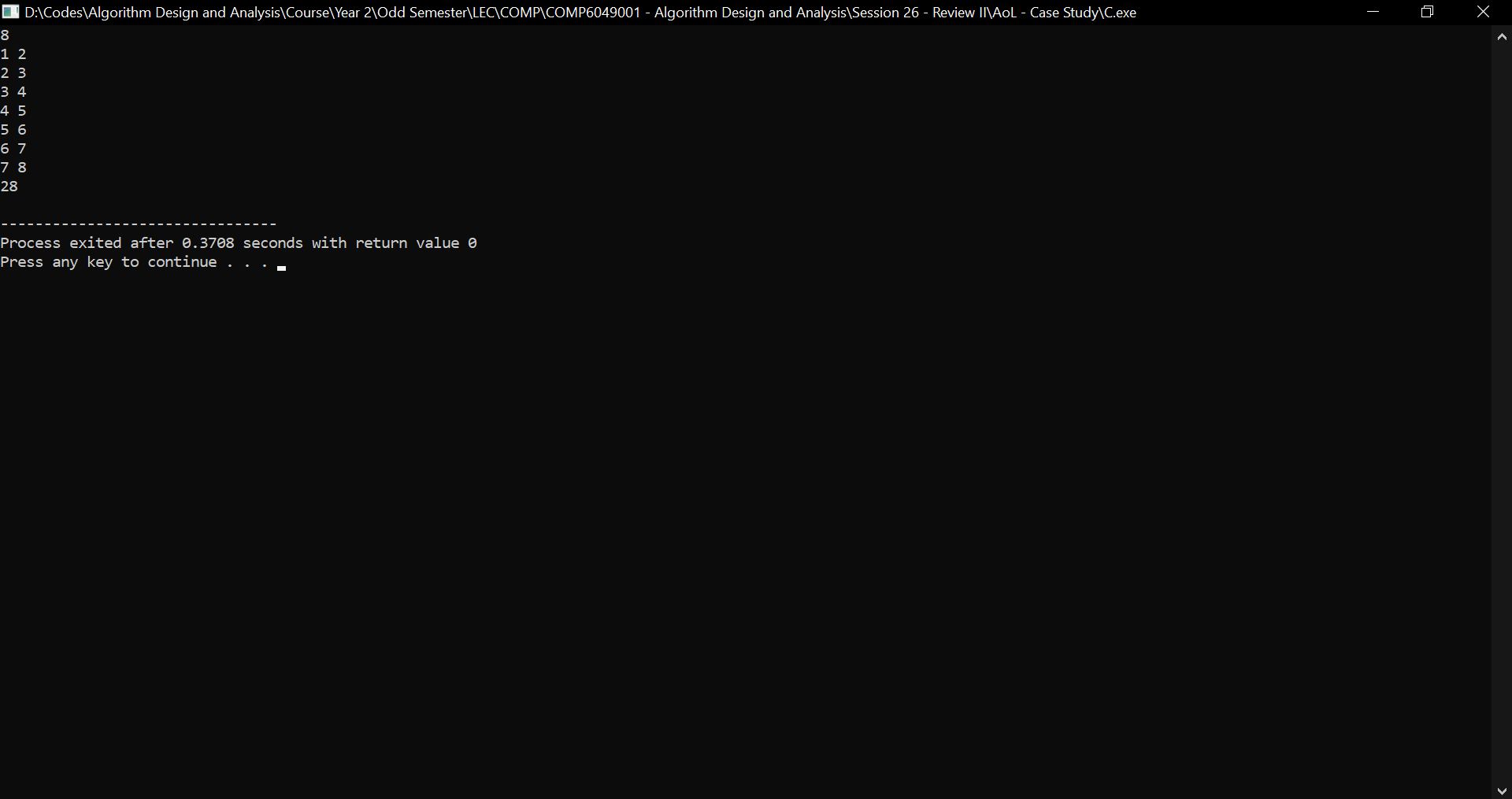
}

}

return 0;

}

*Output #1*:

*Output #2*:

Analisis:

Program di atas pertama-tama membaca jumlah *node* dari *tree* yang diberikan (*n*) dan *n – 1* pasang *node* yang saling terkoneksi (*u* dan *v*). Fungsi *createGraph* dipanggil guna membuat sebuah *graph* baru dengan *vertex* sebanyak *n + 1*. Fungsi addEdge dipanggil guna menambahkan sebuah *edge* ke dalam *graph* tersebut. Struktur *graph* tersebut nantinya akan berisi jumlah *vertex* dan sebuah *array* yang mengandung *adjacency list*. Fungsi *depthFirstSearch* dipanggil guna mengoperasikan *Depth-first Search* (*DFS*) pada *graph* tersebut. Hal tersebut dilakukan dengan menerapkan *dynamicProgramming* pada setiap *node* untuk menentukan skor maksimum jika *node* saat ini (*curr*) termasuk dalam *independent set* dan skor maksimum jika *node* saat ini (*curr*) tidak termasuk dalam *independent set*. Program tersebut akan mencetak skor maksimum yang diperoleh dari *dynamicProgramming* tersebut. Program tersebut memiliki *time complexity* sebesar *O(n)* akibat operasi *Depth-first Search* (*DFS*) yang mengunjungi setiap *node* dan *edge* tepat satu kali. Program tersebut juga memiliki *space complexity* sebesar *O(n)* guna menyimpan *adjacency list* dan tabel *dynamicProgramming*  tersebut.

1. *Problem J – Counting Pairs*

*Code*:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_NUM 200000

long long a[MAX\_NUM], types[1 << 21], activeBits [1 << 21];

int main() {

long long n, mask = 1 << 21;

scanf("%lld", &n);

getchar();

memset(types, 0, sizeof(types));

memset(activeBits, 0, sizeof(activeBits));

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

scanf("%lld", &a[i]);

long long type = 0, j = 1;

while(a[i] > 0) {

if((a[i] % 2) == 1) {

type += j;

}

a[i] /= 4;

j \*= 2;

}

types[type]++;

}

getchar();

for(long long i = 20; i >= 0; i--) {

long long j = 1 << i;

for(long long k = (mask - 1); k >= 0; k--) {

if(!(k & j)) {

types[k] += types[k + j];

}

}

}

for(long long i = 0; i < 21; i++) {

long long j = 1 << i;

for(long long k = 0; k < mask; k++) {

if(k & j){

activeBits[k] = activeBits[k - j] + 1;

}

}

}

long long pairs = ((n \* (n - 1)) / 2);

for(long long i = 1; i < mask; i++) {

if(types[i] > 0){

if(!(activeBits[i] % 2)) {

pairs += ((types[i] \* (types[i] - 1)) / 2);

} else {

pairs -= ((types[i] \* (types[i] - 1)) / 2);

}

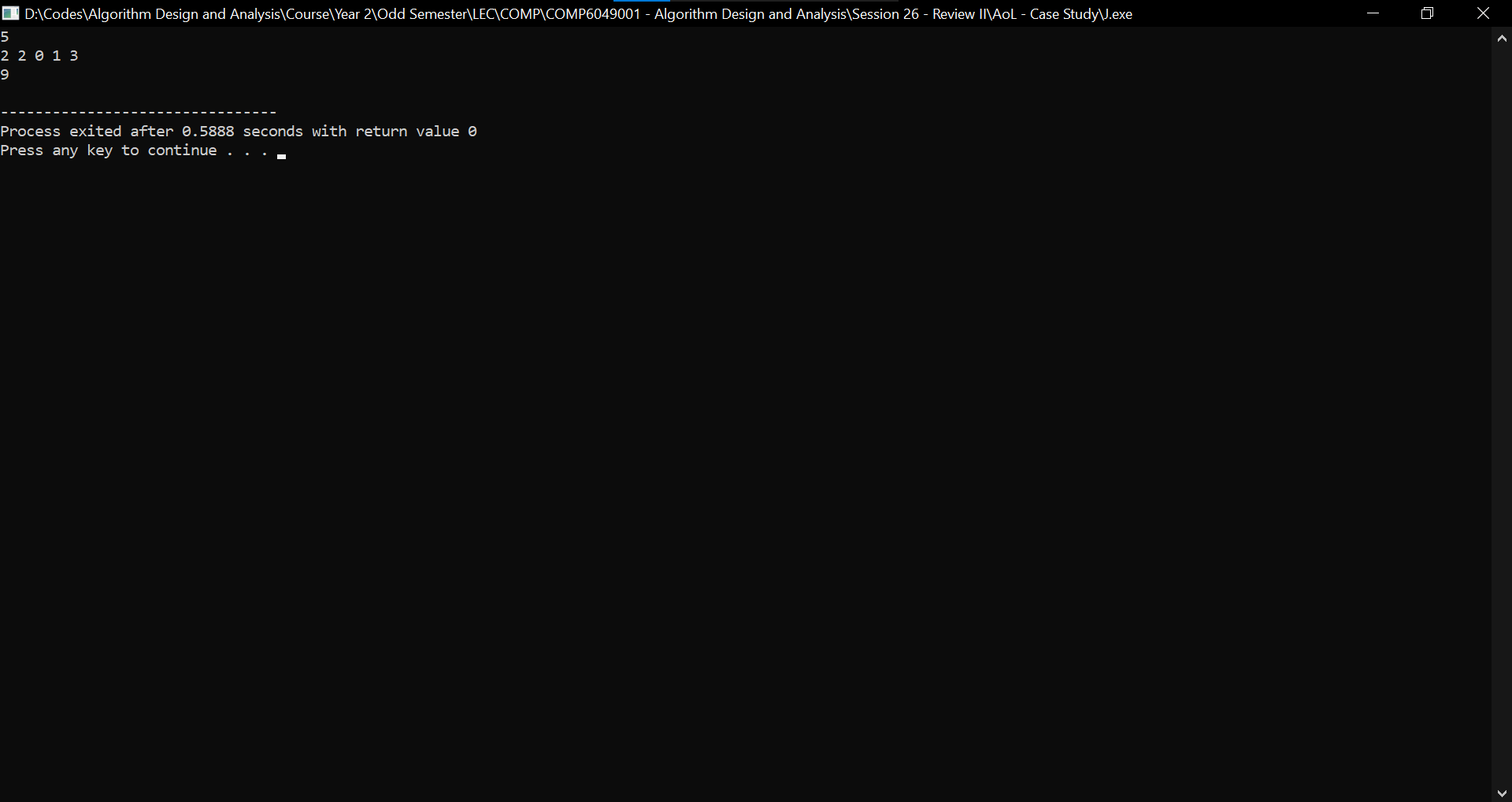
}

}

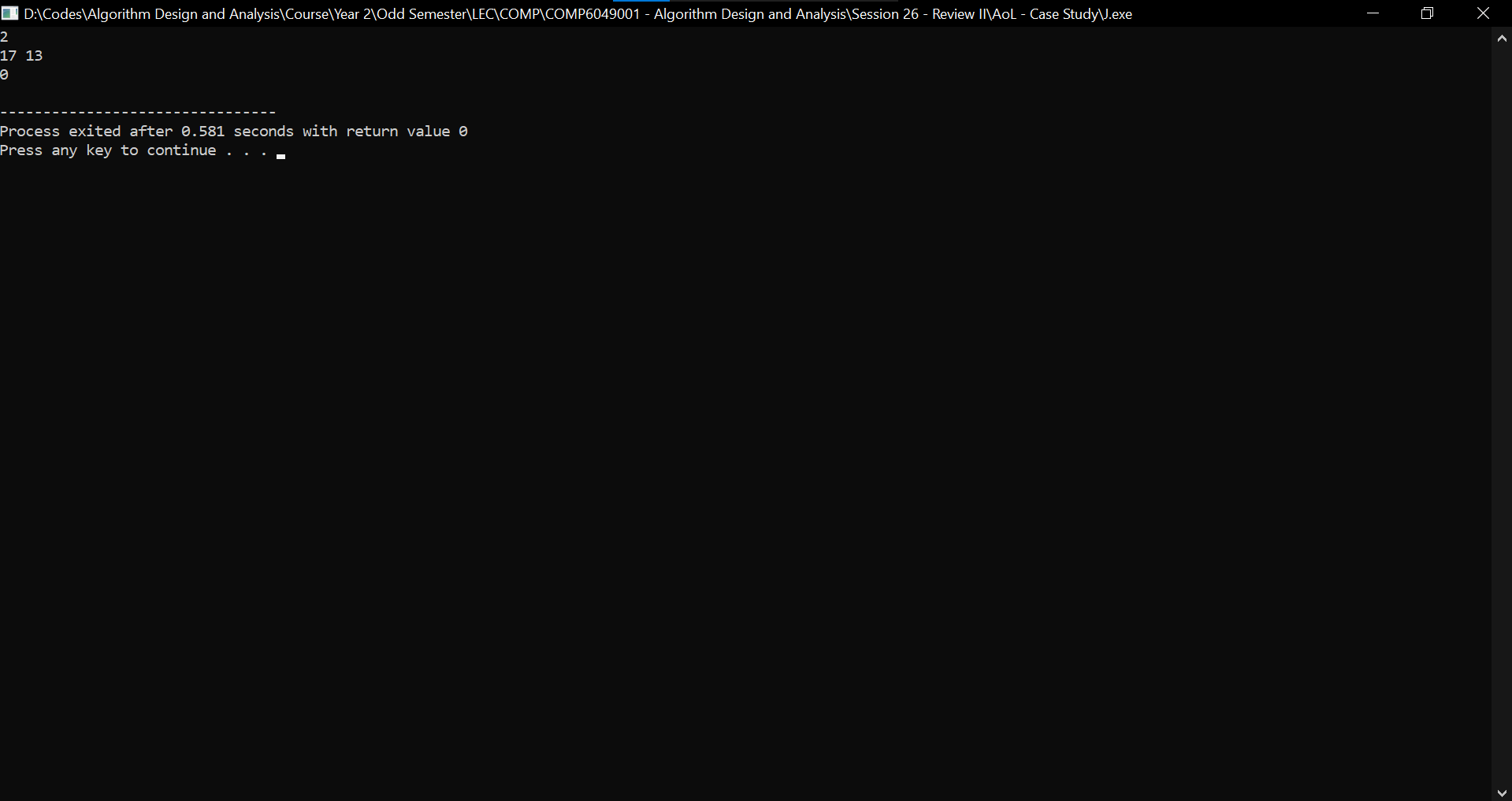
printf("%lld\n", pairs);

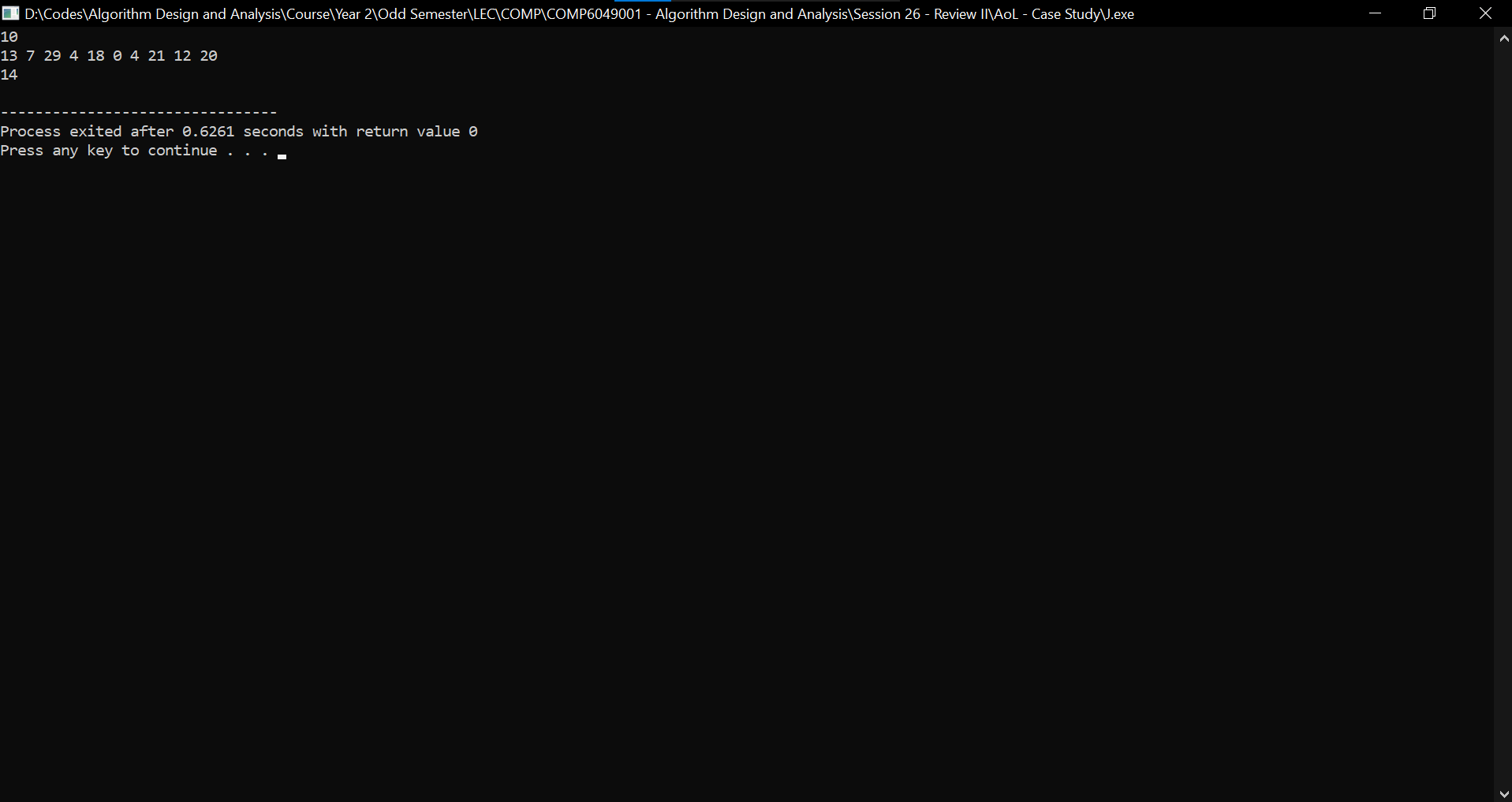
return 0;

}

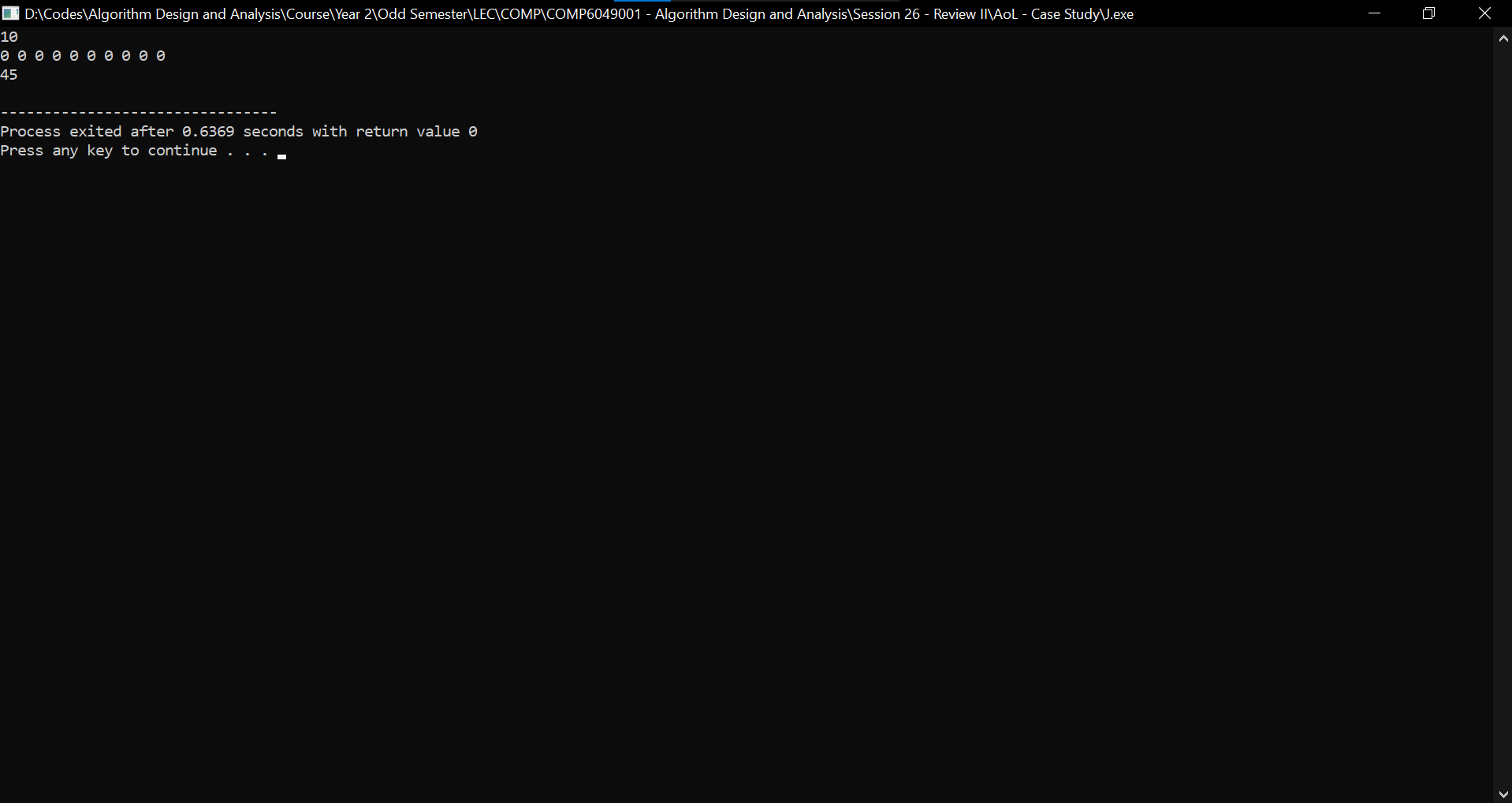
*Output #1*:

*Output #2*:



*Output #3*:

*Output #4*:



Analisis:

Program di atas pertama-tama membaca jumlah bilangan bulat yang diberikan (*n*) dan *n* buah bilangan bulat (*a*). Fungsi *memset* dipanggil guna menginisialisasi daftar representasi biner dari bilangan-bilangan tersebut yang digitnya dikelompokkan berpasangan dalam basis 4 (*types*) dan daftar jumlah bit aktif dalam representasi biner tersebut (*activeBits*) dengan nilai awal 0. Total jenis angka yang dapat dibentuk dari pengelompokan representasi biner tersebut (*mask*) adalah sebanyak 1 << 21, atau yang setara dengan 221. Setiap bilangan bulat yang masuk (*a*) akan dikonversi ke jenisnya (*type*) dan menambahkan entri tipe yang sama (*types*). Setiap tipe yang mungkin (*types*) dilakukan iterasi dan hitungan tipe saat ini ditambahkan ke dalam hitungan tipe yang dihasilkan dengan cara mematikan bit saat ini. Jumlah bit aktif dari setiap jenis yang mungkin (*types*) kemudian akan dicatat dalam *activeBits*. Jumlah total pasangan bilangan (*pairs*) awalnya berjumlah *(n \* (n – 1)) / 2* pasang. Jika suatu tipe memiliki jumlah bit aktif (*activeBits*) genap, jumlah tipe tersebut (*types*) akan ditambahkan ke dalam total pasangan bilangan (*pairs*). Jika tidak, jumlah tipe tersebut (*types*) akan dikurangi dari total pasangan bilangan (*pairs*). Program tersebut kemudian akan mencetak total pasangan bilangan tersebut (*pairs*). Program tersebut memiliki *time complexity* sebesar *O(n log n)* akibat struktur *nested loop* pertama. Program tersebut juga memiliki *space complexity* sebesar *O(n)* guna menyimpan tipe representasi biner yang mungkin, jumlah bit aktif, dan bilangan-bilangan bulat tersebut.

1. *Problem M – Chocolate Bars*

*Code*:

#include <stdio.h>

int main() {

long long n, m, k;

scanf("%lld %lld %lld", &n, &m, &k);

getchar();

if(k == (n \* m)) {

puts("0");

} else if(!(k % n) || !(k % m)) {

puts("1");

} else {

long long i;

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

if(!(k % i) && ((k / i) <= m)) {

break;

}

}

if(i <= n) {

puts("2");

} else {

long long temp = n \* m - k;

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

if(!(temp % i) && ((temp / i) <= m)) {

break;

}

}

if(i <= n) {

puts("2");

} else {

puts("3");

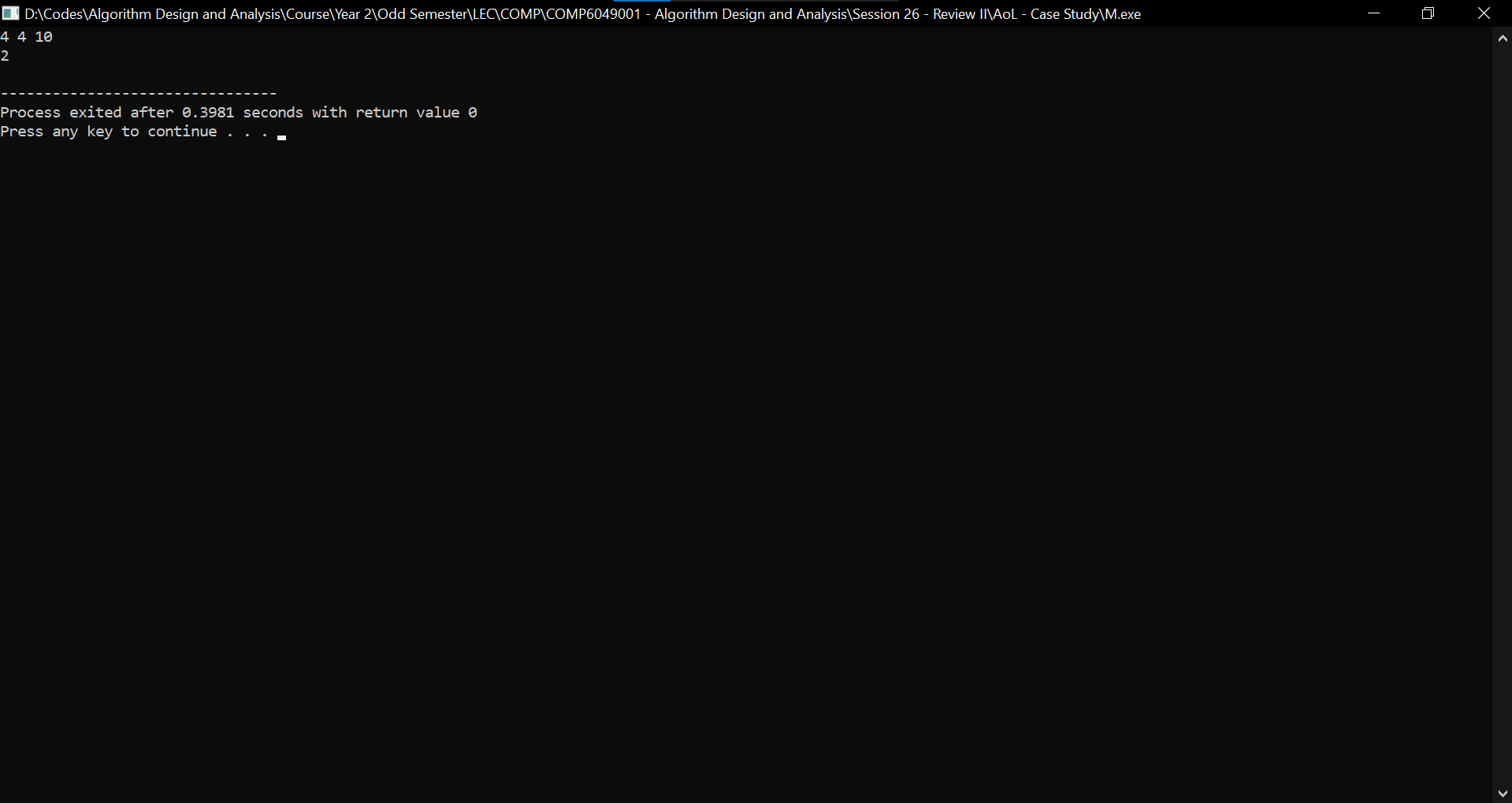
}

}

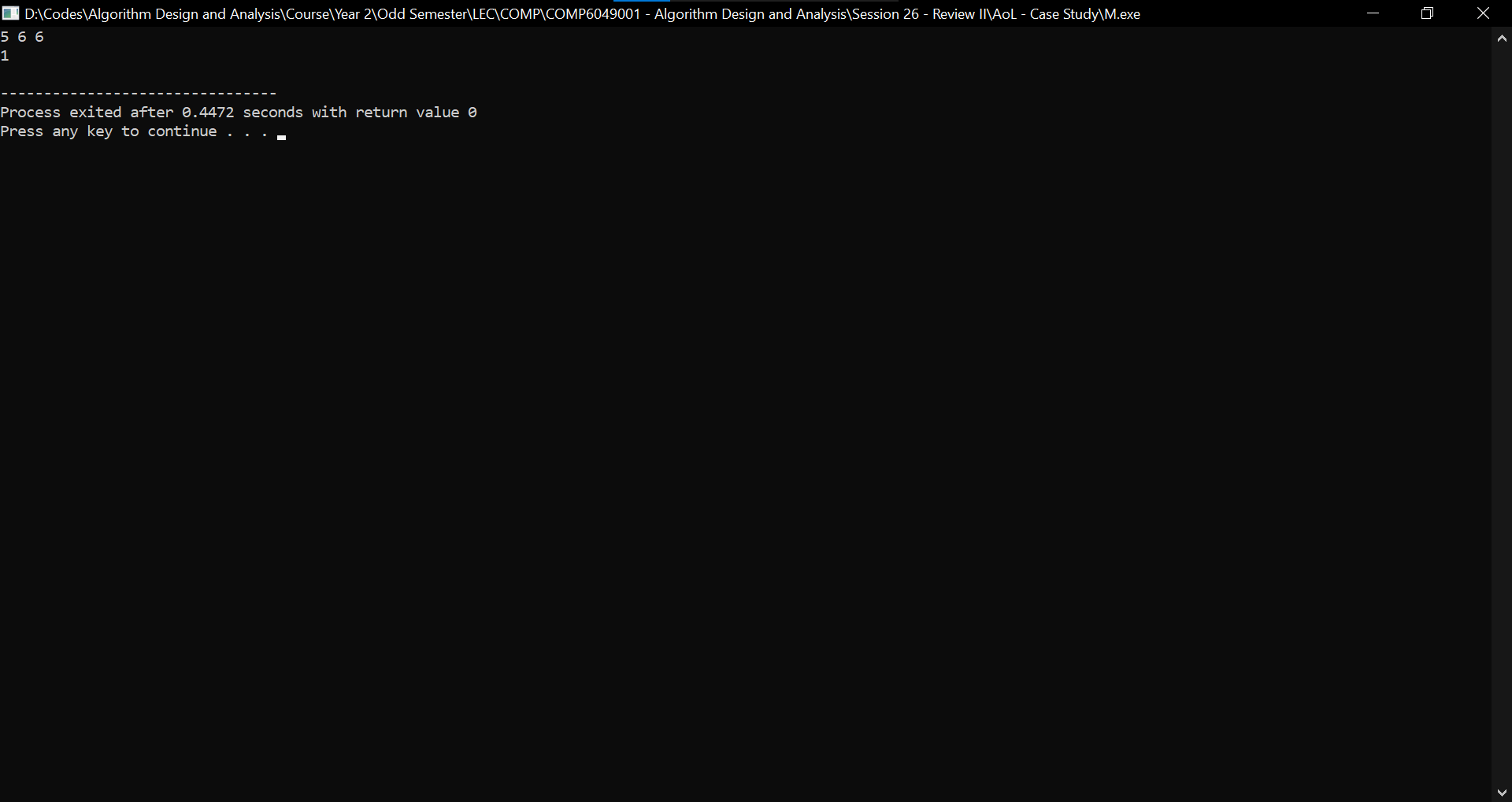
}

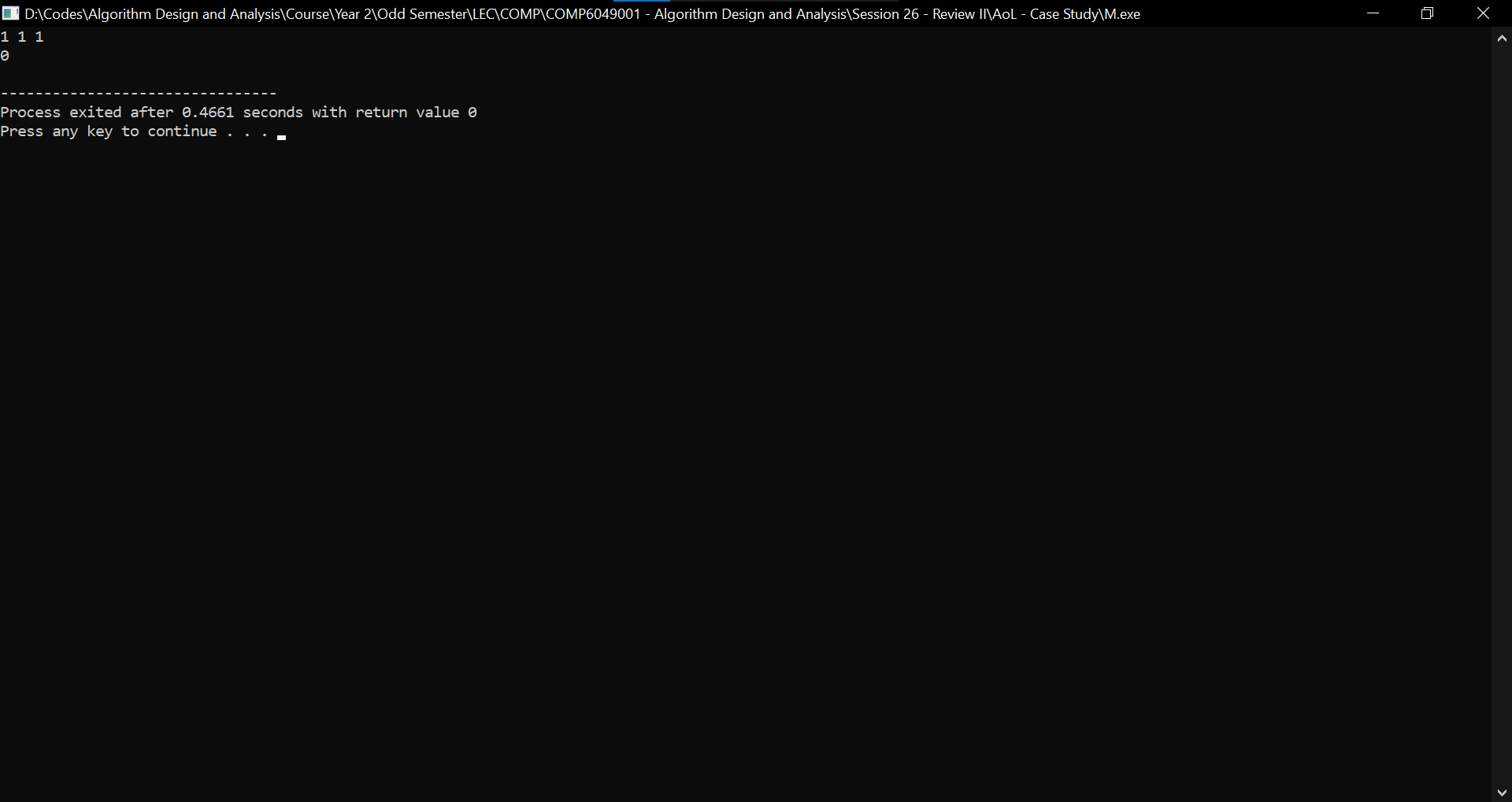
return 0;

}

*Output #1*:

*Output #2*:



*Output #3*:

Analisis:

Program di atas pertama-tama membaca lebar dari cokelat batang yang diberikan (*n*), tinggi dari cokelat batang tersebut (*m*), serta luas total dari cokelat batang yang ingin dimakan (*k*). Program tersebut akan mencetak jumlah operasi minimum sebesar 0 jika luas total dari cokelat batang yang perlu dimakan (*k*) sama dengan total luas yang didapatkan dari hasil perkalian lebar (*n*) dan tinggi (*m*) dari cokelat batang yang diberikan. Program tersebut akan mencetak jumlah operasi minimum sebesar 1 jika luas total dari cokelat batang yang ingin dimakan (*k*) merupakan kelipatan dari lebar (*n*) atau tinggi (*m*) dari cokelat batang yang diberikan. Apabila kedua kondisi tersebut tidak terpenuhi, program tersebut akan mengecek apakah terdapat sebuah bilangan *i* yang lebih kecil dari atau sama dengan lebar dari cokelat batang yang diberikan (*n*) sehingga luas total dari cokelat batang yang ingin dimakan (*k*) merupakan kelipatan dari *i*, dan *k / i* bernilai kurang dari atau sama dengan tinggi dari cokelat batang tersebut (*m*). Apabila bilangan *i* tersebut ada, program akan mencetak jumlah operasi minimum sebesar 2. Apabila bilangan *i* tidak ada, program akan menghitung luas yang tersisa setelah memakan cokelat batang tersebut (*temp*) seluas *k*, dengan persamaan *temp = n \* m – k*. Program tersebut akan mengecek apakah terdapat sebuah bilangan *i*  yang kurang dari atau sama dengan lebar dari cokelat batang yang diberikan (*n*) sehingga luas cokelat yang tersisa merupakan kelipatan dari *i*, serta luas yang tersisa jika dibagi dengan *i* bernilai kurang dari atau sama dengan tinggi dari cokelat batang yang diberikan (*m*). Jika bilangan *i* tersebut ada, program akan mencetak jumlah operasi minimum sebesar 2. Jika bilangan *i* tersebut masih tidak ada juga, program akan mencetak jumlah operasi minimum sebesar 3. Program tersebut memiliki *time complexity* sebesar *O(n)* akibat teknik *brute-force* yang digunakan dalam mengecek seluruh kemungkinan pembagian dari cokelat batang tersebut. Program tersebut juga memiliki *space complexity* sebesar *O(1)* guna menyimpan lebar dari cokelat batang yang diberikan, tingginya, serta luas cokelat yang ingin dimakan.