**Teacher：张涵翠**

Analysis and Design of

Algorithm

**(semester 2, 2019-2020)**

**Final Report**

**Group ID：15**

**Student IDs：2018329621212、2018329621213、2018329621072**

**Student Names：郭丹婷、黄冰慧、陈怡**

**Class and grade：计算机科学与技术（全英文）18**

**College of Information Science, Zhejiang Sci-Tech University**

***Problems***

***Authors: list all team members like “Student ID, Student Name, task”***

**2018329621212, 郭丹婷, Algorithm design and analysis, Code design**

**2018329621213, 黄冰慧, Algorithm design and analysis, PPT design**

**2018329621072, 陈怡, Algorithm design and analysis, Report design**

***Abstract: briefly introduce the work of this paper, including problems, methods and effects, no more than 150 words.***

For each input self-life and number of days, to solve the A-tomato-a-day problem, give the result in order to satisfy tomato-a-day in the smallest cost. we divided the problem into different situations. analyzed them and designed a generation algorithm. There’re three situations, (1) , (The tomato for first day can only be purchased in the first day) (2) and (The tomato purchased on day tmp has not been expired) (3) and (The tomato purchased on day tmp has been expired). For our algorithm, we use a temporary index mark for comparing the price of tomato in self-life and this temporary index represent the day of which tomato should be purchased, and give different method to update tmp and B[] to correct value. Next, we analyze the correctness and efficient of the algorithm, obtain its time complexity , and design code and test it for different input.

# I．Introduction *(describe what the problem is and the goal)*

Professor Kerry loves tomatoes! The professor eats one tomato every day, because she is obsessed with the health benefits of the potent antioxidant lycopene and because she just happens to like them very much, thank you. The price of tomatoes rises and falls during the year, and when the price of tomatoes is low, the professor would naturally like to buy as many tomatoes as she can. Because tomatoes have a shelf-life of only ***d*** days, however, she must eat a tomato bought on dayon some dayin the range , or else the tomato will spoil and be wasted. Thus, although the professor can buy as many tomatoes as she wants on any given day, because she consumes only one tomato per day, she must be circumspect about purchasing too many, even if the price is low.

The professor’s obsession has led her to worry about whether she is spending too much money on tomatoes. She has obtained historical pricing data for n days, and she knows how much she actually spent on those days. The historical data consists of an array , where is the price of a tomato on day . She would like to analyze the historical data to determine what is the minimum amount she could possibly have spent in order to satisfy her tomato-a-day habit, and then she will compare that value to what she actually spent.

**Give an efﬁcient algorithm** to determine the optimal ofﬂine (20/20 hindsight) purchasing strategy on the historical data. Given , , and , your algorithm should output, where is the number of tomatoes to buy on day .

# II．Problem analysis and Algorithm design *(formulate the problem and design proper algorithms, may exist many algorithms)*

**Problem analysis:**

**Input**: (The self-life of tomatoes)

(The number of days)

( (prices of tomatoes on day)

**Output**: () (the number of tomatoes to buy on day )

**Requirement:** (1) The tomatoes purchased should satisfy a-tomato-a-day. (2) The total price should be minimum amount.

**Algorithm design:**

*Min(a, b)* is the function to find the index of the minimum item from to .

The pseudocode of *Min(a, b)* is as follows.

**Min(a, b):**

**1. mIndex ← C[a]**

**2. for i ← a+1 to b do**

**3. if C[i] < C[mIndex] then**

**4. mIndex ← i**

**5. return mIndex**

*Buy(d, n)* is the main function to obtain the result of .

The variable tmp is used to save the No. of the day selling the cheapest tomatoes that does not expire in current day i, that is, to realize a-tomato-a-day in minimum cost, the tomato for day i should be purchased in day tmp.

For each day i,

(1) if, then , (The tomato for first day can only be purchased in the first day).

(2) if and (The tomato purchased on day tmp has not been expired.), then . (Compare the price of tomatoes on day i and day tmp, update tmp if the tomato sold on in day i is cheaper.)

(3) if and (The tomato purchased on day tmp has been expired), then ( Seach the price of tomato sold from day tmp+1 to day i, find the cheapest tomatoes sold on day tmp).

(4) Then, B[tmp]++ (Purchase a tomato sold on day tmp for eating on day i)

The pseudocode of *Buy(d, n)* is as follows.

**Buy(d, n):**

**1. B[1] ← 1**

**2. tmp ← 1**

**3. for i ← 2 to n do**

**4. if i – tmp +1 > d then**

**5. tmp ← Min(tmp, i)**

**6. else if C[i] < C[tmp] then**

**7. tmp ← i**

**8. B[tmp]++**

# III．Algorithm analysis *( correctness & efficiency of the algorithm )*

**Correctness:**

1. The algorithm has limited input.

2. The algorithm has limited output.

3. The running time of algorithm program is limited.

4. Every step of the algorithm is defined exactly.

5. Any computing step in the algorithm can be decomposed into basic executable steps.

For -th item of the input , the tomato for day n should be purchased on day (, making sure the tomato has not been expired and has cheapest price). This logic applies to all situations. In code, the capacity of input and output is defined as 105, and it can be changes to any other value that the computer memory allows. So,

1. The algorithm can get the structure satisfying the requirements for several groups of different input data.

2. The algorithm can get satisfactory results for the typical, harsh and difficult input data.

3. The algorithm produces satisfactory results for all legitimate input data.

**Efficiency of algorithm:**

(1) if, then , (The tomato for first day can only be purchased in the first day).

(2) if and (The tomato purchased on day tmp has not been expired.), then . (Compare the price of tomatoes on day i and day tmp, update tmp if the tomato sold on in day i is cheaper.)

(3) if and (The tomato purchased on day tmp has been expired), then ( Seach the price of tomato sold from day tmp+1 to day i, find the cheapest tomatoes sold on day tmp).

(4) Then, B[tmp]++ (Purchase a tomato sold on day tmp for eating on day i)

For each group of input n and d,

that means

# IV. Experiment *(describe the experiment environment and the design of experiment data (specific examples of this problem): input & output)*

**Experiment environment:**

Software Environment：

Windows10, Code::Blocks 17.12

Hardware Environment：

Computer

Inter(R) core(TM) i5-8300H CPU @ 2.30GHz 2.30GHz

RAM 8.00GB

64x

**Design of experiment data:**

**Input**:

(The self-life of tomatoes)

(The number of days)

( (prices of tomatoes on day)

**Output**:

() (the number of tomatoes to buy on day )

**Temporary variable:**

**tmp** (temporary value saves the index that represents the index of the day, on which tomatoes should be purchased for current day in execution)

**mIndex** (temporary value saves the index that represents the index of the day, on which tomatoes is cheapest from day to day .)

**Example:**

Input:

d=3, n=10, C[10]={2, 3, 1, 4, 2, 3, 7, 6, 3, 1}

Output:

B[1] = 2

B[2] = 0

B[3] = 3

B[4] = 0

B[5] = 2

B[6] = 1

B[7] = 0

B[8] = 0

B[9] = 1

B[10] = 1

# V. Code with annotations *( code with comment & result )*

**Code:**

// File: A tomato a day.cpp

#include<cstdio>

const int Maxn=105;

int C[Maxn]; // Price on each day

int B[Maxn]={0}; // The number of tomatoes should be purchased on each day

int Min(int a, int b) // Find the smallest item from C[a+1] to C[b]

{ // Return the index

int mIndex = a + 1;

for(int i = a + 1; i <= b; i++)

if(C[i] < C[mIndex])

mIndex = i;

return mIndex;

}

void Buy(int d, int n) // Find the result of B[]

{

B[1] = 1; // The first day must buy a tomato

int tmp = 1; // Initialize the tmp

for(int i = 2; i <= n; i++)

{

if(i - tmp + 1 > d) // If the tomato purchased on tmp-th day expires on day i

tmp = Min(tmp, i); // Update tmp

else if(C[i] < C[tmp]) // If the tomato purchased on tmp-th day does not expire on day i

tmp = i; // Update tmp

B[tmp]++; // Purchase a tomato sold on tmp-th day

}

}

int main()

{

int d, n;

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

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

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

Buy(d, n);

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

{

printf("B[%d] = %d ", i, B[i]);

printf("\n");

}

}

**Result:**

**Input 1：**

5 20

3 1 5 4 3

6 4 2 1 3

2 1 3 5 6

4 1 2 1 1

**Output 1：**

B[1] = 1

B[2] = 5

B[3] = 0

B[4] = 0

B[5] = 1

B[6] = 0

B[7] = 0

B[8] = 1

B[9] = 5

B[10] = 0

B[11] = 0

B[12] = 3

B[13] = 0

B[14] = 0

B[15] = 0

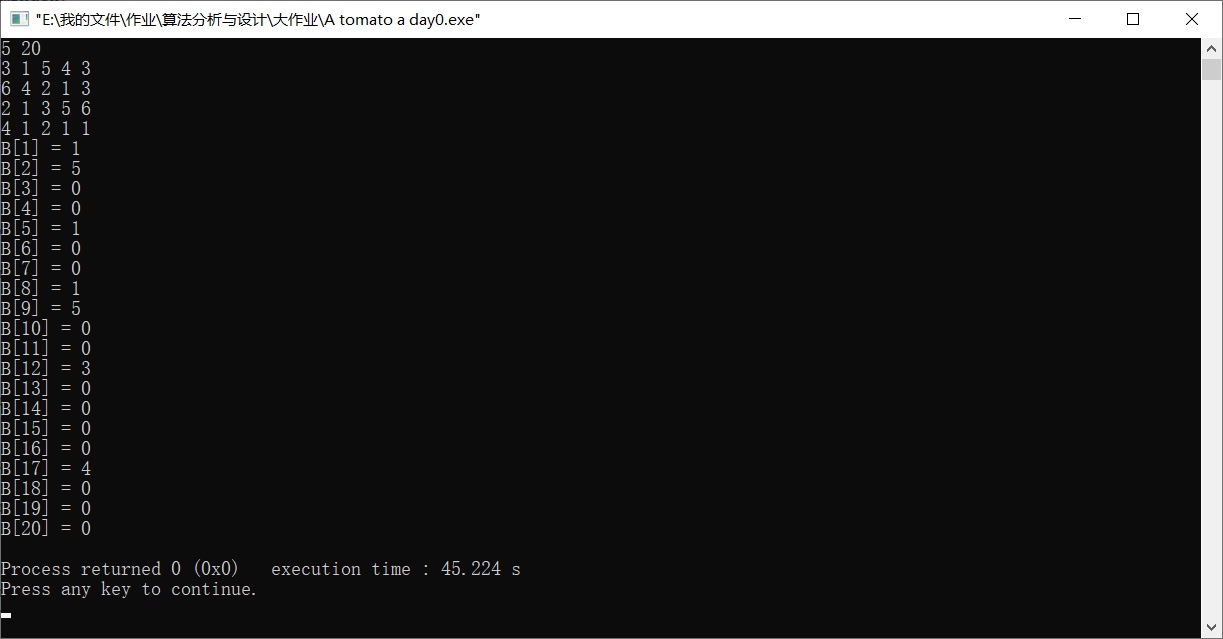
B[16] = 0

B[17] = 4

B[18] = 0

B[19] = 0

B[20] = 0



**Input 2：**

7 20

1 2 3 5 1

2 1 3 5 4

2 3 6 5 7

4 5 1 5 1

**Output 2：**

B[1] = 7

B[2] = 0

B[3] = 0

B[4] = 0

B[5] = 4

B[6] = 0

B[7] = 2

B[8] = 0

B[9] = 0

B[10] = 0

B[11] = 4

B[12] = 0

B[13] = 0

B[14] = 0

B[15] = 0

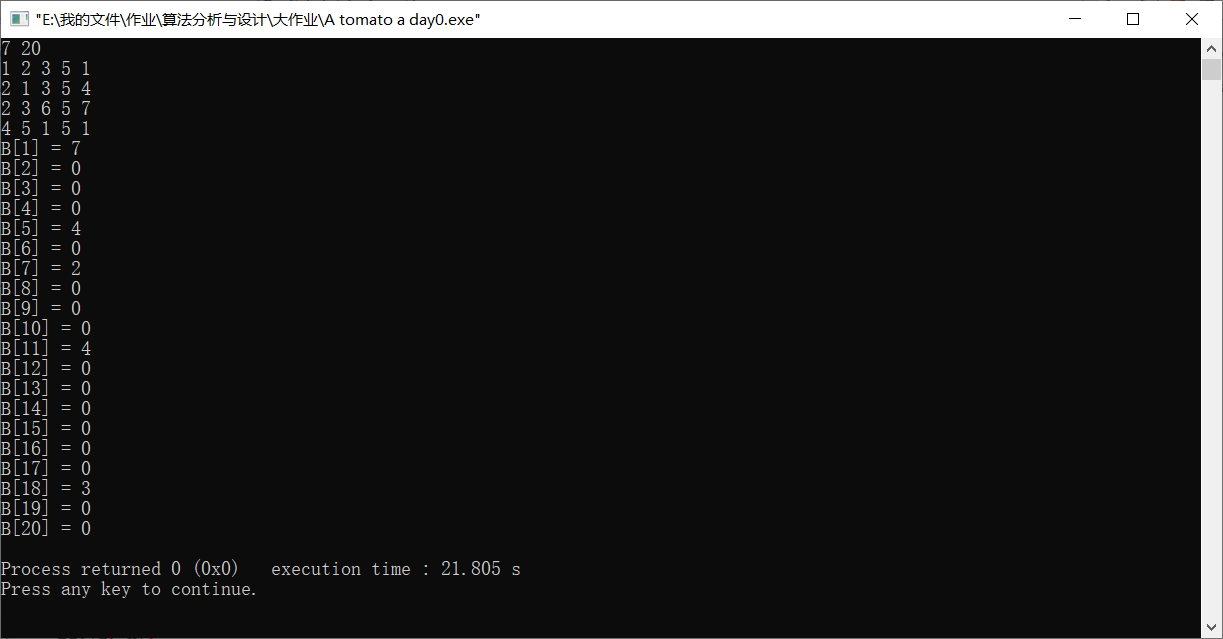
B[16] = 0

B[17] = 0

B[18] = 3

B[19] = 0

B[20] = 0



**Input 3：**

3 20

2 3 1 4 2

3 7 6 3 1

**Output 3：**

B[1] = 2

B[2] = 0

B[3] = 3

B[4] = 0

B[5] = 2

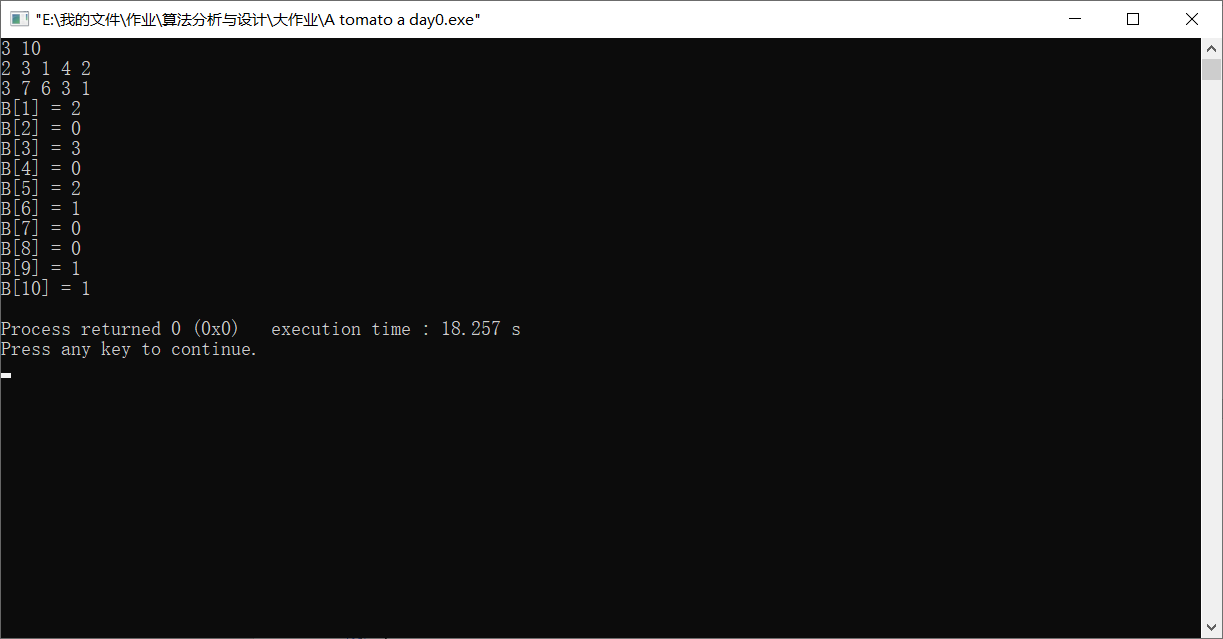
B[6] = 1

B[7] = 0

B[8] = 0

B[9] = 1

B[10] = 1



# VI. Conclusions *( describe what you have done and what result or experience have you got)*

1. Programs that use arrays should handle indexes carefully. To solve the problem that programs need to save various kinds of data as arrays, it is wise to save data from the second element of the array.

2. Before writing code, it is important to consider the functions to be implemented and the relevant data required. Otherwise, the code can be illogical and complex, and spend a lot of time maintaining your code.

3. Predict the errors as far as possible and avoid them in the code, so as to maintain and test the program many times to ensure the correctness of the program

4. With correctness in mind, optimizing the efficiency and performance of the program may be a desirable way to do algorithm analysis before code design.

5. Through this experience, we have a deeper understanding of the greedy algorithm. The greedy algorithm cannot get the overall optimal solution for all problems, but the key is the choice of greedy strategy, which is only related to the current state. Greed is to solve a big problem by solving small problems. For greedy problems, we should not start with big problems, but seek the optimal solution step by step, and finally get the optimal solution of the whole problem naturally.

# References *( list all the masteries, such as website, articles, books that you have referred to. The format is as follows)*

*Article*: [1] Xue Jinyun. A unified approach for developing efficient algorithmic programs[J]. 1997.12: 314-329.

*book*: [2] Jeffrey H, Kingston. Algorithms and Data Structures: Design, Correctness, Analysis [M] J. Comput. Sci. Technol, 1990.

*Website*: [3] Some thoughts on greedy algorithms[EB/CSDN].

https://blog.csdn.net/whaker/article/details/104891291?utm\_medium=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.nonecase&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.nonecase