CMPE 300 – Analysis of Algorithms

Fall 2021
Project 1

Altay Acar
2018400084

Engin Oğuzhan Şenol
2020400324

Altay Acar – 2018400084
Engin Oğuzhan Şenol – 2020400324

Table of Contents

Altay Acar – 2018400084
Engin Oğuzhan Şenol – 2020400324

THEORETICAL ANALYSIS

Basic operation is the comparison marked as (1)

Analyze B(n)

The comparison operation is executed in each iteration of the for loop `for i ← 0 to n − 1 do` So, for each possible input list of size n, the number of basic operations marked as (1) is executed by this algorithm is the same and it is n times.

- ⇨ Input is X, and X = X [0: n−1] (a list of size n)
- ⇨ $B(n) = n \in \theta(n)$

Analyze W(n)

Same as the worst-case analysis above, the comparison operation is executed in each iteration of the `for loop for i ← 0 to n − 1 do` So, for each possible input list of size n, the number of basic operations marked as (1) is executed by this algorithm is the same and it is n times. The worst-case scenario for this basic operation is not changed since it is executed without a condition.

- ⇨ Input is X, and X = X [0: n−1] (a list of size n)
- ⇨ $W(n) = n \in \theta(n)$

Analyze A(n)

$$A(n) = \sum_{B(n)}^{W(n)} i * p_i = \sum_{i=n}^{n} i * p_i$$

$$p_i = 1, \qquad for\ 1 \leq i \leq n$$

$$A(n) = \sum_{i=n}^{n} i * 1 = n$$

- ⇨ $A(n) = n \in \theta(n)$

Basic operations are the two loop incrementations marked as (2)

Analyze B(n)

For this scenario the chosen basic operations are two loop incrementations marked as (2). If the *i*th element of the input list X is 0, then the basic operation `for j ← i to n − 1 do` is executed. If the *i*th element of the input list X is 1, then the basic operation `for m ← i to n−1 do` is executed. The outer loop `for i ← 0 to n − 1 do` is executed n times. Because the loop starts from 0 and ends at n – 1. In each iteration, either one of the basic loop operations will be executed. Both loop incrementations start from i, which is incremented via outer loop until n and end at n – 1. This means that after each iteration of the outer loop, the number of basic operations executed will be decremented. When the outer loop starts to operate, *i* will be 0. That causes the basic operations will be executed n times (from 0 to n – 1). When the second iteration of outer loop starts, the basic operations will be executed n – 1 times (from 1 to n – 1). In the last iteration of the outer loop the basic operations will be executed one time (from n – 1 to n – 1) and when the outer loop's iteration is finished, the function returns y. This gives us the following result for both loop operations:

$\sum_{i=0}^{n-1} n - i = n + (n-1) + (n-2) + \cdots + 1 = \frac{n*(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$

For each iteration of outer loop either one of both basic operations will be executed. If every element of the input list X is 0, then only `for j ← i to n − 1 do` basic loop operation will be executed in each iteration of outer loop. For this case, algorithmic analysis of the given function will be $B(n) = \frac{1}{2}n^2 + \frac{1}{2}n \in \theta(n^2)$. If every element of the input list X is 1, then only `for m ← i to n−1 do` basic loop operation will be executed in each iteration of outer loop. For this case, algorithmic analysis of the given function will be $B(n) = \frac{1}{2}n^2 + \frac{1}{2}n \in \theta(n^2)$ as well. So, the best-case analysis of this algorithm follows:

- ⇨ Input is X, and X = X [0: n−1] (a list of size n)
- ⇨ $B(n) = \frac{1}{2}n^2 + \frac{1}{2}n \in \theta(n^2)$

Analyze W(n)

Just like the above calculations of best-case analysis. The algorithm will execute either one of the basic operations in each iteration of the outer for loop `for i ← 0 to n − 1 do` and that results $\frac{1}{2}n^2 + \frac{1}{2}n$ times of execution of basic operations as total independent of the value of *i*th element of the input list X. Thus, the worst-case analysis of the function is the same as the best-case analysis:

- ⇨ Input is X, and X = X [0: n−1] (a list of size n)
- ⇨ $B(n) = \frac{1}{2}n^2 + \frac{1}{2}n \in \theta(n^2)$

Analyze A(n)

$$A(n) = \sum_{B(n)}^{W(n)} i * p_i = \sum_{i=\frac{1}{2}n^2+\frac{1}{2}n}^{\frac{1}{2}n^2+\frac{1}{2}n} i * p_i$$

$$p_i = 1, \qquad for\ 1 \le i \le n$$

$$A(n) = \sum_{i=\frac{1}{2}n^2+\frac{1}{2}n}^{\frac{1}{2}n^2+\frac{1}{2}n} i * 1 = \frac{1}{2}n^2 + \frac{1}{2}n$$

- ⇨ $A(n) = \frac{1}{2}n^2 + \frac{1}{2}n \in \theta(n^2)$

Basic operation is the assignment marked as (3)

Analyze B(n)

In each iteration of outer for loop `for i ← 0 to n − 1 do` the *i*th element of the list X will be examined as follows: If this *i*th element is 0 then the basic operation of `y ← y+1` will be further executed. If the *i*th element is 1, then the basic operation of `y ← y+1` will not be executed in that iteration of outer loop. If the input list X consists only 1's, then the basic operation will not be executed for each iteration of outer loop and ultimately when the

function returns y, the basic operation would not be executed at all. This results the best-case analysis as follows:

- ⇨ Input is X, and X = X [0: n−1] (a list of size n) and $X[i] = 1\ for\ \forall i \in (0, n-1)$
- ⇨ $B(n) = 0 \in \theta(0)$ => constant complexity

Analyze W(n)

In each iteration of outer for loop `for i ← 0 to n − 1 do` the *i*th element of the list X will be examined as follows: If this *i*th element is 0 then the basic operation of `y ← y+1` will be further executed. If the *i*th element is 1, then the basic operation of `y ← y+1` will not be executed in that iteration of outer loop. If the input list X consists only 0's, then in each iteration of outer loop, the basic operation will be executed. The first for loop inside the `if X[i] = 0` condition, which is `for j ← i to n − 1 do` will be executed a total of $\frac{1}{2}n^2 + \frac{1}{2}n$ times, as calculated in the second algorithm analysis, when the *i*th element of the input list X is always 0. The second for loop, which is `for k ← n downto 1 by k ← ⌊k/2⌋ do` is nested to the first for loop. Thus, in each iteration of the first for loop, this nested for loop iterates $\lfloor log_2(n) \rfloor + 1$ times. Because the iteration starts with k being equal to n and will be decreased until it is smaller than 1. The decreasing factor is a division by 2. So, n will be divided by 2, until it is smaller than 1. For example, if $n = 19$, then this for loop will iterate 5 times ($\lfloor log_2(19) \rfloor + 1 = 4 + 1 = 5$). This gives us the following worst-case analysis:

- ⇨ Input is X, and X = X [0: n−1] (a list of size n) and $X[i] = 0\ for\ \forall i \in (0, n-1)$
- ⇨ $W(n) = \left(\frac{1}{2}n^2 + \frac{1}{2}n\right) * \left(\lfloor log_2(n) \rfloor + 1\right) = \frac{1}{2}n^2\lfloor log_2(n) \rfloor + \frac{1}{2}n^2 + \frac{1}{2}n\lfloor log_2(n) \rfloor + \frac{1}{2}n$
- ⇨ $W(n) = \frac{1}{2}n^2\lfloor log_2(n) \rfloor + \frac{1}{2}n^2 + \frac{1}{2}n\lfloor log_2(n) \rfloor + \frac{1}{2}n \in \theta(n^2 \log(n))$

Analyze A(n)

$$A(n) = \sum_{I \in T_n} \tau(I) * p(I) = B(n) * \frac{2}{3} + W(n) * \frac{1}{3}$$
$$= 0 * \frac{2}{3} + \left(\frac{1}{2}n^2\lfloor log_2(n) \rfloor + \frac{1}{2}n^2 + \frac{1}{2}n\lfloor log_2(n) \rfloor + \frac{1}{2}n\right) * \frac{1}{3}$$
$$= \frac{1}{6}n^2\lfloor log_2(n) \rfloor + \frac{1}{6}n^2 + \frac{1}{6}n\lfloor log_2(n) \rfloor + \frac{1}{6}n \in \theta(n^2 \log(n))$$

Where $\tau(I)$ is the number of basic operations for input *I* -in this case both for best and worst cases- and $p(I)$ is the probability of input *I*. Probability distribution of best-case is $\frac{2}{3}$ and probability distribution of worst-case is $\frac{1}{3}$.

Basic operations are the two assignments marked as (4)

Analyze B(n)

In the previous calculation it is found that if every element of the input list X is 0, then the algorithm executes the basic assignment operation `y ← y+1` in each iteration. This results

a complexity of $\frac{1}{2}n^2\lfloor log_2(n)\rfloor + \frac{1}{2}n^2 + \frac{1}{2}n\lfloor log_2(n)\rfloor + \frac{1}{2}n \in \theta(n^2\log(n))$. If every element of the input list X is 1, then the algorithm does not execute the first basic operation at all and in each iteration the second basic assignment operation `y ← y+1` is executed. The outer loop `for i ← 0 to n − 1 do` iterates n times. When the algorithm enters the else statement, the first for loop we come across `for m ← i to n−1 do` will be iterated $\frac{1}{2}n^2 + \frac{1}{2}n$ times as it is found in the second algorithm analysis. The nested for loop `for t ← 1 to n do` will iterate n times in each iteration of the first for loop `for m ← i to n−1 do`. So, this nested for loop will be iterated $n * \left(\frac{1}{2}n^2 + \frac{1}{2}n\right) = \frac{1}{2}n^3 + \frac{1}{2}n^2$ times total. In each iteration of this nested for loop, a nested while loop, which contains the basic operation `y ← y+1` will be executed. This while loop iterates as long as x is greater than 0, and $x = n$ and $t = 1$ in the first iteration of the nested for loop `for t ← 1 to n do`. Thus, this while loop will be iterated n times since x is decremented by one ($t = 1$) in each iteration. In the second iteration of the nested for loop `for t ← 1 to n do` $x = n$ and $t = 2$. So, in this second iteration, while loop will be iterated $\left\lceil\frac{n}{2}\right\rceil$ times since x is decremented by two ($t = 2$) in each iteration. This pattern goes until $t = n$. In this iteration, the while loop will be executed only one time since $x = n, t = n, and\ x - t = 0$. The total number of iterations for this while loop thus becomes the multiplication of n and harmonic series $\left(1 + \left\lceil\frac{1}{2}\right\rceil + \left\lceil\frac{1}{3}\right\rceil + \cdots + \left\lceil\frac{1}{n}\right\rceil\right) * n$. The calculation will follow:

⇨ $\left(\frac{1}{2}n^3 + \frac{1}{2}n^2\right) * \left(1 + \left\lceil\frac{1}{2}\right\rceil + \left\lceil\frac{1}{3}\right\rceil + \cdots + \left\lceil\frac{1}{n}\right\rceil\right) * n = \left(\frac{1}{2}n^4 + \frac{1}{2}n^3\right) * \left(1 + \left\lceil\frac{1}{2}\right\rceil + \left\lceil\frac{1}{3}\right\rceil + \cdots + \left\lceil\frac{1}{n}\right\rceil\right)$

The harmonic series $\left(1 + \left\lceil\frac{1}{2}\right\rceil + \left\lceil\frac{1}{3}\right\rceil + \cdots + \left\lceil\frac{1}{n}\right\rceil\right)$ grows with log(n) complexity. Thus, to simplify the calculation:

⇨ $\left(\frac{1}{2}n^4 + \frac{1}{2}n^3\right) * \log(n) = \frac{1}{2}n^4 * \log(n) + \frac{1}{2}n^3 * \log(n)$

Concluding this calculation, the total number of basic operations executed when the input is a list of all 1's is much greater than it being a list of all 0's:

⇨ $\frac{1}{2}n^4 * \log(n) + \frac{1}{2}n^3 * \log(n) > \frac{1}{2}n^2\lfloor log_2(n)\rfloor + \frac{1}{2}n^2 + \frac{1}{2}n\lfloor log_2(n)\rfloor + \frac{1}{2}n$

The best-case scenario of this algorithm with the two basic assignment operations follows:

⇨ Input is X, and X = X [0: n−1] (a list of size n) and $X[i] = 0\ for\ \forall i \in (0, n-1)$
⇨ $B(n) = \frac{1}{2}n^2\lfloor log_2(n)\rfloor + \frac{1}{2}n^2 + \frac{1}{2}n\lfloor log_2(n)\rfloor + \frac{1}{2}n \in \theta(n^2\log(n))$

Analyze W(n)

As calculated above in the best-case analysis. When the input list X consists of all 1's, $\frac{1}{2}n^4 * \log(n) + \frac{1}{2}n^3 * \log(n)$ number of basic operations are executed. When the input list X consists of all 0's $\frac{1}{2}n^2\lfloor log_2(n)\rfloor + \frac{1}{2}n^2 + \frac{1}{2}n\lfloor log_2(n)\rfloor + \frac{1}{2}n$ number of basic operations are executed. Since the total number of basic operations executed when the input is a list of all 1's is much greater than it being a list of all 0's:

⇨ $\frac{1}{2}n^4 * log_2(n) + \frac{1}{2}n^3 * log_2(n) > \frac{1}{2}n^2\lfloor log_2(n)\rfloor + \frac{1}{2}n^2 + \frac{1}{2}n\lfloor log_2(n)\rfloor + \frac{1}{2}n$

The worst-case scenario of this algorithm with the two basic assignment operations follows:

⇨ Input is X, and X = X [0: n−1] (a list of size n) and $X[i] = 1\ for\ \forall i \in (0, n-1)$

⇨ $W(n) = \frac{1}{2}n^4 * \log(n) + \frac{1}{2}n^3 * \log(n) \in \theta(n^4 * \log(n))$

Analyze A(n)

$$A(n) = \sum_{I \in T_n} \tau(I) * p(I) = B(n) * \frac{1}{3} + W(n) * \frac{2}{3}$$
$$= \left(\frac{1}{2}n^2\lfloor log_2(n)\rfloor + \frac{1}{2}n^2 + \frac{1}{2}n\lfloor log_2(n)\rfloor + \frac{1}{2}n\right) * \frac{1}{3}$$
$$+ \left(\frac{1}{2}n^4 * \log(n) + \frac{1}{2}n^3 * \log(n)\right) * \frac{2}{3}$$
$$= \frac{1}{6}n^2\lfloor log_2(n)\rfloor + \frac{1}{6}n^2 + \frac{1}{6}n\lfloor log_2(n)\rfloor + \frac{1}{6}n + \frac{1}{3}n^4 * \log(n) + \frac{1}{3}n^3 * \log(n)$$
$$= \frac{1}{3}n^4 * \log(n) + \frac{1}{3}n^3 * \log(n) + \frac{1}{6}n^2\lfloor log_2(n)\rfloor + \frac{1}{6}n^2 + \frac{1}{6}n\lfloor log_2(n)\rfloor + \frac{1}{6}n$$
$$\in \theta(n^4 * \log(n))$$

Where $\tau(I)$ is the number of basic operations for input *I* -in this case both for best and worst cases- and $p(I)$ is the probability of input *I*. Probability distribution of best-case is $\frac{2}{3}$ and probability distribution of worst-case is $\frac{1}{3}$.

IDENTIFICATION OF BASIC OPERATION(S)

The basic operation is the operation that has the most influence on the algorithm's total running time and it is the operation that characterizes the efficiency of that algorithm. From our analyses in the first section "THEORETICAL ANALYSIS," the basic operation of two assignment operations `y ← y+1`, which was marked as (4) in the provided pseudo-code is the most dominant operation that has the most influence in the running time of the algorithm since its number of executions for the best-and-worst-case are the highest compared to the other operations.

REAL EXECUTION

Best Case

N Size	Time Elapsed
1	2.86102294922e-06
10	2.21729278564e-05
50	0.00056004524231
100	0.00208306312561

200	0.00879621505737
300	0.0260651111603
400	0.0402271747589
500	0.0685570240021
600	0.111665964127
700	0.150266885757

Worst Case

N Size	Time Elapsed
1	2.86102294922e-06
10	0.000124931335449
50	0.0158112049103
100	0.114341020584
200	1.02890515327
300	3.72343015671
400	9.05582499504
500	18.3324861526
600	31.6729500294
700	55.9882860184

Average Case

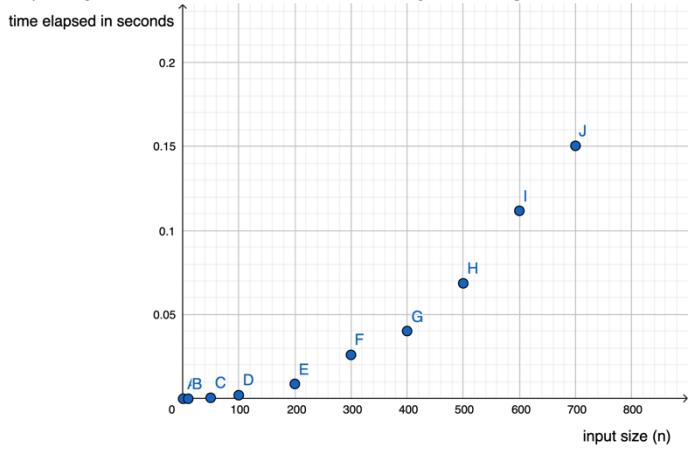
N Size	Time Elapsed
1	1.81198120117e-06
10	8.41617584229e-05
50	0.00974955558777
100	0.0743873119354
200	0.676896429062
300	2.3643535614
400	5.76846017838
500	12.3754524231
600	21.861568594
700	37.1249567986

Altay Acar – 2018400084
Engin Oğuzhan Şenol – 2020400324

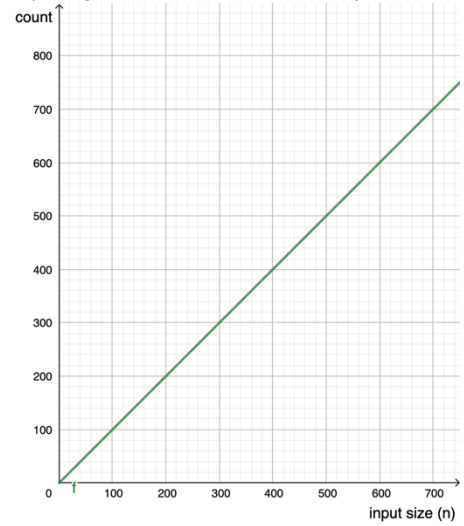
COMPARISON

Best Case

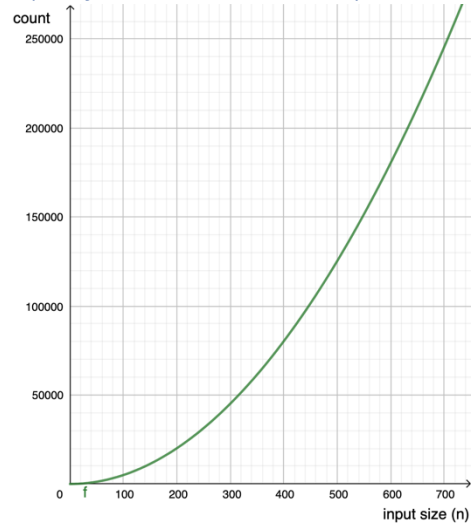
Graph of the real execution time of the algorithm



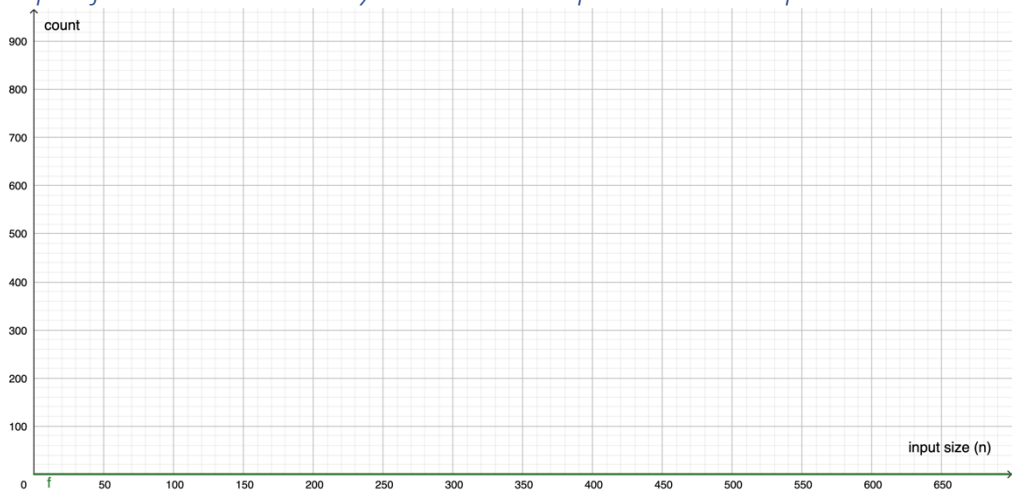
Graph of the theoretical analysis when basic operation is the operation marked as (1)



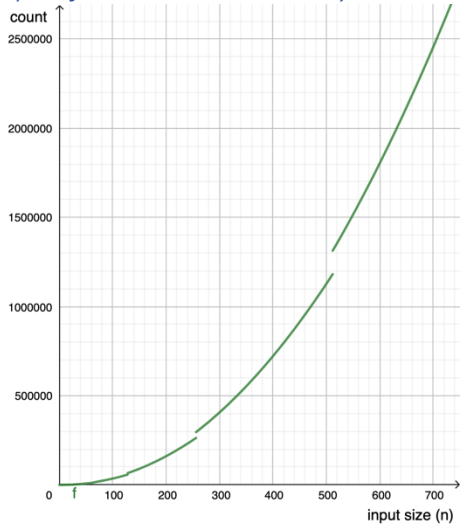
Graph of the theoretical analysis when basic operation is the operation marked as (2)



Graph of the theoretical analysis when basic operation is the operation marked as (3)



Graph of the theoretical analysis when basic operation is the operation marked as (4)



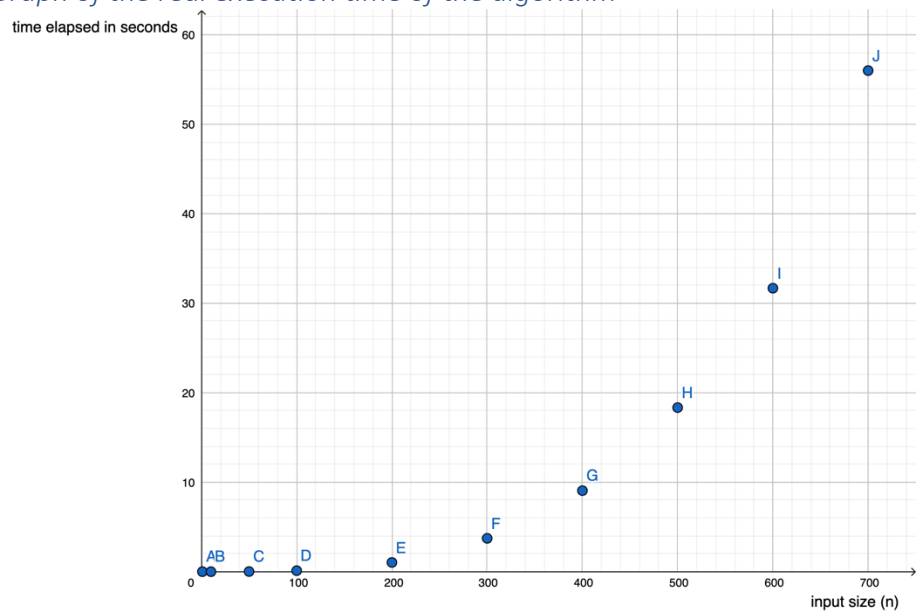
Comments

Best-case graph of the real execution time of the algorithm resembles an exponential growth in the terms of form of the function graph. Y-axis provides us the elapsed time in seconds while X-axis provides us the total size of the input, which is denoted by n. In terms of real execution, the total elapsed time is very small even the input size is as big as 700. Because this analysis examines the best-case behavior of the graph. The theoretical analysis of the basic operation (1) resulted a first-degree polynomial graph. It is growing but not as fast as an exponential growth. While the growth rate of the graph increases exponentially in the real execution, it stays same for the graph of basic operation (1). So, we can say that this basic operation does not resemble our algorithm well. Same can be said for the basic operation (3). Its theoretical execution graph shows a non-growing function of x=0, which gives a constant complexity for our algorithm, although it should be near exponential. Among the two other graphs (2) and (4), the one for the theoretical execution of basic operation (4) is the most similar graph for our algorithm. Because, although the real execution graph is like the exponential growth, it is not a perfect exponential growth. The growth rate of the function varies not like how a normal exponential function grows. The growth rate of the growth rate of the function differs in intervals. For the graph of (3) it is a normal exponential function behavior. Thus, it can be said that the basic operation of two

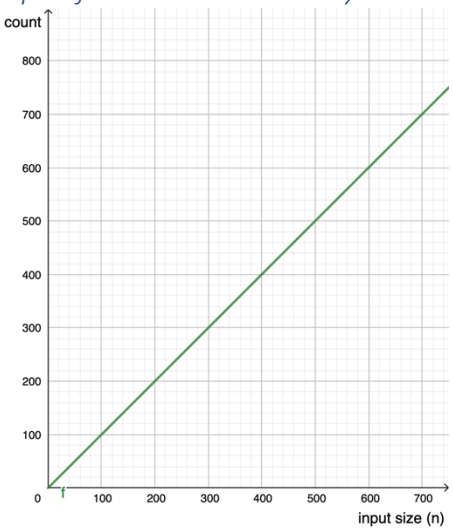
assignment operations in (4) is the correct choice of basic operation for the algorithm analysis of time complexity for the given function.

Worst Case

Graph of the real execution time of the algorithm

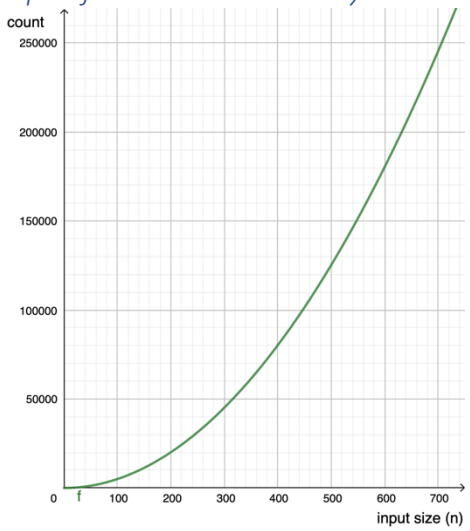


Graph of the theoretical analysis when basic operation is the operation marked as (1)

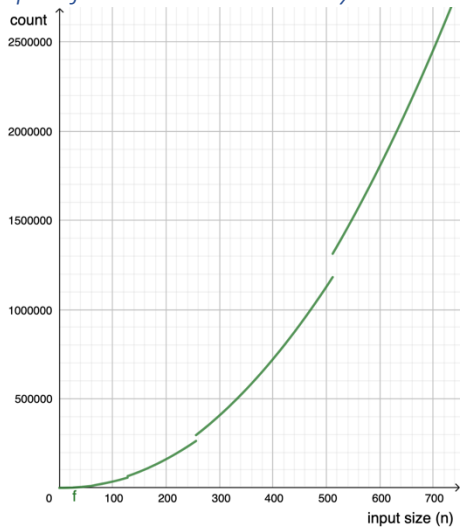


Altay Acar – 2018400084
Engin Oğuzhan Şenol – 2020400324

Graph of the theoretical analysis when basic operation is the operation marked as (2)

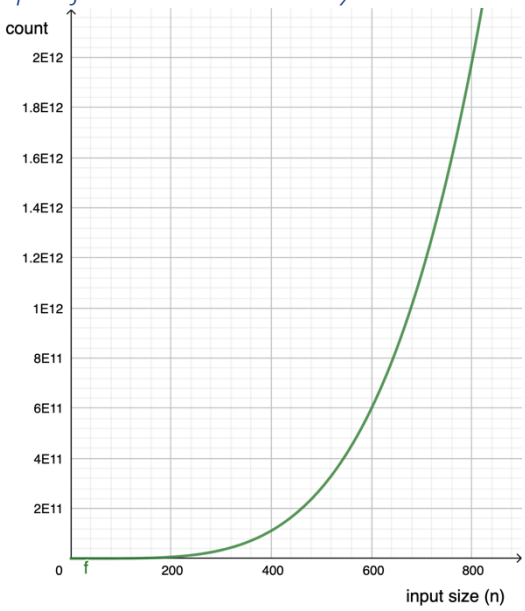


Graph of the theoretical analysis when basic operation is the operation marked as (3)



Altay Acar – 2018400084
Engin Oğuzhan Şenol – 2020400324

Graph of the theoretical analysis when basic operation is the operation marked as (4)



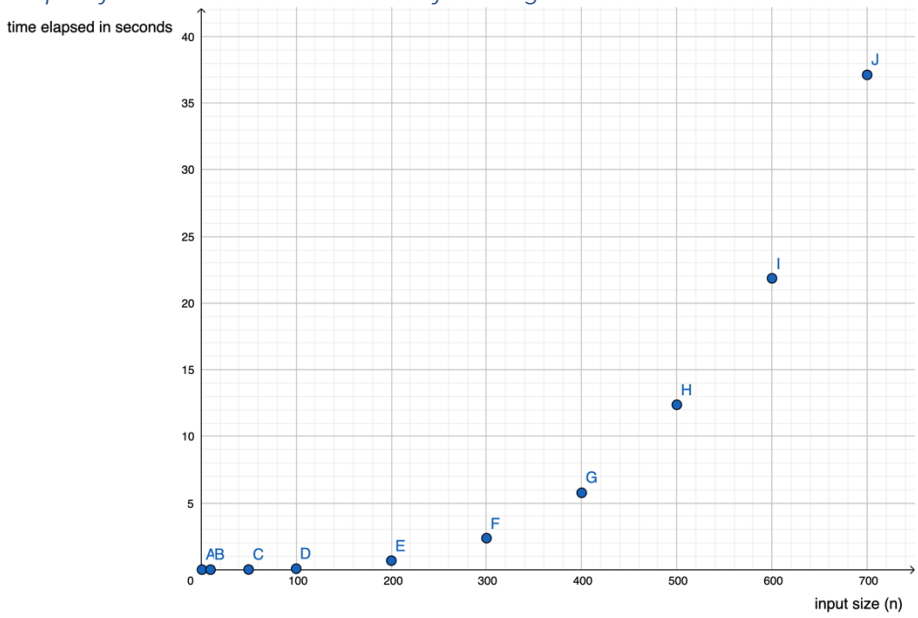
Comments

Worst-case graph of the real execution time of the algorithm resembles again an exponential growth in the terms of form of the function graph just like the best-case graph did. Y-axis provides us the elapsed time in seconds while X-axis provides us the total size of the input, which is denoted by n. In terms of real execution, the total elapsed time is considerably high when it is compared to the best-case execution. Because this analysis examines the worst-case scenario of the function, meaning the highest possible elapsed time that this function results for a given input size. Although it almost takes 0 seconds to execute for smaller input sizes like 1 to 100, for an input size of 700 it takes approximately 55 seconds. The graph of theoretical analysis when the basic operation is (1) belongs to a first-degree polynomial function. Thus, it is not convenient to choose (1) as basic operation. Our real execution graph grows higher than a polynomial growth. Graph of the basic operation (3) shows an exponential growth. However, because of the floor operations in the function the growth rate of the growth rate of the function varies in intervals. On the other hand, our real execution graph illustrates a normal exponential graph behavior. So, it is not convenient to choose (3) as the basic operation as well. Two other graphs of theoretical analysis (2) and (4) resembles exponential growth but differs in the growth rate. The graph of basic operation (4) does not grow fast until the input size of 200. But then it grows fast while the graph of (2) grows slow until approximately 50 and fast after that. Our real execution graph is very similar to the graph of basic operation of two assignment operations (4). It also grows relatively small until the input size of 200 and then much faster just like (4). Thus, the basic operation of two assignment operations in (4) is chosen as the correct choice of basic operation for the algorithm analysis of time complexity for the given function.

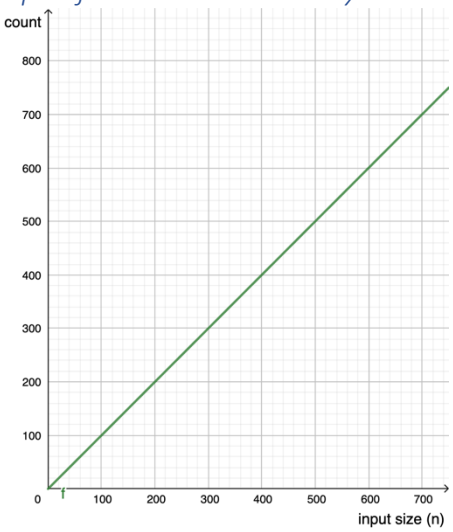
Altay Acar – 2018400084
Engin Oğuzhan Şenol – 2020400324

Average Case

Graph of the real execution time of the algorithm

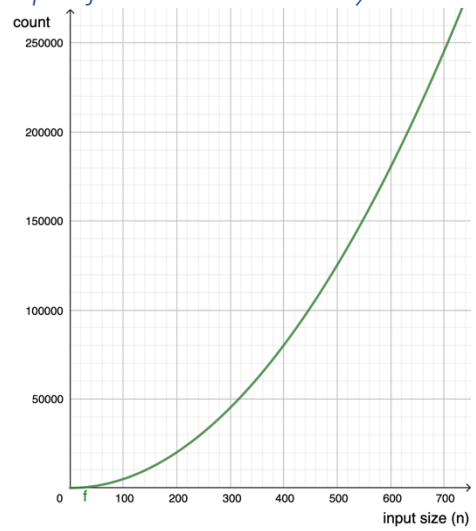


Graph of the theoretical analysis when basic operation is the operation marked as (1)

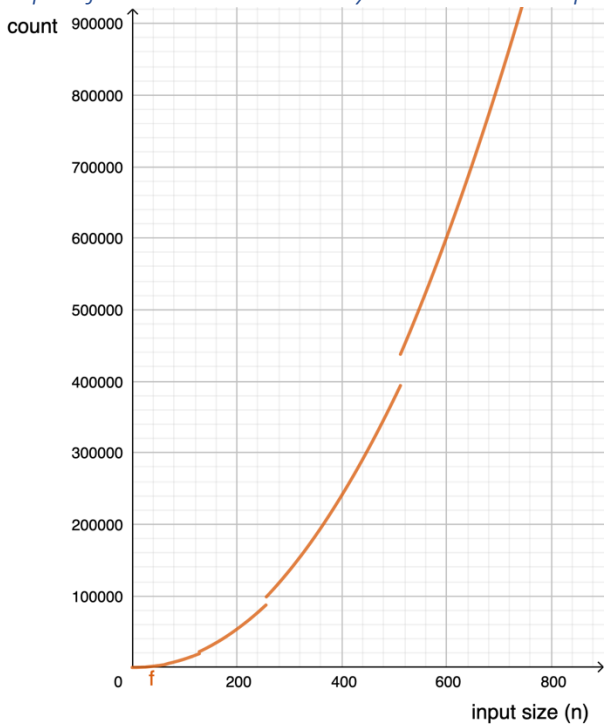


Altay Acar – 2018400084
Engin Oğuzhan Şenol – 2020400324

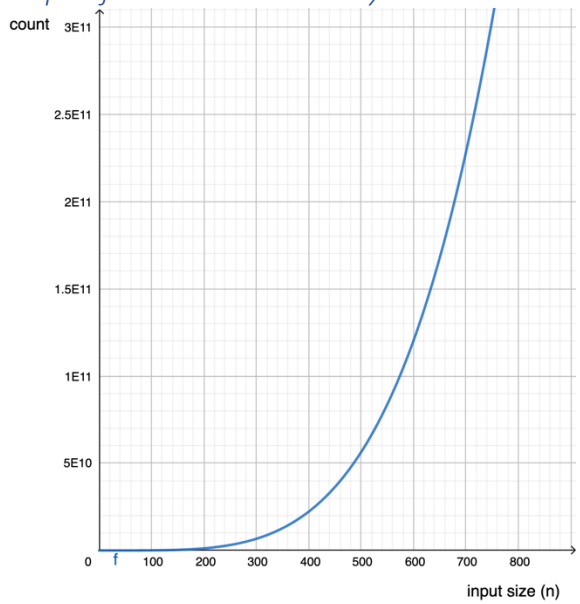
Graph of the theoretical analysis when basic operation is the operation marked as (2)



Graph of the theoretical analysis when basic operation is the operation marked as (3)



Graph of the theoretical analysis when basic operation is the operation marked as (4)



Comments

Average-case graph of the real execution time of the algorithm resembles again an exponential growth in the terms of form of the function graph just like the best-case and worst-case graph did. Y-axis provides us the elapsed time in seconds while X-axis provides us the total size of the input, which is denoted by n. In terms of real execution, the total elapsed time is considerably high when it is compared to the best-case execution and relatively low when it is compared to the worst-case execution. Because this analysis examines the average-case scenario of the function based on the five different executions with five various inputs of the same size. Although it almost takes 0 seconds to execute for smaller input sizes like 1 to 100, for an input size of 700 it takes approximately 37 seconds as average. The graph of theoretical analysis when the basic operation is (1) belongs to a first-degree polynomial function. Thus, it is not convenient to choose (1) as basic operation. Our real execution graph grows higher than a polynomial growth. Graph of the basic operation (3) shows an exponential growth. However, because of the floor operations in the function the growth rate of the growth rate of the function varies in intervals. On the other hand, our real execution graph illustrates a normal exponential graph behavior. So, it is not convenient to choose (3) as the basic operation as well. Two other graphs of theoretical analysis (2) and (4) resembles exponential growth but differs in the growth rate. The graph of basic operation (4) does not grow fast until the input size of 200. But then it grows fast while the graph of (2) grows slow until approximately 50 and fast after that. Our real execution graph is very similar to the graph of basic operation of two assignment operations (4). It also grows relatively small until the input size of 200 and then much faster just like (4). Thus, the basic operation of two assignment operations in (4) is chosen as the correct choice of basic operation for the algorithm analysis of time complexity for the given function.