

## ***AOL - Algorithm Design and Analysis***

### ***Kelompok 9 (Team shailushai)***

Anggota :

1. Ivan Setiawan - 2602109473
  2. Vincent Oei -2602072584
  3. Rivanno H.R. - 2602176774
- 

## **1. Solve problems in INC 2023 during the competition!**

### Certificate:



#### **Certificate of Achievement**



The 2023 ICPC Asia Jakarta - Indonesia National Contest  
04 - 05 November 2023

**Ranked**

**Bina Nusantara University  
shailushai**

Vincent Oei  
Rivanno Haider Rozzaan  
Ivan Setiawan  
Surya Sujarwo, Coach

William B. Poucher, Ph. D.  
ICPC Executive Director



#### **Certificate of Achievement**



The 2023 ICPC Asia Jakarta - Indonesia National Contest  
04 - 05 November 2023

**Ninety-seventh Place**

**Bina Nusantara University  
shailushai**

Vincent Oei  
Rivanno Haider Rozzaan  
Ivan Setiawan  
Surya Sujarwo, Coach

William B. Poucher, Ph. D.  
ICPC Executive Director



#### **Certificate of Achievement**



The 2023 ICPC Asia Jakarta - Indonesia National Contest  
04 - 05 November 2023

**Bina Nusantara University  
shailushai**

Vincent Oei  
Rivanno Haider Rozzaan  
Ivan Setiawan  
Surya Sujarwo, Coach

William B. Poucher, Ph. D.  
ICPC Executive Director

### Solved Problem:

- A. Golden Ticket
- B. Diet Plan
- H. Horse Carts

## 2. Write an analysis of the solutions used in INC 2023!

Analisa : **Algorithm Explanation**, **Time Complexity**, dan **Space Complexity**

### A. Golden Ticket

#### **Source Code:**

```
#include<iostream>
#include<vector>

using namespace std;

int main() {
    int N, M, K;
    cin >> N >> M >> K;

    vector<string> ban;
    vector< pair<string, string> > result;

    for (int i=0; i<N; i++) {
        string name, inst;
        cin >> name >> inst;

        if (i<M) {
            ban.push_back(inst);
        } else {

            if (K>0) {
                bool isValid = 1;
                for (int j=0; j<(int)ban.size(); j++) {
                    if (inst==ban[j]) {
                        isValid = 0;
                        break;
                    }
                }
            }
        }
    }
}
```

```

    }

    if (isValid) {
        result.push_back(make_pair(name, inst));
        ban.push_back(inst);
        K--;
    }
}

}

cout << (int)result.size() << '\n';
for (int i=0; i<(int)result.size(); i++) {
    cout << result[i].first << '\n';
}

return 0;
}

```

### Algorithm Explanation:

Algorithm ini akan meminta nilai N, M, dan K untuk menentukan pemenang Golden Ticket dalam sebuah kompetisi dengan berbagai syarat. N sebagai jumlah data tim yang masuk kedalam kompetisi, M sebagai jumlah top tim yang menjadi pemenang, dan K sebagai top K tim yang institusinya tidak masuk dalam top M tim sebagai pemenang Golden Ticket. Oleh karena itu, dibutuhkan sebuah array result untuk menampung data tim yang mendapat golden ticket dan array ban untuk meng-blacklist institusi yang telah menang sehingga tidak dipilih lagi untuk top K tim.

Algorithm ini bekerja dengan looping sebanyak N kali untuk menganalisa setiap tim. Dalam looping, jika merupakan top M tim, institusi M top tim akan langsung dimasukan dalam vector ban agar tim dari institusi tersebut tidak akan dipilih lagi untuk Golden Ticket. Setelah M top tim sudah dimasukkan, tim sisa akan dianalisis dengan looping sebanyak jumlah data di vector ban dan mengecek apakah institusi tersebut sudah ada di ban tersebut atau tidak. Jika tidak, tim tersebut akan ditambahkan kedalam vector

result dan institusinya akan ditambahkan kedalam vector ban. Boolean isValid untuk mengecek apakah tim tersebut layak/cocok untuk mendapatkan Golden Ticket. Setelah looping utama selesai, akan dilakukan looping lagi sebanyak data dalam vector result dan ditampilkan hasilnya.

Untuk time complexity dan space complexity nya, diasumsikan bahwa M dan K adalah N karena tidak bisa lebih besar dari N sesuai dengan syarat dari soal.

$$1 \leq M, K \leq N \leq 100$$

### Time Complexity Analysis:

Source Code	Cost	Times
<pre> #include&lt;iostream&gt; #include&lt;vector&gt;  using namespace std;  int main() {     int N, M, K;     cin &gt;&gt; N &gt;&gt; M &gt;&gt; K;      vector&lt;string&gt; ban;     vector&lt; pair&lt;string, string&gt; &gt; result;      for (int i=0; i&lt;N; i++) {         string name, inst;         cin &gt;&gt; name &gt;&gt; inst;          if (i&lt;M) {             ban.push_back(inst);         } else {              if (K&gt;0) {                 bool isValid = 1;                 for (int j=0; j&lt;(int)ban.size(); j++) {                     if (inst==ban[j]) { </pre>		
	C1	1
	C2	1
	C3	1
	C4	1
	C5	N+1
	C6	N
	C7	N
	C8	N
	O(1)	1 (agar worst case)
	C9	N-1
	C10	N-1
	C11+O(1)	(N-1) (N-1)
	C12	(N-1) (N-2)

<pre>                 isValid = 0;                 break;             }         }          if (isValid) {             result.push_back(make_pair(name, inst));             ban.push_back(inst);             K--;         }     }      }      cout &lt;&lt; (int)result.size() &lt;&lt; '\n';     for (int i=0; i&lt;(int)result.size(); i++) {         cout &lt;&lt; result[i].first &lt;&lt; '\n';     }      return 0; } </pre>	C13	(N-1) (N-2)
	C14	(N-1) (N-2)
	C15	N-1
	C16+O(1)	N-1
	O(1)	N-1
	C17	N-1
	O(1)	1
	C18	N+1
	C19	N
	C20	1

$T(n) =$

$O(C_1 + C_2 + C_3 + C_4 + C_5 * N + C_5 + C_6 * N + C_7 * N + C_8 * N + O(1) + C_9 * N + C_9 + C_{10} + C_{11} N^2 - C_{11} * 2 * N + C_{11} + O(N^2 - 2N + 1) + C_{12} * N^2 - C_{12} * 3 * N + C_{12} * 2 + C_{13} * N^2 - C_{13} * 3 * N + C_{13} * 2 + C_{14} * N^2 - C_{14} * 3 * N + C_{14} * 2 + C_{15} * N - C_{15} + O(N - 1) + C_{16} * N - C_{16} + O(N - 1) + C_{17} * N - C_{17} + O(1) + C_{18} * N + C_{18} + C_{19} * N + C_{20})$

Mengambil leading value, sehingga worst case:  $O(N^2)$

### Space Complexity Analysis:

Vector ban memiliki maksimal banyak data sebanyak M, dan M memiliki nilai paling maksimum adalah N.

Vector result memiliki maksimal banyak data sebanyak N.

$O(N + N) = O(2N)$ , sehingga worst case:  $O(N)$

## **B. Diet Plan**

### **Source Code:**

```
#include <stdio.h>
#include <string.h>

int max(int total, int* arr, int* arr2){

    int i, temp=-1;

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

        if(!arr2[i] && (temp == -1 || arr[i] > arr[temp])){
            temp = i;
        }

    }

    return temp;
}

int main(){

    int n, m, k, i, check, sum_milk=0, count=0;

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

    int arr[n+1], arr2[n+1];
    memset(arr2, 0, sizeof(arr2));

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

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

        sum_milk += arr[i];
```

```

while(k>0 && sum_milk>m){

    check = max(i, arr, arr2);

    if(check != -1){

        sum_milk -= arr[check];
        arr2[check]=1;
        k--;
    }
    else{
        break;
    }
}

if(sum_milk>m){
    break;
}

count++;
}

printf("%d", count);

return 0;
}

```

### Algorithm Explanation :

```

#include <stdio.h>
#include <string.h>

```

Penyelesaian soal B ini diselesaikan dengan library stdio.h untuk input dan output, serta library string.h untuk manipulasi string.

Ada 2 function yang digunakan dalam kasus ini, yang pertama tentu saja 'main' sebagai function wajib, dan yang kedua adalah function 'max'. Pertama-tama, mari kita bahas function 'main' secara mendetail terlebih dahulu. Berikut adalah ulasannya :

```
int n, m, k, i, check, sum_milk=0, count=0;

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

Isi function dimulai dengan mendeklarasikan n, m, k, i, check, sum\_milk, dan count sebagai integer. Integer n mewakili jumlah hari dalam rencana diet, m mewakili jumlah mL susu yang tersedia, k mewakili jumlah biskuit yang tersedia, i adalah variabel yang nantinya digunakan untuk looping, check mewakili nilai maksimum antara arr dan arr2 (akan dibahas lebih lanjut pada algoritma check di bawah), sum\_milk mewakili total mL susu dari rencana diet tersebut (itulah mengapa dideklarasikan = 0 saat awal karena akan menghitung totalnya), dan count mewakili jumlah hari maksimum dalam mempertahankan rencana diet (dideklarasikan = 0 karena nantinya digunakan untuk menghitung jumlahnya). Lalu, akan di ambil inputan untuk n, m, dan k, kemudian digunakan getchar() untuk mengambil ‘\n’ dari input buffer.

```
int arr[n+1], arr2[n+1];
memset(arr2, 0, sizeof(arr2));
```

Dalam bagian ini, kita mendeklarasikan arr[n+1] dan arr2[n+1] sebagai integer array. arr[n+1] ini nantinya digunakan untuk menyimpan jumlah mL susu yang perlu dikonsumsi per hari. Di sini digunakan n+1 dan bukan n karena nanti ada looping yang dimulai dari i=1. Jadi, karena array itu 0-based index, n nya ditambah 1 karena index 0 tidak dipakai. Kemudian, array arr2[n+1] digunakan untuk mengisi suatu blok memori dengan 0 terlebih dahulu dengan memset, yang nantinya akan digunakan sebagai tempat sementara dimana nilainya akan dibandingkan dengan arr pada function ‘max’. Berbicara mengenai memset, memset tersebut akan mengisi arr2 dari starting address-nya arr2 dengan nilai 0, sebanyak jumlah byte dari sizeof(arr2).

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

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

    sum_milk += arr[i];

    while(k>0 && sum_milk>m){

        check = max(i, arr, arr2);

        if(check != -1){
```



```

        sum_milk -= arr[check];
        arr2[check]=1;
        k--;
    }
    else{
        break;
    }
}

if(sum_milk>m){
    break;
}

count++;
}

printf("%d", count);

return 0;
}

```

Setelah itu, kita masuk ke dalam looping for. Looping ini dimulai dari  $i=1$  sampai  $i \leq n$  (dimulai dari index 1 karena sebagai penunjuk hari pertama agar lebih mudah divisualisasikan). Di dalam looping tersebut, dimulai dengan mengambil inputan untuk arr, yakni inputan jumlah mL susu pada hari ke-i (dimulai dari hari pertama sampai n). Kemudian, jumlah mL susu pada hari tersebut akan ditambahkan ke sum\_milk.

Lebih lanjut, ada looping while yang berfungsi untuk mengecek apakah biskuit perlu dimakan pada hari tersebut atau tidak. Untuk lebih jelasnya lagi, while loop tersebut pertama-tama akan mengecek apakah kondisinya sudah sesuai, yakni jumlah biskuit ( $k > 0$ ) dan  $\text{sum\_milk} > \text{total mL susu yang tersedia (m)}$ . Jika ya, maka akan dicek nilai maksimum antara elemen di arr and arr2 menggunakan function 'max' (function akan dijelaskan lebih lanjut di bawah). Jika  $\text{check} == -1$ , maka artinya kita tidak perlu memakan biskuit karena return value  $\text{check} == -1$  menandakan bahwa jumlah mL susu pada hari tersebut bukan maksimum di antara hari-hari lainnya yang sudah dicek oleh looping for. Jadi, langsung di break saja jika  $\text{check} == -1$ . Namun jika  $\text{check} != -1$ , maka artinya kita perlu memakan biskuit sehingga sum\_milk akan dikurangkan dengan jumlah mL susu pada hari yang memiliki mL susu

terbanyak, kemudian arr2 dengan index check akan dijadikan = 1 (untuk kebutuhan looping pada function 'max' nanti untuk menunjukkan bahwa arr2 pada index tertentu bukan 0, serta jumlah biskuit (k) akan dikurangi dengan 1 karena dikonsumsi.

Selanjutnya, jika sum\_milk > jumlah mL susu yang tersedia (m), artinya pada hari tersebut rencana diet berhenti karena stok susu sudah tidak mencukupi dan biskuit sudah habis, sehingga langsung break keluar dari for loop dan tidak perlu menambah count lagi. Jika sudah keluar dari for loop, maka artinya jumlah maksimum hari untuk mempertahankan rencana diet sudah didapatkan yakni count, dan count akan di print untuk dijadikan output.

Untuk pembahasan function 'max', berikut adalah penjelasannya,

```
int max(int total, int* arr, int* arr2){  
  
    int i, temp=-1;  
  
    for(i=1; i<=total; i++){  
  
        if(!arr2[i] && (temp == -1 || arr[i] > arr[temp])){  
            temp = i;  
        }  
    }  
  
    return temp;  
}
```

Jadi, function 'max' ini membutuhkan 3 parameter dari 'main', yakni i (total hari rencana diet sudah berjalan) dinamai total, arr, serta arr2. Dalam function tersebut, ditambahkan variabel i untuk looping dan variabel temp = -1 (-1 sebagai penanda saat return bahwa tidak ada hari yang memiliki jumlah mL susu terbanyak karena kondisi if dalam for loop tidak benar). Lalu, akan dijalankan for looping dari hari pertama sampai hari ke-total, untuk cek jumlah mL susu maksimum dan nanti hasil akhirnya adalah index (hari) yang akan disimpan ke dalam temp.

Setelah membahas mengenai algoritmanya, berikut adalah analisa terkait time complexity dan space complexity dari algoritma tersebut :

## Time Complexity Analysis:

Di sini analisisnya dibagi menjadi 2 bagian terlebih dahulu baru digabung, yakni analisis function 'max' dan function 'main'. Berikut adalah analisis untuk function 'max' :

```
int i, temp=-1;
```

Statement di atas hanyalah initialization dengan time complexity-nya constant sehingga cost = 1 (C1).

```
for(i=1; i<=total; i++){  
    if(!arr2[i] && (temp == -1 || arr[i] > arr[temp])){  
        temp = i;  
    }  
}
```

Untuk looping for, i nya akan mengiterasi dari 1 sampai total sehingga cost untuk time complexity-nya sebanyak total (C2). Kemudian untuk if di dalam looping tersebut, dalam worst case, kondisi if ini dapat true sebanyak total kali, artinya constant terhadap banyaknya iterasi for (bergantung terhadap banyaknya total), sehingga artinya cost-nya bisa ditulis sebagai 1 (C3). Lalu, statement di dalam if-nya memiliki cost = 1 karena merupakan assignment. Dengan demikian,

$$T(n) = C1(1) + C2(\text{total}) + C3(1) + C4(1)$$

$$T(n) = (C2)\text{total} + (C1+C3+C4)$$

Misalkan,

$$a = (C2)$$

$$b = (C1+C3+C4)$$

Maka,

$$T(n) = a*\text{total} + b$$

Kita ambil leading term, yakni total, sehingga,

$T(n) = O(\text{total})$

Worst Case :  $O(\text{total}) = O(n)$

Setelah mengulas function 'max', selanjutnya adalah function 'main', yakni sebagai berikut :

```
int n, m, k, i, check, sum_milk=0, count=0;

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

Statement-statement di atas yakni initialization, mengambil inputan dengan scanf, serta read newline dengan getchar() memiliki time yang constant sehingga cost = 1 (C1, C2, C3).

```
int arr[n+1], arr2[n+1];
memset(arr2, 0, sizeof(arr2));
```

Statement di atas yang merupakan initialization memiliki cost = 1 (C4), sedangkan untuk memset sendiri dia memiliki linear time complexity karena tergantung jumlah byte yang ditetapkan dalam memori sehingga cost-nya sebanyak n (C5).

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

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

    sum_milk += arr[i];

    while(k>0 && sum_milk>m){

        check = max(i, arr, arr2);

        if(check != -1){

            sum_milk -= arr[check];
            arr2[check]=1;
            k--;
        }
    }
}
```

```

        }
        else{
            break;
        }
    }

    if(sum_milk>m){
        break;
    }

    count++;
}

printf("%d", count);

```

Untuk looping for, worst case nya adalah saat for loop tidak break saat kondisi  $\text{if}(\text{sum\_milk} > m)$  bernilai false terus menerus, sehingga  $i$  nya akan mengiterasi dari 1 sampai  $n$  sehingga cost untuk time complexity-nya sebanyak  $n$  (C6). Kemudian, dilanjut dengan mengambil inputan dengan `scanf` sehingga  $\text{cost} = 1$  (C7), operasi penjumlahan sehingga  $\text{cost} = 1$  (C8), lalu ada while loop. While loop ini berisikan pemanggilan function 'max' yang memiliki time complexity  $O(n)$ , sehingga bisa dikatakan bahwa ada sebanyak  $n*n$  kali (C9) karena nested loops. Kemudian, dilanjut pengecekan kondisi `if else`, yang mana kedua kondisi ini bernilai constant dengan  $\text{cost} = 1$  karena bergantung terhadap while itu sendiri, sehingga keseluruhan tersebut kita rangkum menjadi satu saja menjadi C10. Lalu sehabis while loop, dilanjut dengan `if statement` sehingga  $\text{cost}$ -nya juga 1 (C11) karena bergantung terhadap for, dan di dalamnya ada `break statement` yang juga constant sehingga  $\text{cost} = 1$  (C12). Dan terakhir pada looping, ada increment count yang bersifat constant juga sehingga  $\text{cost}$ -nya juga 1 (C13). Setelah keluar dari loop, ada `printf` yang juga constant sehingga  $\text{cost} = 1$  (C14). Dengan demikian,

$$T(n) = C1(1) + C2(1) + C3(1) + C4(1) + C5(n) + C6(n) + C7(1) + C8(1) + C9(n^2) + C10(1) + C11(1) + C12(1) + C13(1) + C14(1)$$

$$T(n) = (C9)n^2 + (C5+C6)n + (C1+C2+C3+C4+C7+C8+C10+C11+C12+C13+C14)$$

Misalkan,

$$a = (C9)$$

$$b = (C5+C6)$$

$$c = (C1+C2+C3+C4+C7+C8+C10+C11+C12+C13+C14)$$

Maka,

$$T(n) = a*n^2 + b*n + c$$

Kita ambil leading term, yakni total, sehingga,

$$T(n) = O(n^2)$$

Worst Case :  $O(n^2)$

Sehingga, time complexity untuk problem B “Diet Plan” adalah  **$O(n^2)$** .

### **Space Complexity Analysis:**

Pada kode di atas, bisa dilihat bahwa ada 2 array yang dibuat yakni `arr[n+1]` dan `arr2[n+1]`. Hal ini mengambil space sebanyak  $O(n)$  karena array tersebut akan menyimpan elemen-elemen sebanyak  $n$  (bukan  $n+1$  karena itu ditambah satu akibat index iterasi akan dimulai dari 1 dan bukan 0). Atau dengan kata lain, space nya akan bertambah secara linier terhadap ukuran inputan  $n$ .

Kemudian, space yang digunakan oleh variabel-variabel lainnya seperti  $n$ ,  $m$ ,  $k$ , `sum_milk`, `dst` itu memiliki time yang constant.

Sehingga, space complexity secara keseluruhan adalah  **$O(n)$**  dikarenakan `arr` dan `arr2`.

## H. Horse Carts

Source code:

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>

using namespace std;

bool sortCompare(const pair<int,int> &a, const pair<int,int> &b) {
    if (a.second < b.second) {
        return true;
    } else {
        return false;
    }

    return (a.first > b.first);
}

int main() {

    int N, M;
    cin >> N >> M;

    pair<int, int> treasure[N];
```

```
for (int i=0; i < N; i++) {
    int w, v;
    cin >> w >> v;

    treasure[i] = make_pair(v, w);
}

sort(treasure, treasure + N, sortCompare);

int s[M];
for (int i=0; i < M; i++) {
    cin >> s[i];
}

sort(s, s+M);

long long int sum = 0;
priority_queue< pair<int, int> > pq;

int j = 0;
for (int i = 0; i < M; i++) {
    while (j < N && treasure[j].second <= s[i]) {
        pq.push(treasure[j]);
        j++;
    }
}
```



```

        if (!pq.empty()) {
            sum += (pq.top()).first;
            pq.pop();
        }
    }

    cout << sum << '\n';

    return 0;
}

```

### Algorithm Explanation:

```

#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>

```

Penyelesaian kasus H menggunakan 2 library tambahan yaitu queue dan algorithm. Library queue digunakan untuk membuat container berupa priority queue, sedangkan algorithm digunakan untuk memanggil function sort.

```

int N, M;
cin >> N >> M;

pair<int, int> treasure[N];
for (int i=0; i < N; i++) {
    int w, v;
    cin >> w >> v;
}

```

```
    treasure[i] = make_pair(v, w);  
}
```

Pada bagian awal function main di bentuk dua integer N dan M untuk menampung jumlah treasures dan horse carts. Lalu dibuat array dengan ukuran tetap sebesar N yaitu jumlah treasure. Array ini adalah array container pair yang berisi 2 integer, dengan integer awal value dari treasure dan integer kedua weight. Deklarasi tersebut memakan waktu. Lalu menggunakan for loop sebanyak N-treasure untuk memasukan value dan weight ke dalam array container pair treasure. Proses for loop ini dijalankan sebanyak N kali sehingga memakan waktu  $O(N)$

```
sort(treasure, treasure + N, sortCompare);
```

Bagian ini menggunakan built in function dari C++ yaitu `sort()`, yang menerima 3 parameter. Parameter pertama adalah alamat pertama array, parameter kedua alamat terakhir, dan parameter ketiga adalah pembanding. Disini dibuat sebuah fungsi custom `sortCompare()` yang menampung dua alamat container. Fungsi ini digerakan oleh fungsi built in `sort()` untuk menyusun array treasure.

```
bool sortCompare(const pair<int,int> &a, const pair<int,int> &b) {  
    if (a.second < b.second) {  
        return true;  
    } else {  
        return false;  
    }  
  
    return (a.first > b.first);  
}
```

Secara rinci, fungsi sort compare membandingkan data kedua dari container pair, dalam kasus ini adalah array treasure yang data keduanya adalah weight dari treasure tersebut. Fungsi custom ini digunakan oleh fungsi built in `sort()` untuk mengurutkan data di dalam array treasure dari weight paling ringan ke paling berat atau secara ascending. Fungsi `sort()` sendiri memiliki time complexity

**$O(N \log N)$**  untuk average dan worst case. Kesamaan untuk average dan worst case ini dikarenakan fungsi `sort()` menggunakan algoritma hibrida introsort.

```
int s[M];
for (int i=0; i < M; i++) {
    cin >> s[i];
}

sort(s, s+M);
```

Bagian berikut ini menginisialisasi sebuah array sebesar M-horse carts, lalu memasukan nilai kapasitas weight dari setiap horse cart melalui for loop yang berjalan sepanjang  $O(M)$ . Setelah nilai kapasitas dimasukan, fungsi built in `sort()` digunakan untuk mengurutkan horse cart dari yang memiliki kapasitas weight paling kecil ke kapasitas weight paling besar atau secara ascending. Fungsi `sort()` sendiri memiliki time complexity  $O(N \log N)$  untuk best, average, dan worst case.

```
long long int sum = 0;
priority_queue< pair<int, int> > pq;
```

Bagian berikut ini mendeklarasikan variabel `sum` untuk menampung value maximum treasure yang dapat diambil. Lalu mendeklarasikan sebuah container berupa priority queue yang digunakan untuk mengatasi treasure dengan weight yang sama tetapi value yang berbeda. Priority queue digunakan untuk mengambil treasure dengan value terbesar.

```
int j = 0;
for (int i = 0; i < M; i++) {
    while (j < N && treasure[j].second <= s[i]) {
        pq.push(treasure[j]);
        j++;
    }
}
```

```
        if (!pq.empty()) {  
            sum += (pq.top()).first;  
            pq.pop();  
        }  
    }  
}
```

Bagian ini mendeklarasikan variabel *j* untuk mengiterasi array *treasure* sedangkan for loop *i* untuk mengiterasi array *s* yang merepresentasikan horse carts. Di dalam for loop terdapat while loop yang mengiterasi array *treasure* sebesar *N*, apabila weight dari *treasure* mampu ditampung oleh horse cart index ke *i* maka *treasure* tersebut akan dimasukan ke dalam priority queue *pq*, lalu mengiterasi ke array *treasure* index berikutnya. Lalu data teratas di dalam priority queue akan dijumlahkan ke dalam *sum* sebelum menghapus data paling atas dalam priority queue melalui fungsi built in *pop()*.

Mekanisme priority queue ini mengambil *treasure* dengan value terbesar walaupun memiliki weight ringan. Seperti pada sample input #4 berikut:

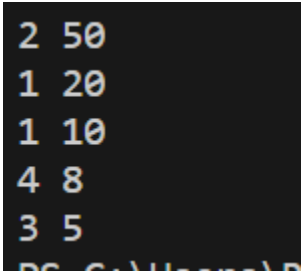
#### Sample Input #4

```
7 4  
1 10  
1 20  
2 50  
3 5  
4 8  
10 100  
12 40  
2 2 5 7
```

Pada sample input tersebut walaupun horse cart index ke 2 memiliki kapasitas 5 yang akan diambil adalah treasure dengan weight 1 dan value 10. Hal ini dikarenakan berdasarkan urutan priority queue yang dapat kita cek nilainya dengan snippet ini

```
while (!pq.empty()){
    cout << (pq.top()).second << " " << (pq.top()).first << endl;
    pq.pop();
}
```

Maka akan berurutan seperti ini:

	<p>Berdasarkan urutan disamping maka horse cart pertama berkapasitas 2 akan mengambil treasure dengan weight 2 bernilai 50. Horse cart kedua berkapasitas 2 mengambil treasure dengan weight 1 bernilai 20. Horse cart ketiga berkapasitas 5 mengambil treasure dengan <b>weight 1 bernilai 10</b>. Horse cart keempat berkapasitas 7 mengambil treasure dengan <b>weight 4 bernilai 8</b>. Sehingga memiliki total value yang sesuai dengan sample output #4</p> <p><b>Sample Output #4</b></p> <div data-bbox="579 829 1885 883" style="border: 1px solid black; padding: 2px;">88</div>
---	--

Tanpa priority queue terdapat resiko menambahkan treasure duplikat atau treasure dengan weight terkecil tapi value yang tidak maksimal.

### Time Complexity Analysis:

Source Code	Cost	Times
<pre>#include &lt;iostream&gt; #include &lt;vector&gt; #include &lt;queue&gt;</pre>		

```
#include <algorithm>
```

```
using namespace std;
```

```
bool sortCompare(const pair<int,int> &a, const pair<int,int> &b) {
```

```
    if (a.second < b.second) {
```

```
        return true;
```

```
    } else {
```

```
        return false;
```

```
    }
```

```
    return (a.first > b.first);
```

```
}
```

```
int main() {
```

```
    int N, M;
```

```
    cin >> N >> M;
```

```
    pair<int, int> treasure[N];
```

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

```
        int w, v;
```

```
        cin >> w >> v;
```

```
        treasure[i] = make_pair(v, w);
```

```
    }
```

O(1)

1

C1

1

C2

1

C3

1

C4

O(N+1)

<pre> sort(treasure, treasure + N, sortCompare);  int s[M]; for (int i=0; i &lt; M; i++) {     cin &gt;&gt; s[i]; }  sort(s, s+M);  long long int sum = 0; priority_queue&lt; pair&lt;int, int&gt; &gt; pq;  int j = 0; for (int i = 0; i &lt; M; i++) {     while (j &lt; N &amp;&amp; treasure[j].second &lt;= s[i]) {         pq.push(treasure[j]);         j++;     }      if (!pq.empty()) {         sum += (pq.top()).first;         pq.pop();     }  }  cout &lt;&lt; sum &lt;&lt; '\n'; </pre>	C5	O(N logN)
	C6	1
	C7	O(N)
	C8	O(N logN)
	C9	1
	C10	1
	C11	1
	C12	O(N)
	C13	O(N <sup>2</sup> )
	C14	1
	C15	1

<pre>return 0; }</pre>		
------------------------	--	--

$$\begin{aligned}
 T(n) &= C1 + C2 + C3 + C4 + C5 + C6 + C7 + C8 + C9 + C10 + C11 + C12 + C13 + C14 \\
 &= 1 + 1 + 1 + O(N+1) + O(N \log N) + 1 + O(N) + O(N \log N) + 1 + 1 + 1 + O(N) + O(N^2) + 1 + 1 \\
 &= O(N^2) + 2*O(N \log N) + 3*O(N) + 10
 \end{aligned}$$

Mengambil leading term sehingga,

$$T(n) = O(N^2) = \mathbf{O(n^2)}$$

### Space Complexity Analysis:

Penggunaan space terbesar ada di dalam dynamic data structure priority queue.

```
for (int i = 0; i < M; i++) {
    while (j < N && treasure[j].second <= s[i]) {
        pq.push(treasure[j]);
        j++;
    }

    if (!pq.empty()) {
        sum += (pq.top()).first;
        pq.pop();
    }
}
```

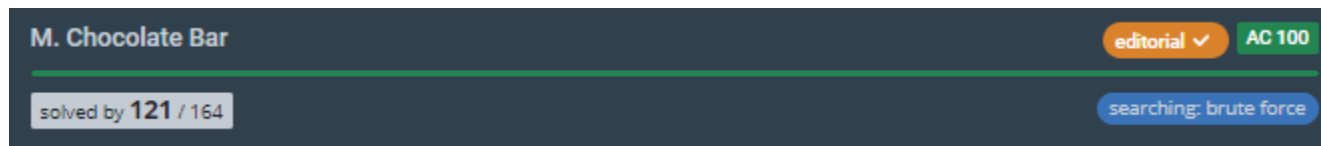
Pada kodingan diatas worst case yang dapat terjadi ketika semua treasure dapat dimasukkan ke dalam semua horse cart yang dimasukkan melalui `pq.push(treasure[j])`. Sehingga ukuran terbesar priority queue pq adalah sebesar  **$O(M*N)$** .



### 3. Upsolve the unsolved problems in INC 2023 after the competition!

#### M. Chocolate Bar

Problem ini sulit karena kelompok kami tidak kepikiran untuk membagi permasalahan yang besar menjadi permasalahan yang kecil terlebih dahulu. Kelompok kami juga tidak terpikirkan bahwa setiap area coklat dapat didapatkan dengan 3 potongan atau kurang. Kami juga terlanjur mengira jika menggunakan teknik bruteforce, akan memakan terlalu banyak waktu dan mengakibatkan algoritma terkena time limit. Selain itu, faktor waktu dan tekanan membuat kelompok kami tidak dapat menyelesaikan permasalahan ini.



#### Source Code:

```
#include<iostream>

using namespace std;

int main() {
    long long int N, M, K;
    cin >> N >> M >> K;

    int cut = 3;

    if (N*M==K) {
        cut = 0;
    }
}
```

```

    } else if (K%N==0 || K%M==0) {
        cut = 1;
    } else {
        for (long long int i=1; i<N; i++) {
            if (K<=M*i && K%i==0){
                cut=2;
                break;
            }
        }

        K = N*M-K;
        for (long long int i=1; i<M; i++) {
            if (K<=N*i && K%i==0){
                cut=2;
                break;
            }
        }
    }

    cout << cut << '\n';

    return 0;
}

```

### Algorithm Explanation:

```
#include<iostream>
```

Penyelesaian kasus M menggunakan library iostream untuk mengatur input dan output dari program.

```
using namespace std;
```

Namespace std digunakan untuk mempersingkat pemanggilan fungsi dalam standard library iostream, sehingga tidak perlu menulis std saat memanggil fungsi cin dan cout.

```
long long int N, M, K;  
cin >> N >> M >> K;
```

Selanjutnya, variabel-variabel yang dibutuhkan diinisialisasi, yakni N (lebar dari coklat), M (tinggi dari coklat), serta K (total area yang bisa dimakan). Lalu, ada pengambilan inputan untuk N, M, dan K dengan cin.

```
int cut = 3;
```

Kemudian, cut di-assign = 3. Nilai cut ditetapkan sebagai 3 terlebih dahulu karena berdasarkan analisis kami, 3 adalah jumlah maksimum coklat tersebut bisa dipotong-potong untuk mendapatkan nilai K yang diharapkan. Hal ini didasari oleh ketentuan bahwa coklat bisa dibagi 2 dengan rumus antara  $(n*i)$  dan  $(n*(m-i))$  dengan  $1 \leq i < m$ , atau  $(i*m)$  dan  $((n-i)*m)$  dengan  $1 \leq n < i$ .

```
if (N*M==K) {  
    cut = 0;  
} else if (K%N==0 || K%M==0) {  
    cut = 1;  
} else {  
    for (long long int i=1; i<N; i++) {  
        if (K<=M*i && K%i==0){  
            cut = 2;  
            break;  
        }  
    }  
}
```

```

    }

    K = N*M-K;
    for (long long int i=1; i<M; i++) {
        if (K<=N*i && K%i==0){
            cut = 2;
            break;
        }
    }
}
}

```

Setelah cut di-assign ke 3, sekarang masuk ke kondisi if else. Kondisi if yang pertama adalah jika  $N*M == K$ , maka  $cut = 0$ . Di sini, maksudnya adalah karena area  $N*M$  sudah memenuhi nilai  $K$ , maka artinya tidak perlu lagi memotong coklatnya sehingga  $cut = 0$ . Namun jika kondisi if tersebut tidak terpenuhi, maka akan lanjut memeriksa ke kondisi else if  $K\%N == 0 \parallel K\%M == 0$ . Maksud dari kondisi ini adalah jika nilai  $K$  habis dibagi  $N$ , atau nilai  $K$  habis dibagi  $M$ , maka  $cut = 1$ . Kalau habis dibagi, berarti artinya ada bilangan pengali untuk mendapatkan nilai  $K$ . Jadi, cukup dipotong sekali saja karena sisi lainnya nanti selain bilangan pengali tersebut, bisa dikondisikan sesuai dengan nilai  $K$ . Selanjutnya, jika kondisi else if ini tidak terpenuhi, maka dilanjutkan ke kondisi else. Di dalam else ini, ada 2 looping for, yakni yang pertama yang di loop adalah sampai  $N-1$ , sedangkan yang kedua yang di loop sampai  $M-1$ . Pada looping pertama, ada kondisi if  $K \leq M*i \text{ \&\& } K\%i == 0$  dan jika terpenuhi, maka  $cut = 2$ . Maksud dari kondisi ini adalah  $M*i$  adalah ukuran coklat yang bisa terbentuk jika  $K\%i == 0$  terpenuhi. Kondisi  $K\%i == 0$  harus terpenuhi karena untuk memastikan bahwa  $i$  nya valid untuk menjadi bilangan pengali dari  $M$  tersebut, kurang lebih konsepnya mirip dengan else if di atas. Lalu jika kondisi if tersebut tidak terpenuhi, maka kita akan masuk ke looping kedua. Namun sebelum masuk, ada statement  $K = N*M - K$ . Statement untuk mencari selisih antara  $N*M$  dengan  $K$  yang lama agar menemukan  $K$  yang baru yang kemudian digunakan untuk looping kedua, dimana konsep looping kedua sama seperti looping pertama, hanya saja bilangan untuk dikali berbanding terbalik dengan looping pertama.

```
cout << cut << '\n';
```

Terakhir, cut akan di print menjadi hasil output. Jika kondisi-kondisi if else tersebut tidak terpenuhi, maka berartinya cut nya tetap sama dengan 3.

### Time Complexity Analysis:

Source Code	Cost	Times
<pre> #include&lt;iostream&gt;  using namespace std;  int main() {     long long int N, M, K;     cin &gt;&gt; N &gt;&gt; M &gt;&gt; K;      int cut = 3;      if (N*M==K) {         cut = 0;     } else if (K%N==0    K%M==0) {         cut = 1;     } else {         for (long long int i=1; i&lt;N; i++) {             if (K&lt;=M*i&amp;&amp;K%i==0){                 cut=2;                 break;             }         }          K = N*M-K;         for (long long int i=1; i&lt;M; i++) {             if (K&lt;=N*i&amp;&amp;K%i==0){                 cut=2;                 break;             }         }     } </pre>		
	C1	1
	C2	1
	C4	1
	C5	1
	C6	1
	C7	1
	C8	1
	C9	N
	C10	N-1
	C11	1
	C12	1
	C13	1
	C14	M
	C15	M-1
	C16	1
	C17	1

<pre>         }     }      cout &lt;&lt; cut &lt;&lt; '\n';      return 0; } </pre>	C18	1
---	-----	---

$T(n) = O(C1+C2+C3+C4+C5+C6+C7+C8+C9*N+C10*N-C10+C11+C12+C13+C14*M+C15*M-C15+C16+C17+C18)$

$T(n) = O(N+M)$

Mengambil leading value, sehingga worst case:  **$O(N+M)$**

### Space Complexity Analysis:

Karena hanya memerlukan variabel sebanyak constant angka dan tidak menggunakan array sebanyak n-variabel (variabel dinamis), oleh karena itu hanya diperlukan  $O(4)$  untuk 4 variabel yang telah diinisialisasi, yaitu N, M, K, dan cut, sehingga worst case:  **$O(1)$**