The Report of QUIZ 2

“Job Sequencing with Deadline”

Group 3

1. Isnaini Nurul KurniaSari (05111740000010)
2. Bima Satria Ramadhan (05111740000081)
3. Meila Kamilia (05111740000189)

Design & Analysis of Algorithms (F)

Institut Teknologi Sepuluh Nopember

Year Lessons 2018/2019

Job Sequencing Problem with Deadline

- **Problem Description:**

In the industrial world, time efficiency is very important. The more efficient production process, the more total production produced or minimum costs that must be incurred. A convection company is a company that produces apparel, such as shirts, pants, shirts and jackets. Each type of clothing requires different finishing times, for example T-shirt production is faster than jacket production. The convection industry uses operators and sewing machines for each of its work stations.

In terms of product manufacturing, convection companies are included in the category of make to order (MTO) because products will be made if there are orders from customers. Every order from customers varies greatly from the type of clothing made and the amount. The purpose of this case is to solve the problem of scheduling apparel production in a convection company in order to minimize makespan (total work completion time) for each order (job order) received from customers.

Based on the order, the convection company must determine the production scheduling in the form of a sequence of types (priority) of products produced for each job and how much profit is earned for each job.

  - **Points to remember**
    1. In this problem we have n jobs j1, j2, … jn each has an associated deadline d1, d2, … dn and profit p1, p2, ... pn.
    2. Profit will only be awarded or earned if the job is completed on or before the deadline.
    3. We assume that each job takes unit time to complete.
    4. The objective is to earn maximum profit when only one job can be scheduled or processed at any given time.

  - **Input :**

Job	Deadline	Profit
j1	2	60
j2	1	100
j3	3	20
j4	2	40
j5	1	20

- **Output :**

Job	Deadline	Profit
j2	1	100
j1	2	60
j4	2	40
j3	3	20
j5	1	20

dmax: 3

Required Jobs: j2 --> j1 --> j3
Max Profit: 180

- **Problem Abstraction**

The convection industry is a company that produces various kinds of apparel such as shirts, and trousers. In terms of product manufacturing, this industry uses order (MTO), meaning the product will be made if there is a customer order. Each order received varies greatly in terms of the type of clothing to be produced and the amount so that it is necessary to schedule production on each order. In preparing the production schedule, the company must be able to allocate every variety of work into the work station (sewing machine and operator) in a balanced manner to produce a minimum makespan (minimum total work completion time).

To arrange a production schedule on parallel machines to produce a minimum makespan we can use the greedy algorithm. From the results of the study, the greedy algorithm always produces the optimal solution for this case. In addition, the greedy algorithm is always the fastest in generating solutions compared to the exhaustive search algorithm. Greedy is an algorithm to find the most optimum solution which has the maximum value at each step, or it can be called a local maximum. But in most cases, greedy cannot produce optimal values, but usually can provide a solution that approaches the most optimum value.

The setting is that we have n jobs, each of which takes unit time, and a processor on which we would like to schedule them in as profitable a manner as possible. Each job has a profit associated with it, as well as a deadline; if the job is not scheduled by the deadline, then we don't get the profit. Because each job takes the same amount of time, we will think of a Schedule S as consisting of a sequence of job "slots" 1, 2, 3, . . . where S(t) is the job scheduled in slot t. (If one wishes, one can think of a job scheduled in slot t as beginning at time t − 1 and ending at time t, but this is not really necessary.)

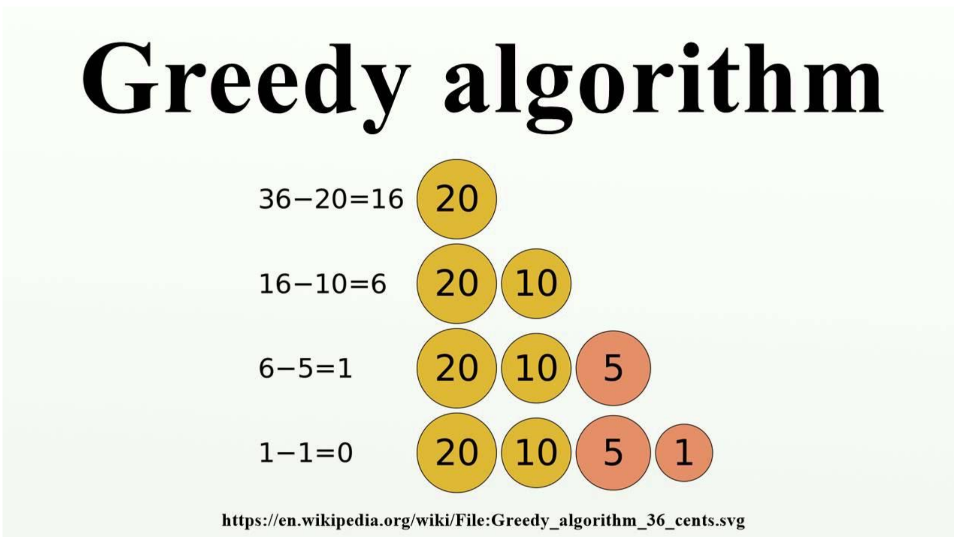
More formally, the input is a sequence (d1, g1),(d2, g2), · · · ,(dn, gn) where gi is a nonnegative real number representing the profit obtainable from job i, and di ∈ N is the deadline for job i; it doesn't hurt to assume that 1 ≤ di ≤ n. (The reason why we can assume that every deadline is less than or equal to n is because even if some deadlines were bigger, every feasible schedule could be "contracted" so that no job was placed in a slot bigger than n.)

Definition 1 A schedule S is an array: S(1), S(2), ..., S(n) where S(t) ∈ {0, 1, 2, · · · n} for each t ∈ {1, 2, · · · , n}.The intuition is that S(t) is the job scheduled by S in slot t; if S(t) = 0, this means that no job is scheduled in slot t.

Definition 2 S is feasible if (a) If S(t) = i > 0, then t ≤ di. (Every scheduled job meets its deadline) (b) If t1 6= t2 and S(t1) 6= 0, then S(t1) 6= S(t2). (Each job is scheduled at most once).

We define the profit of a feasible schedule S by P(S) = gS(1) + gS(2) + ... + gS(n) , where g0 = 0 by definition.

Goal: Find a feasible schedule S whose profit P(S) is as large as possible; we call such a schedule optimal. We shall consider the following greedy algorithm. This algorithm begins by sorting the jobs in order of decreasing (actually nonincreasing) profits. Then, starting with the empty schedule, it considers the jobs one at a time; if a job can be (feasibly) added, then it is added to the schedule in the latest possible (feasible) slot.



Greedy:

Sort the jobs so that:

g1 ≥ g2 ≥ . . . ≥ gn

for t : 1..n S(t) ← 0 {Initialize array S(1), S(2), ..., S(n)}

end for

for i : 1..n

Schedule job i in the latest possible free slot meeting its deadline;

if there is no such slot, do not schedule i.

end for

Example. Input of Greedy:

Job i:	1	2	3	4	Comments
Deadline d_i:	3	2	3	1	(when job must finish by)
Profit g_i:	9	7	7	2	(already sorted in order of profits)

Initialize S(t):

t	1	2	3	4
$S(t)$	0	0	0	0

Apply Greedy: Job 1 is the most profitable, and we consider it first. After 4 iterations:

t	1	2	3	4
$S(t)$	3	2	1	0

Job 3 is scheduled in slot 1 because its deadline t = 3, as well as slot t = 2, has already been filled.

P(S) = g3 + g2 + g1 = 7 + 7 + 9 = 23.

Theorem 1 The schedule output by the greedy algorithm is optimal, that is, it is feasible and the profit is as large as possible among all feasible solutions.

We will prove this using our standard method for proving correctness of greedy algorithms. We say feasible schedule S 0 extends feasible schedule S iff for all t (1 ≤ t ≤ n),

if S(t) 6= 0 then S 0 (t) = S(t).

Definition 3 A feasible schedule is promising after stage i if it can be extended to an optimal feasible schedule by adding only jobs from {i + 1, · · · , n}.

Lemma 1 For 0 ≤ i ≤ n, let Si be the value of S after i stages of the greedy algorithm, that is, after examining jobs 1, · · · , i. Then the following predicate P(i) holds for every i, 0 ≤ i ≤ n:

P(i) : Si is promising after stage i.This Lemma implies that the result of Greedy is optimal. This is because P(n) tells us that the result of Greedy can be extended to an optimal schedule using only jobs from ∅. Therefore the result of Greedy must be an optimal schedule.

Proof of Lemma: To see that P(0) holds, consider any optimal schedule Sopt. Clearly Sopt extends the empty schedule, using only jobs from {1, · · · , n}.So let 0 ≤ i < n and assume P(i).

We want to show P(i + 1). By assumption, Si can be extended to some optimal schedule Sopt using only jobs from {i + 1, · · · , n}.

Case 1: Job i + 1 cannot be scheduled, so Si+1 = Si . Since Sopt extends Si , we know that Sopt does not schedule job i + 1. So Sopt extends Si+1 using only jobs from {i + 2, · · · , n}.

Case 2: Job i + 1 is scheduled by the algorithm, say at time t0 (so Si+1(t0) = i + 1 and t0 is the latest free slot in Si that is ≤ di+1).

Subcase 2A: Job i + 1 occurs in Sopt at some time t1 (where t1 may or may not be equal to t0).

Then t1 ≤ t0 (because Sopt extends Si and t0 is as large as possible) and Sopt(t1) = i+ 1 = Si+1(t0).

If t0 = t1 we are finished with this case, since then Sopt extends Si+1 using only jobs from {i + 2, · · · , n}. Otherwise, we have t1 < t0. Say that Sopt(t0) = j 6= i + 1. Form S 0 opt by interchanging the values in slots t1 and t0 in Sopt. Thus S 0 opt(t1) = Sopt(t0) = j and S 0 opt(t0) = Sopt(t1) = i + 1. The new schedule S 0 opt is feasible (since if j 6= 0, we have moved job j to an earlier slot), and S 0 opt extends Si+1 using only jobs from {i + 2, · · · , n}. We also have P(Sopt) = P(S 0 opt), and therefore S 0 opt is also optimal.

Subcase 2B: Job i + 1 does not occur in Sopt.

Define a new schedule S 0 opt to be the same as Sopt except for time t0, where we define S 0 opt(t0) = i+1. Then S 0 opt is feasible and extends Si+1 using only jobs from {i + 2, · · · , n}.

To finish the proof for this case, we must show that S 0 opt is optimal. If Sopt(t0) = 0, then we have P(S 0 opt) = P(Sopt)+gi+1 ≥ P(Sopt). Since Sopt is optimal, we must have P(S 0 opt) = P(Sopt) and S 0 opt is optimal. So say that Sopt(t0) = j, j > 0, j 6= i + 1. Recall that Sopt extends Si using only jobs from {i + 1, · · · , n}. So j > i + 1, so gj ≤ gi+1. We have P(S 0 opt) = P(Sopt) + gi+1 − gj ≥ P(Sopt). As above, this implies that S 0 opt is optimal.

We still have to discuss the running time of the algorithm. The initial sorting can be done in time O(n log n), and the first loop takes time O(n). It is not hard to implement each body of the second loop in time O(n), so the total loop takes time O(n 2). So the total algorithm runs in time O(n 2). Using a more sophisticated data structure one can reduce this running time to O(n log n), but in any case it is a polynomial-time algorithm.

While for sorting (sorting) our profits, we use bubble sort. Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where **n** is the number of items.

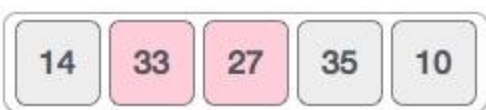
How Bubble Sort Works?

We take an unsorted array for our example. Bubble sort takes $O(n^2)$ time so we're keeping it short and precise.

Bubble sort starts with very first two elements, comparing them to check which one is greater.

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

We find that 27 is smaller than 33 and these two values must be swapped.



The new array should look like this −

Next we compare 33 and 35. We find that both are in already sorted positions.

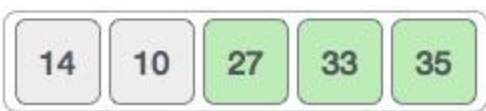
Then we move to the next two values, 35 and 10.

We know then that 10 is smaller 35. Hence they are not sorted.

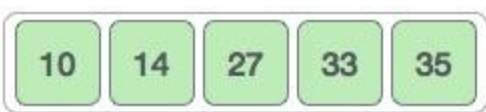
We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this −

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this −

Notice that after each iteration, at least one value moves at the end.



And when there's no swap required, bubble sorts learns that an array is completely sorted.



Now we should look into some practical aspects of bubble sort.

- **Solution**

Job sequencing is the set of jobs, associated with the job i where deadline $d_i >= 0$ and profit $p_i > 0$. For any job i the profit is earned if and only if the job is completed by its deadline. To complete a job, one has to process the job on a machine for one unit of time. Only one machine is available for processing the jobs.

Steps for performing job sequencing with deadline using greedy approach is as follows:

1. Sort all the jobs based on the profit in an decreasing order.
2. Let α be the maximum deadline that will define the size of array.

3. Create a solution array S with d slots.
4. Initialize the content of array S with zero.
5. Check for all jobs.
    a. If scheduling is possible a lot i^{th} slot of array s to job i.
    b. Otherwise look for location (i-1), (i-2)...1.
    c. Schedule the job if possible else reject.
6. Return array S as the answer.
7. End.

- **Point to remember :**
    1. In this problem, we have n jobs with id jobs; j1, j2,j3,…, j_n each has an associated deadline d1, d2,…,d_n and profit p1,p2,…,p_n.
    2. Profit will only be awarded or earned if the job is completed or before the deadline.
    3. We assume that each job takes a unit time to complete.
    4. The objective is to earn maximum profit when only one job can be scheduled or processed at any given time.

Consider the following 5 id jobs and their associated deadline and profit:

Index	1	2	3	4	5
ID Job	j1	j2	j3	j4	j5
Deadline	2	1	3	2	1
Profit	60	100	20	40	20

Then, we can sort the id jobs according to their profit in descending order. If two or more jobs are having the same profit, we can sort them as their entry in the job list :

Index	1	2	3	4	5
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

Job j3 and j5 are having profit 20 so we placed j3 first as it came before j5.

Then we find the maximum deadline value, then looking at the id jobs that we can say the max deadline value is 3

Index	1	2	3	4	5
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

dmax

Index	1	2	3	4	5
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

As dmax = 3, so we will have THREE slots to keep track of free time slots, and then we set the time slot status to **EMPTY**;

Time slot	1	2	3
status	EMPTY	EMPTY	EMPTY

Total number of job is 5 so we can write n = 5 and dmax = 3

Index	1	2	3	4	5
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

If we look at job j2, it has a deadline 1, it means that we have to complete job j2 in time slot 1 if we want to earn its profit.

Index	1	2	3	4	5
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

We get n = 5 and dmax = 3

Index	1	2	3	4	5
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

Similarly, if we look at id job j1. It has a deadline 2, it means we have to complete job j1 on or before time slot 2 in order to earn its profit.

Index	1	2	3	4	5
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

Similarly, if we look at job j3 it has a deadline 3 this means we have to complete job j3 on or before time slot 3 in order to earn its profit.

Our objective is to select jobs that will give us higher profit.

i

Index	1	2	3	4	5
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

I = 1

k = min(dmax, Deadline(i))

k = min(3, Deadline(1))

k = min (3,1)

k = 1

is k >= 1 ?

1 >= 1

YES it's true

Then we can check to the time slot(k) == EMPTY

Time slot	**1**	**2**	**3**
status	EMPTY	EMPTY	EMPTY

Time slot(1) == EMPTY

Because Time slot(1) is EMPTY, we can fill the time slot (1) with j2

Time slot	**1**	**2**	**3**
status	j2	EMPTY	EMPTY

i

Index	**1**	**2**	**3**	**4**	**5**
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

I = 2

K = min(dmax, Deadline(i))

K = min(3, Deadline(2))

K = min (3,2)

K = 2

is k >= 1 ?

2 > = 1 ; Yes, it's true

After that we can check the time slot(k) == EMPTY. Then we get the time slot (2) == EMPTY

Time slot	**1**	**2**	**3**
Status	j2	EMPTY	EMPTY

YES, time slot(2) is EMPTY

Because time slot(2) is empty, we can fill the time slot (2) with j1

Time slot	**1**	**2**	**3**
status	j2	j1	EMPTY

i

Index	**1**	**2**	**3**	**4**	**5**
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

We can assume that :

n = 5

dmax = 3

i = 3

k = min (dmax, deadline(i))

k = min (3, deadline (3))

k = min (3,2)

k = 2

is k >= 1 ?

2 > = 1

YES

After that we can check the time slot(k) == EMPTY or NOT EMPTY

Time slot	**1**	**2**	**3**
status	j2	j1	EMPTY

Time slot (2) is **NOT EMPTY**

Because time slot (k) is NOT EMPTY, so we reduce k by 1

n = 5

k = 1

is k > = 1 ?

1 > = 1

YES **it's true**

Then, check time slot(k) == EMPTY

Time slot	**1**	**2**	**3**
status	j2	j1	EMPTY

Time slot (1) is **NOT EMPTY**

Because time slot (k) is NOT EMPTY, so we reduce k by 1

i

Index	**1**	**2**	**3**	**4**	**5**
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

With n = 5;

dmax = 3;

We can assume that:

i = 4

k = min (dmax , deadline (i))

k = min (3, deadline (4))

k = min (3,3)

k = 3

is k >= 1?

so, 3 >= 1

YES, it's TRUE

Time slot	**1**	**2**	**3**
Status	j2	j1	EMPTY

Time slot (3) is **EMPTY**

Because time slot(3) is EMPTY, we can fill the time slot (3) with j3

Time slot	**1**	**2**	**3**
status	j2	j1	j3

Index	**1**	**2**	**3**	**4**	**5**
ID Job	j2	j1	j4	j3	j5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

n = 5

dmax = 3

required job sequence is

j2 → j1 → j3

and we know that max profit is

100 + 60 +20 = 180

- Source Code

```
#include <stdio.h>

#define MAX 100

using namespace std;

typedef struct Job {
  char id[MAX];
  int deadline;
  int profit;
} Job;

//fungsi untuk menghitung job mana yang akan diambil berdasar deadline dan profit secara greedy
void jobSequencingWithDeadline(Job jobs[], int n);

//fungsi mencari nilai terkecil dari dua nilai yang dibandingkan
int minValue(int x, int y) {
  if(x < y) return x;
  return y;
}

int main(void) {
  //variables
  int i, j;

  //jobs with deadline and profit
//  Job jobs[6] = {
//    {"j1", 2,  60},
//    {"j2", 1, 100},
//    {"j3", 3,  20},
//    {"j4", 3,  40},
//    {"j5", 1,  20},
//    {"j6", 4,  50},
//  };
```

```c
    //jumlah jobs
    printf("Masukkan jumlah job yang diinginkan : ");
    int n;
    scanf("%d", &n);
    Job jobs[n];
    //temp
    Job temp;
    for(int i=0;i<n;i++) {
      printf("Masukkan id, deadline dan profit job %d secara berurutan (contoh : j1 1 10) : ", i+1);
      scanf("%s%d%d", &jobs[i].id, &jobs[i].deadline, &jobs[i].profit);
    }

    //sort profit job secara descending (dari profit terbesar)
    for(i = 1; i < n; i++) {
      for(j = 0; j < n - i; j++) {
        if(jobs[j+1].profit > jobs[j].profit) {
          temp = jobs[j+1];
          jobs[j+1] = jobs[j];
          jobs[j] = temp;
        }
      }
    }

    //print id job, deadline dan profit yang sudah diurutkan berdasar profit terbesar
    printf("\n\t%s\t|%s\t|%s\n", "Id Job", "Deadline", "Profit");
    for(i = 0; i < n; i++) {
      printf("\t%s\t|%d\t\t|%d\n", jobs[i].id, jobs[i].deadline, jobs[i].profit);
    }

    //memanggil fungsi untuk menghitung job mana yang akan diambil berdasar deadline dan profit secara greedy
    jobSequencingWithDeadline(jobs, n);

    return 0;
}
```

```c
//fungsi untuk menghitung job mana yang akan diambil berdasar deadline dan profit secara greedy
void jobSequencingWithDeadline(Job jobs[], int n) {
  //variables
  int i, j, k, maxprofit;

  //time slots kosong
  int timeslot[MAX];

  //time slots yang sudah terisi
  int filledTimeSlot = 0;

  //cari nilai deadline maksimal
  int dmax = 0;
  for(i = 0; i < n; i++) {
    if(jobs[i].deadline > dmax) {
      dmax = jobs[i].deadline;
    }
  }

  //time slots yang kosong diinisialisasikan dengan -1 (-1 menandakan kosong/tidak ada isinya)
  for(i = 1; i <= dmax; i++) {
    timeslot[i] = -1;
  }

  printf("\ndmax: %d\n", dmax);

  //perulangan untuk memasukkan job ke timeslot yang tersedia
  for(i = 0; i < n; i++) {
    k = minValue(dmax, jobs[i].deadline);
    while(k >= 1) {
      //jika timeslot kosong maka diisi
      if(timeslot[k] == -1) {
        timeslot[k] = i;
        filledTimeSlot++;
        break;
```

```c
            }
            //jika timeslot sudah terisi decrement k untuk melakukan cek timeslot lain yang belum terisi
            k--;
        }

        //jika semua timeslots sudah terisi maka berhenti
        if(filledTimeSlot == dmax) {
            break;
        }
    }

    //hasil job yang diambil
    printf("\nUrutan Job yang diambil adalah : ");
    for(i = 1; i <= dmax; i++) {
        printf("%s", jobs[timeslot[i]].id);

        if(i < dmax) {
            printf(" --> ");
        }
    }

    //hasil profit
    maxprofit = 0;
    for(i = 1; i <= dmax; i++) {
        maxprofit += jobs[timeslot[i]].profit;
    }
    printf("\nProfit maksimum yang dihasilkan adalah : %d\n", maxprofit);
}
```

➢ Output Program

```
C:\Users\asus\Documents\PAA\cobain.exe
Masukkan jumlah job yang diinginkan : 5
Masukkan id, deadline dan profit job 1 secara berurutan (contoh : j1 1 10) : j1 2 60
Masukkan id, deadline dan profit job 2 secara berurutan (contoh : j1 1 10) : j2 1 100
Masukkan id, deadline dan profit job 3 secara berurutan (contoh : j1 1 10) : j3 3 20
Masukkan id, deadline dan profit job 4 secara berurutan (contoh : j1 1 10) : j4 2 40
Masukkan id, deadline dan profit job 5 secara berurutan (contoh : j1 1 10) : j5 1 20

        Id Job  |Deadline       |Profit
        j2      |1              |100
        j1      |2              |60
        j4      |2              |40
        j3      |3              |20
        j5      |1              |20

dmax: 3

Urutan Job yang diambil adalah : j2 --> j1 --> j3
Profit maksimum yang dihasilkan adalah : 180

--------------------------------
Process exited after 39.79 seconds with return value 0
Press any key to continue . . .
```

"By the name of Allah (God) Almighty, herewith we pledge and truly declare that we have solved quiz 2 by ourselves, didn't do any cheating by any means, didn't do any plagiarism, and didn't accept anybody's help by any means. We are going to accept all of the consequences by any means if it has proven that we have been done any cheating and/or plagiarism."

Surabaya, 29 March 2019

Isnaini Nurul KurniaSari	Bima Satria Ramadhan	Meila Kamilia
(05111740000010)	(05111740000081)	(05111740000189)