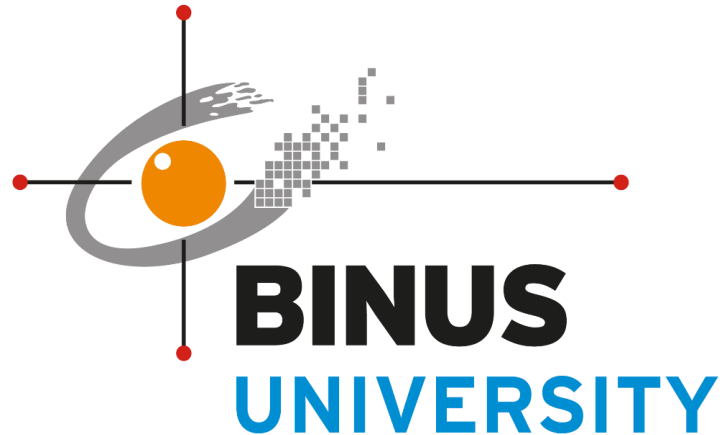


ASSURANCE OF LEARNING
ALGORITHM DESIGN AND ANALYSIS



Kelas LF01

DISUSUN OLEH:

KELOMPOK 4 (HOLY)

2602168734 Benedictus Kenneth Lukito

2602052802 Kevin














2602054764 Bryan Mulia

COMPUTER SCIENCE
BINA NUSANTARA UNIVERSITY
JAKARTA
2023

1. Solve problems in INC 2023 during the competition!

Final Standings

[Scoreboard inc-2023 \(binus.ac.id\)](https://binus.ac.id)

DOMjudge Scoreboard Problemset Login contest over														
RANK	TEAM	SCORE	A	B	C	D	E	F	G	H	I	J	K	L
97	 radeon Bina Nusantara University	3 564	35 1 try	210 5 tries						239 1 try		11 tries		
98	 PT.HashtableIndonesia Bina Nusantara University	3 568	147 2 tries	127 3 tries						214 2 tries		9 tries		
99	 IdiotsOfTheRoundTable Bina Nusantara University	3 571	93 1 try	167 5 tries			1 try			231 1 try				
100	 YOGA Bina Nusantara University	3 583	68 2 tries	160 7 tries						175 3 tries				
101	 Holy Bina Nusantara University	3 584	88 1 try	41 1 try						275 10 tries				
96	 Pils Ijo Bina Nusantara University	3 554	91 3 tries	159 1 try						184 5 tries		1 try		
97	 radeon Bina Nusantara University	3 564	35 1 try	210 5 tries						239 1 try		11 tries		
98	 PT.HashtableIndonesia Bina Nusantara University	3 568	147 2 tries	127 3 tries						214 2 tries		9 tries		
99	 IdiotsOfTheRoundTable Bina Nusantara University	3 571	93 1 try	167 5 tries			1 try			231 1 try				
100	 YOGA Bina Nusantara University	3 583	68 2 tries	160 7 tries						175 3 tries				
101	 Holy Bina Nusantara University	3 584	88 1 try	41 1 try						275 10 tries				
102	 TanamUbi Bina Nusantara University	3 589	55 1 try	143 4 tries			1 try			171 8 tries		4 tries		
103	 heavenlyrestrictions Bina Nusantara University	3 590	75 1 try	272 3 tries						203 1 try				

Sertifikat Bukti Tidak Terdiskualifikasi



Certificate of Achievement

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

Ninety-seventh Place

Bina Nusantara University
Holy

Benedictus Lukito
Bryan Mulla
Kevin Kevin

Kanyadian Idananta, Coach



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

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

Problem A

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

int main(void)
{
    int team = 0, top = 0, ticket = 0;
    scanf("%d %d %d", &team, &top, &ticket);
    char nama[101][11];
    char univ[11] = "";
    char univ_lolos[101][11];
    int lolos = 0;
    int banyak = 0;
    int index[101] = {0};
    for (int i = 0; i < team; i++)
    {
        scanf("%s %s", nama[i], univ);
        if (top > 0)
        {
            top--;
            strcpy(univ_lolos[lolos], univ);
            lolos++;
        }
        else
        {
            if (ticket > 0)
            {
                int ada = 0;
                for (int j = 0; j < lolos; j++)
                {
```

```

        if (strcmp(univ, univ_lolos[j]) == 0)
        {
            ada = 1;
        }
    }
    if (ada != 1)
    {
        index[banyak] = i;
        strcpy(univ_lolos[lolos], univ);
        lolos++;
        banyak++;
        ticket--;
    }
}

}

printf("%d\n", banyak);
for (int i = 0; i < banyak; i++)
{
    printf("%s\n", nama[index[i]]);
}
}

```

A.cpp

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main(void)
5  {
6      int team = 0, top = 0, ticket = 0;
7      scanf("%d %d %d", &team, &top, &ticket);
8      char nama[101][11];
9      char univ[11] = "";
10     char univ_lolos[101][11];
11     int lolos = 0;
12     int banyak = 0;
13     int index[101] = {0};
14     for (int i = 0; i < team; i++)
15     {
16         scanf("%s %s", nama[i], univ);
17         if (top > 0)
18         {
19             top--;
20             strcpy(univ_lolos[lolos], univ);
21             lolos++;
22         }
23         else
24         {
25             if (ticket > 0)
26             {
27                 int ada = 0;
28                 for (int j = 0; j < lolos; j++)
29                 {
30                     if (strcmp(univ, univ_lolos[j]) == 0)
31                     {
32                         ada = 1;
33                     }
34                 }
35                 if (ada != 1)
36                 {
37                     index[banyak] = i;
38                     strcpy(univ_lolos[lolos], univ);
39                     lolos++;
40                     banyak++;
41                     ticket--;
42                 }
43             }
44         }
45     }
46     printf("%d\n", banyak);
47     for (int i = 0; i < banyak; i++)
48     {
49         printf("%s\n", nama[index[i]]);
50     }
51 }
```

Analisa permasalahan

Dari soal tersebut diminta jumlah berapa input universitas dan nama timnya, top berapa tim yang lolos, dan berapa wildcard yang akan diambil diluar dari tim top yang ditentukan, pada saat kita

akan mengambil wildcard persyaratannya adalah universitas dengan tim tidak masuk ke dalam top yang ditentukan dan diambil maksimal 1 peringkat teratas di bawah top yang ditentukan dari universitas yang sama. Kemudian, akan ditampilkan jumlah yang akan bisa menjadi wildcard dengan menampilkan nama tim dari universitas yang mendapatkan tiket wildcard.

Cara Penyelesaian secara Algoritma

Kita menerima input dari user jumlah dari timnya, top berapa yang akan lolos, dan tiket wildcardnya kemudian kita akan melakukan looping sebanyak jumlah team untuk menerima nama tim dan universitasnya yang disimpan ke dalam array dan melakukan pengecekan terhadap kondisi yang ada. Kondisi yang pertama kita cek apakah kuota yang lolos masih tersedia atau tidak jika iya maka masukkan ke dalam array universitas yang lolos. Kemudian jika kuota lolos advance langsung tanpa wildcard sudah habis, kita lakukan pengecekan jika tiket wildcard masih tersedia maka kita akan melakukan looping untuk mengecek apakah universitas dengan peringkat di luar top secara ascending dengan yang sudah lolos apakah bernilai true/ tidak atau dengan kata lain sudah lolos apa belum, jika sudah lulus maka kita tandai dengan memberikan variabel ada dengan nilai 1. Kemudian kita cek, apakah variabel ada bernilai 1 atau tidak jika tidak maka masukkan value dari indeks tim yang akan lolos dengan variabel `index[banyak]` yang bertumbuh seiring dengan banyaknya wildcard yang memungkinkan, kemudian kita masukkan ke dalam penandaan universitas yang lolos kemudian mengurangi jumlah tiket wildcard yang tersedia. Kita akhiri looping dan kita cetak banyaknya wild card yang tergunakan dan cetak tim mana saja yang lolos dengan mengakses penandaan indeks yang ada.

Analisa Kompleksitas Waktu dan Ruang dari Algoritma

- Kompleksitas ruang

Masing-masing inisialisasi variabel memiliki nilai kompleksitas ruang sebanyak $O(1)$. pengecualian ada pada variabel nama yang memiliki kompleksitas yaitu $O(101 \times 11) = O(1111)$ dan univ lolos dengan jumlah kompleksitas yang sama yaitu $O(1111)$, kemudian untuk index memiliki kompleksitas $O(101)$. Jika dijumlahkan keseluruhannya maka sama dengan kompleksitas konstan yaitu $O(1)$. Perbedaan terlihat pada looping dengan kompleksitas yang berbeda dimana bergantung pada jumlah team sehingga

kompleksitasnya adalah $O(\text{team})$, kompleksitas looping yang bergantung pada variabel lolos dengan kompleksitas $O(\text{lolos})$, dan kompleksitas looping untuk mencetak yang bergantung pada variabel banyak dengan kompleksitas $O(\text{banyak})$ sehingga jumlah dari kompleksitas ruang yang ada adalah $O(\text{team} + \text{lolos} + \text{banyak} + 1) = O(3n + 1) = O(n)$.

- Kompleksitas waktu

Setiap baris dari selain dari looping yang ada memiliki kompleksitas $O(1)$ utamanya baris 6 hingga 13 dan baris 46. Pada baris 14, utamanya dimulainya looping berdasarkan value team akan memiliki kompleksitas waktu yaitu $O(\text{team})$ dan pada baris 28 terdapat looping di dalamnya lagi yang bergantung pada value lolos sehingga kompleksitasnya adalah $O(\text{team} * \text{lolos})$ dikarenakan pertumbuhannya secara kuadratik. Pada baris ke 47 terdapat looping yang bergantung pada value banyak dengan pertumbuhannya secara linear sehingga kompleksitas waktunya adalah $O(\text{banyak})$. Sehingga total dari kompleksitas waktunya adalah $O(\text{team} * \text{lolos} + \text{banyak}) = O(n * n + n) = O(n^2)$

Problem B

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

```
int main(void)
```

```
{
```

```
    int banyak = 0, susu = 0, biskuit = 0;
```

```
    scanf("%d %d %d", &banyak, &susu, &biskuit);
```

```
    int diet[banyak + 1] = {0};
```

```
    for (int i = 0; i < banyak; i++)
```

```
    {
```

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

```
    }
```

```
    int i = 0;
```

```
    for (i = 0; i < banyak; i++)
```

```
    {
```

```
        int max = 0;
```

```

        int index = 0;
        susu -= diet[i];
        if (susu >= 0)
        {

        }
        else if (biskuit > 0)
        {
            biskuit--;
            for (int j = 0; j <= i; j++)
            {
                if (max < diet[j])
                {
                    max = diet[j];
                    index = j;
                }
            }
            susu += max;
            diet[index] = 0;
        }
        else
        {
            printf("%d\n", i);
            break;
        }
    }

    if (i == banyak)
    {
        printf("%d\n", banyak);
    }

```


}

```
A.cpp B.cpp
1 #include <stdio.h>
2
3 int main(void)
4 {
5     int banyak = 0, susu = 0, biskuit = 0;
6     scanf("%d %d %d", &banyak, &susu, &biskuit);
7     int diet[banyak + 1] = {0};
8     for (int i = 0; i < banyak; i++)
9     {
10         scanf("%d", &diet[i]);
11     }
12     int i = 0;
13     for (i = 0; i < banyak; i++)
14     {
15         int max = 0;
16         int index = 0;
17         susu -= diet[i];
18         if (susu >= 0)
19         {
20             }
21         else if (biskuit > 0)
22         {
23             biskuit--;
24             for (int j = 0; j <= i; j++)
25             {
26                 if (max < diet[j])
27                 {
28                     max = diet[j];
29                     index = j;
30                 }
31             }
32             susu += max;
33             diet[index] = 0;
34         }
35         else
36         {
37             {
38                 printf("%d\n", i);
39                 break;
40             }
41         }
42     }
43     if (i == banyak)
44     {
45         printf("%d\n", banyak);
46     }
```

Analisa Permasalahan

Pada soal B, kita diminta untuk membuat sebuah program diet dengan ketentuan sebagai berikut : N sebagai rencana hari kita melakukan diet, M sebagai jumlah susu yang kita punya dan K sebagai jumlah biskuit yang kita punya. Dalam sehari, kita perlu meminum susu sebanyak P1(jumlah susu yang dikonsumsi dalam sehari) atau kita dapat mengganti susu dengan biskuit dalam

program diet kita. Fungsi program yang kita buat adalah mengetahui berapa hari diet kita dapat berjalan Ketika kita mempunyai jumlah susu N , Biscuit K dengan konsumsi susu per hari $P1$.

Cara Penyelesaian secara Algoritma

Pada bagian input kita memasukan N sebagai rencana hari kita akan melakukan diet, lalu kita menginput M sebagai jumlah susu dan K sebagai jumlah biscuit. Inti dari algoritma ini adalah terus mencari sampai mana diet bisa dilakukan dengan memakan susu pada hari yang membutuhkan biscuit paling banyak. Sehingga algoritma ini melakukan looping dari hari pertama dan memasukan ke variabel berisi max number setiap hari kita akan memakan biscuit sesuai dengan kebutuhan. Sampai hari di mana biscuit habis yaitu $\text{biscuit} \leq 0$ maka susu akan diminum sehingga hari dengan biscuit terbanyak bisa dipakaikan susu. Sehingga bisa dipastikan bahwa susu digunakan secara optimal pada hari yang paling membutuhkan susu tersebut. Program akan melakukan loop dari hari ke-0 hingga hari ke- $N-1$. Di setiap iterasi, program akan mengecek kebutuhan kalori untuk hari tersebut. Program akan melakukan loop nested untuk mencari elemen maksimum di array diet yang dimulai dari index 0 hingga index saat ini (i). Untuk masalah output jika susu dan biscuit habis maka akan di print di hari mana mereka habis. Jika sampai hari terakhir tidak habis maka akan memprint total hari awal.

Analisa Kompleksitas Waktu dan Ruang dari Algoritma

- Kompleksitas Ruang

Terdapat variabel banyak, variabel susu dan variabel biscuit yang masing-masing memiliki nilai $O(1)$ sehingga menjadi $O(1) + O(1) + O(1) = O(3)$. Terdapat juga array diet yang dipengaruhi oleh variabel banyak, $\text{diet}[\text{banyak} + 1] = \{0\}$ dimana ukurannya menjadi $O(N)$. Sehingga kompleksitas ruang menjadi $O(3) + O(N)$ yang mana karena ukuran $O(N)$ jauh lebih signifikan daripada $O(3)$ sehingga $O(3)$ dapat diabaikan dan kita dapatkan kompleksitas dari program ini adalah sebesar $O(N)$.

- Kompleksitas Waktu

Dalam program ini terdapat pengulangan iterasi. di dalam loop for ($i = 0$; $i < \text{banyak}$; $i++$), terdapat nested loop yang mencari elemen maksimum di dalam array diet yang

dimulai dari index 0 hingga index saat ini (i). Dengan demikian, kompleksitas waktu dari loop nested adalah $O(N)$. $O(N)$ ini dimasukan ke dalam kompleksitas waktu dari loop utama yang memiliki kompleksitas waktu $O(N)$. Oleh karena itu, kompleksitas waktu total dari kode tersebut adalah $O(N^2)$ sebab $O(N) * O(N)$ menjadi $O(N^2)$. Semua hal tersebut terjadi akibat adanya dua nested loop atau loop dalam loop dalam pembuatan algoritma kami.

Problem H

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

```
#include <stdlib.h>
```

```
#define MAX 100000
```

```
struct Item {  
    int berat;  
    int harta;  
};
```

```
int compareForQuickSort(const void *a, const void *b) {  
    return ((struct Item *)a)->berat - ((struct Item *)b)->berat;  
}
```

```
int main(void) {  
    int berat, harta;  
    scanf("%d %d", &berat, &harta);  
    struct Item items[MAX];  
    for (int i = 0; i < berat; i++) {  
        scanf("%d %d", &items[i].berat, &items[i].harta);  
    }  
    int kapasitas[MAX];
```

```

for (int i = 0; i < harta; i++) {
    scanf("%d", &kapasitas[i]);
}
qsort(items, berat, sizeof(struct Item), compareForQuickSort);
qsort(kapasitas, harta, sizeof(int), compareForQuickSort);

struct Item maxHeap[MAX];
int maxHeapSize = 0;
int curr = 0;
long long total = 0;
int index = 0;
while (index < harta) {
    while (curr < berat && items[curr].berat <= kapasitas[index]) {
        maxHeap[maxHeapSize] = items[curr];
        int i = maxHeapSize;
        while (i > 0) {
            int j = (i-1) / 2;
            if (maxHeap[j].harta >= items[curr].harta){
                break;
            }
            maxHeap[i] = maxHeap[j];
            i=j;
        }
        maxHeap[i] = items[curr];
        maxHeapSize++;
        curr++;
    }
    if (maxHeapSize > 0) {
        total += maxHeap[0].harta;
        maxHeapSize--;
        maxHeap[0] = maxHeap[maxHeapSize];
    }
}

```

```

int i = 0;

while (i*2+1<maxHeapSize) {
    int left = i * 2 + 1;
    int right = i * 2 + 2;
    int j = left;
    if (right < maxHeapSize && maxHeap[right].harta > maxHeap[left].harta){
        j = right;
    }
    if (maxHeap[j].harta <= maxHeap[maxHeapSize].harta){
        break;
    }
    maxHeap[i]=maxHeap[j];
    i = j;
}
maxHeap[i]=maxHeap[maxHeapSize];
} index++;
}
printf("%lld\n", total);
return 0;
}

```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAX 100000
5
6  struct Item {
7      int berat;
8      int harta;
9  };
10
11 int compareForQuickSort(const void *a, const void *b) {
12     return ((struct Item *)a)->berat - ((struct Item *)b)->berat;
13 }
14
15 int main(void) {
16     int berat, harta;
17     scanf("%d %d", &berat, &harta);
18     struct Item items[MAX];
19     for (int i = 0; i < berat; i++) {
20         scanf("%d %d", &items[i].berat, &items[i].harta);
21     }
22     int kapasitas[MAX];
23     for (int i = 0; i < harta; i++) {
24         scanf("%d", &kapasitas[i]);
25     }
26     qsort(items, berat, sizeof(struct Item), compareForQuickSort);
27     qsort(kapasitas, harta, sizeof(int), compareForQuickSort);
28
29     struct Item maxHeap[MAX];
30     int maxHeapSize = 0;
31     int curr = 0;
32     long long total = 0;
33     int index = 0;
34     while (index < harta) {
35         while (curr < berat && items[curr].berat <= kapasitas[index]) {
36             maxHeap[maxHeapSize] = items[curr];
37             int i = maxHeapSize;
38
39             while (i > 0) {
40                 int j = (i-1) / 2;
41                 if (maxHeap[j].harta >= items[curr].harta){
42                     break;
43                 }
44                 maxHeap[i] = maxHeap[j];
45                 i = j;
46             }
47             maxHeap[i] = items[curr];
48             maxHeapSize++;
49             curr++;
50         }
51         if (maxHeapSize > 0) {
52             total += maxHeap[0].harta;
53             maxHeapSize--;
54             maxHeap[0] = maxHeap[maxHeapSize];
55             int i = 0;
56
57             while (i*2+1 < maxHeapSize) {
58                 int left = i * 2 + 1;
59                 int right = i * 2 + 2;
60                 int j = left;
61                 if (right < maxHeapSize && maxHeap[right].harta > maxHeap[left].harta){
62                     j = right;
63                 }
64                 if (maxHeap[j].harta <= maxHeap[maxHeapSize].harta){
65                     break;
66                 }
67                 maxHeap[i] = maxHeap[j];
68                 i = j;
69             }
70             maxHeap[i] = maxHeap[maxHeapSize];
71             index++;

```

```
71 |  
72 |  
73 |  
74 | }  
75 |  
    }  
    printf("%lld\n", total);  
    return 0;
```

Analisa Permasalahan

Pada problem H kita diminta untuk membuat sebuah program dengan algoritma greedy dengan cara mengambil harta semaksimal mungkin dengan batas berat yang ditentukan oleh user untuk setiap kantong yang tersedia. Pada problem ini kita diminta untuk mencari total harta maksimum yang dapat ditampung dalam horse cart yang telah ada. Setiap parameter baik itu jumlah kantongnya dan maksimal beratnya, jumlah kantong yang akan dipertimbangkan untuk diangkut dengan masing-masing beratnya semua itu ditentukan oleh user dengan metode input. Dalam soal ini setelah kami teliti lebih lanjut dikehendaki untuk mengurutkan menggunakan quicksort atau dengan kata lain priority queue dimana disorting berdasarkan berat masing-masing kantong baik itu yang tersedia maupun yang akan dikehendaki dibawa. Jika beban sama maka akan diprioritaskan dengan harta yang lebih besar untuk dimasukkan ke dalam kantong-kantong yang ada. Pemanfaatan metode fractional knapsack problem ditekankan dalam pembuatan kode ini.

Cara Penyelesaian secara Algoritma

Di dalam pemecahan masalah dalam algoritma yang diminta kami menggunakan implementasi dari Max Heap dengan menggunakan priority queue di mana melibatkan algoritma greedy dalam pelaksanaannya. Detailnya sendiri adalah kami menggunakan dua library dalam bahasa C yaitu stdio untuk input/output dan stdlib untuk mengalokasikan memori secara dinamis. Kemudian kami melakukan define value untuk variabel MAX agar mengikat secara konstan nilainya yaitu berisi dengan value 100000. Kemudian kami mendeklarasi suatu struct bernama Item yang digunakan untuk menyimpan value berisi berat dan harta. Pembuatan function int compareForQuickSort(const void *a, const void *b) digunakan untuk membandingkan value Item sebelum dan sesudahnya yang akan menjadi trigger untuk digunakan function qsort untuk mengurutkan Item berdasarkan beratnya. Kemudian kita masuk dalam fungsi main dimana kita meminta user untuk menginput berat untuk ada berapa data yang akan didata dan harta sebagai jumlah maksimal karung yang dapat ditampung oleh gerobak. Kita deklarasikan sebuah array yang akan menampung data items yang kita miliki. Kita melakukan looping dengan

menggunakan parameter value berat sebagai pembatas lalu kita meminta input user untuk memberikan masing-masing value berat dan harta setiap itemsnya. Kita deklarasikan array kapasitas untuk menyimpan value maksimal dari masing-masing karung di dalam gerobak yang kita akan minta kepada user melalui looping dengan batas harta yang kita tentukan di baris ke 17. Kemudian kita lakukan qsort Items berdasarkan berat dan qsort kapasitas berdasarkan batas atas masing-masing karung. Kita membuat sebuah array kembali dengan nama maxHeap untuk menyimpan item dengan nilai tertinggi.

Kita deklarasikan variabel-variabel baru. Kemudian kita mulai pengecekan dengan menggunakan while index < harta yang artinya untuk mengecek dari awal hingga akhir items yang ada. while (curr < berat && items[curr].berat <= kapasitas[index]) { digunakan untuk mengecek apabila item masih ada yang belum dicek dan berat item lebih kecil sama dengan dengan kapasitas yang ada. Kemudian kita mengassign value items ke dalam maxHeap. Deklarasikan int i dengan value maxHeapSize untuk menyimpan index item yang baru dimasukkan ke dalam array maxHeap.

Setelah selesai memasukkan value ke maxHeap maka akan diketahui berapa hasil index tertinggi yang ada. Lalu pengecekan selama i > 0 yang dikenal sebagai operasi upheap untuk memastikan agar isi dari maxHeap terjaga. int j = (i-1) / 2; digunakan untuk menembak indeks dari parent heap yang ada yang akan dibandingkan indeks value pada maxHeap yaitu parentnya dengan items yang ada jika parent lebih besar sama dengan maka kita akan break jika belum terpenuhi juga maka kita perbarui indeks i dengan indeks dari parent. Jika berhasil break dan kondisi i > 0 sudah selesai maka kita masukkan items dengan ketentuan yang sesuai ke dalam array maxHeap[i] lakukanlah hingga kapasitas terpenuhi atau batas nya terpenuhi untuk setiap karung.

Kita cek apakah maxHeapSize terdapat isi atau tidak jika iya maka kita tambahkan ke dalam total harta yang ada di dalam array maxHeap dengan indeks 0 atau dengan kata lain dengan harta terbesar. Kemudian kita hapus satu ukuran maxHeapSize karena satu elemen terhapus karena sudah ditotal. Kemudian gantikan indeks ke 0 maxHeap dengan maxHeap[maxHeapSize] atau nilai terkecil untuk value hartanya.

Lakukan operasi downheap dari elemen teratas maxHeap, while ($i * 2 + 1 < \text{maxHeapSize}$) { ... } digunakan untuk operasi downheap setelah kita menghapus elemen teratas. $\text{int left} = i * 2 + 1$; untuk indeks anak kiri elemen ke i sebaliknya untuk $\text{int right} = i * 2 + 2$; untuk indeks anak kanan elemen ke i. $\text{int j} = \text{left}$; untuk menyimpan dan menggagap anak kiri adalah nilai harta tertinggi yang disimpan di variabel j. Mengecek apakah ada anak kanan dan apabila nilai harta anak kanan ada yang lebih besar dengan menggunakan kode seperti ini `if (right < maxHeapSize && maxHeap[right].harta > maxHeap[left].harta) { j = right; }`. Kemudian kita cek apakah anak terbesar memiliki harta yang kurang dari atau sama dengan nilai harta elemen terakhir dari maxHeap atau tidak jika terpenuhi maka kita hentikan looping untuk menambahkan total dari value yang bisa ditampung dengan mengawalinya dengan `maxHeap[i] = maxHeap[j]; i = j;` untuk melakukan swap elemen ke i dengan anak terbesar. `maxHeap[i] = maxHeap[maxHeapSize];` digunakan untuk menempatkan elemen terakhir maxHeap pada posisi yang sesuai setelah proses downHeap baru kita tambahkan indeksnya untuk proses lebih lanjut. Akhirnya kita cetak total dari harta yang bisa ditampung berdasarkan berat yang tersedia setiap kantongnya di gerobak.

Analisa Kompleksitas Waktu dan Ruang dari Algoritma

- Kompleksitas Ruang

Variabel -variabel seperti berat dan harta memiliki kompleksitas ruang sebesar $O(1)$. Lalu kita bisa melihat juga bahwa terdapat array berat dan harta yang masing-masing memiliki kompleksitas $O(N)$. Terdapat fungsi quicksort dalam program ini dimana quicksort sendiri mempunyai kompleksitas ruang sebesar $O(1)$. Array dari maxheap tidak membuat array yang baru melainkan menggunakan array yang sama sehingga kompleksitas ruangnya adalah $O(1)$. Sehingga kompleksitas ruang dari program ini adalah $O(N)$ sebab worst case dari problem ini adalah $O(N)$ dan konstanta tidak dihitung sehingga kompleksitas ruang problem ini adalah $O(N)$.

- Kompleksitas Waktu

Dalam program tersebut terdapat 2 buah integer berat dan harta yang menghasilkan kompleksitas $O(1)$. Bagian ini `scanf("%d", &kapasitas[i])` memiliki kompleksitas $O(N)$ untuk membaca input dari kapasitas. Terdapat 2 quicksort yang memiliki kompleksitas sebesar $O(N \log N)$ dan $O(M \log M)$ dimana n adalah berat dan m adalah harta sesuai

dengan tujuan pengurutan dengan faktor n dan m tersebut. While pada line 34 akan berjalan sebanyak N yang dimana N adalah harta sesuai dengan prinsipnya while yang bergantung selama lebih besar dari harta dan pada while yang kedua bergantung pada berat pada pernyataan while ($\text{curr} < \text{berat} \ \&\& \ \text{items}[\text{curr}].\text{berat} \leq \text{kapasitas}[\text{index}]$) { yang menyebabkan kompleksitasnya adalah $O(n)$ dan di line 35 dan 56 whilenya berjalan sebanyak $O(\log N)$ diakibatkan oleh iterasi di dalam while dimana ada while ($i > 0$) { yang menyebabkan looping sebanyak logaritma basis 2 dari maxHeapSize dikarenakan proses pengaturan ulang max heap (heapify) setelah penambahan elemen baru ke dalam max heap dan terdapatnya pembagian index pada saat pencarian parent dari elemen indeks i , selain itu juga diketahui ketinggian pohon biner max heap adalah logaritma basis 2 dari jumlah elemen dalam heap sehingga didapati kompleksitasnya adalah $O(\log(\text{maxHeapSize}))$ maka dari itu didapati jumlah waktu untuk semua iterasi while loop kedua adalah $O(\text{berat} * \log(\text{maxHeapSize}))$ yang dapat diconvert menjadi $O(n \log n)$. Sehingga kompleksitas dari program ini adalah $O(N \log N)$ sebab $O(N \log N)$ merupakan worst case dalam program ini alasanya karena N yang mana adalah berat merupakan variabel yang dominan di kode ini sehingga kasus seperti $O(1)$ dan $O(N)$ tidak diperhitungkan sebab $O(N \log N)$ sudah merupakan kemungkinan terburuk dan konstanta tidak diperhitungkan di dalam perhitungan kompleksitas waktu.

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

Problem M

```
#include<stdio.h>

int main(void)
{
    long long int n , m , k;
    scanf("%lld %lld %lld", &n, &m, &k);

    if (n * m == k)
    {
        printf("0\n");
        return 0;
    }
    if (k % n == 0 || k % m == 0)
    {
        printf("1\n");
        return 0;
    }
    if (m > n)
    {
        long long int temp = n;
        n = m;
        m = temp;
    }
    for (int i = 1; i < n; i++)
    {
```

```
        if (k % i == 0 && k <= m * i)
        {
            printf("2\n");
            return 0;
        }
    }
    k = n * m - k;
    for (int i = 1; i < n; i++)
    {
        if (k % i == 0 && k <= m * i)
        {
            printf("2\n");
            return 0;
        }
    }
    printf("3\n");
}
```

```
A.cpp B.cpp ProblemH.cpp M.cpp
1  #include<stdio.h>
2
3  int main(void)
4  {
5      long long int n , m , k;
6      scanf("%lld %lld %lld", &n, &m, &k);
7
8      if (n * m == k)
9      {
10         printf("0\n");
11         return 0;
12     }
13     if (k % n == 0 || k % m == 0)
14     {
15         printf("1\n");
16         return 0;
17     }
18     if (m > n)
19     {
20         long long int temp = n;
21         n = m;
22         m = temp;
23     }
24     for (int i = 1; i < n; i++)
25     {
26         if (k % i == 0 && k <= m * i)
27         {
28             printf("2\n");
29             return 0;
30         }
31     }
32     k = n * m - k;
33     for (int i = 1; i < n; i++)
34     {
35         if (k % i == 0 && k <= m * i)
36         {
37             printf("2\n");
38
39             return 0;
40         }
41     }
42     printf("3\n");
}
```

Analisa Permasalahan

Soal meminta untuk mebuatkan algoritma menghitung berapa banyak potongan yang perlu dilakukan untuk memakan coklat dalam jumlah yang pas. Sebagai contoh terdapat sebuah coklat dengan tinggi 4 dan lebar 4 dan user ingin memakan 10. Maka hal yang perlu dilakukan adalah memecah dari 4×4 jadi $4 \times 2 + 4 \times 2$ dan kemudian satunya di potong lagi menjadi 2×1 . Sehingga potongan yang perlu dilakukan adalah 2 kali untuk mencapai tujuan 10 yaitu $4 \times 2 + 2 \times 1$.

Cara Penyelesaian secara Algoritma

Ketika user memasukan 3 angka yaitu pertama lebar, panjang dan total coklat (n , m , k) yang dimakan. Yang pertama harus mengecek jika total coklat merupakan lebar kali panjang maka tidak perlu dipotong sehingga keluarkan angka 0. Jika total coklat jika habis di bagi oleh panjang atau lebar maka keluarkan angka 1 karena hanya perlu di potong sekali. Kemudian jika tidak terkena kondisi di atas maka lakukan looping dimana tujuan dari looping ini mencari patahan yang terkecil yang bisa dipotong jika tidak ada maka diperlukan 2 step untuk hal tersebut. Step ini perlu dilakukan dua kali dengan coklat yang sudah di pisahkan dilakukan lagi looping mencari terkecil. Kemudian jika semua kondisi tidak terpenuhi maka keluarkan angka 3.

Analisa Kompleksitas Waktu dan Ruang dari Algoritma

- Kompleksitas ruang:

Algoritma ini menyimpan data panjang di variabel n , lebar di m dan jumlah luas coklat yang ingin dimakan di k . Kemudian untuk looping juga digunakan i.e. n , m , k dan i merupakan sebuah variabel yang konstan sehingga kompleksitas ruangnya adalah $O(1)$.

- Kompleksitas Waktu :

Pada baris 5 sampai 23 waktu yang diggunakan adalah 1. Karena scanf dan if hanya dijalankan sekali. Siapa yang menjadi n itu ditentukan oleh panjang atau lebar yang lebih kecil. n akan selalu menjadi lebih kecil atau sama dengan m dan saya akan menyebutnya sebagai x . Kemudian pada 24 sampai 30 terdapat sebuah looping yang beawal 1 sampai n yang berarti waktu yang dibutuhkan adalah $O(x-1)$. Terjadi lagi looping tersebut pada 33 sampai 40 yang juga $O(x-1)$. Kemudian ditutup dengan line 41 yaitu print yang merupakan $O(1)$. Sehingga total dari kompleksitas waktunya adalah $O(x-1) = O(n)$.