Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

Table of Contents

Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

THEORETICAL ANALYSIS

Basic operation is the comparison marked as (1)

Analyze B(n) = n,
B(n) ∈ Θ(n)
Analyze W(n) = n,
W(n) ∈ Θ(n)
Analyze A(n) = n,
A(n) ∈ Θ(n)

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

Analyze B(n) = (n*(n+1))/2,
B(n) ∈ Θ (n^2)
Analyze W(n) = (n*(n+1))/2,
W(n) ∈ Θ (n^2)
Analyze A(n) = (n*(n+1))/2,
A(n) ∈ Θ (n^2)

Basic operation is the assignment marked as (3)

Analyze B(n) = 0,
B(n) ∈ Θ (1)
Analyze W(n) = ((n*(n+1))/2) *$\lfloor \log_2(2n) \rfloor$,
W(n) ∈ Θ (n^2 *log(n))
Analyze A(n) = (((n*(n+1))/2) *$\lfloor \log_2(2n) \rfloor$)/3,
A(n) ∈ Θ (n^2 *log(n))

Basic operations are the two assignments marked as (4)

Analyze B(n) = ((n*(n+1))/2) *$\log_2(2n)$,
B(n) ∈ Θ (n^2 *log(n))
Analyze W(n) = $(\sum_{x=1}^{n} \lceil n/x \rceil)$*((n*(n+1))/2),
W(n) ∈ Θ (n^3 *log(n))
Analyze A(n) = (((n*(n+1))/2) *$\log_2(2n)$)/3 + 2*$(\sum_{x=1}^{n} \lceil n/x \rceil)$*((n*(n+1))/2))/3,
A(n) ∈ Θ (n^3 *log(n))

IDENTIFICATION OF BASIC OPERATION(S)

Here, state clearly which operation(s) in the algorithm must be the basic operation(s). Also, you should provide a simple explanation about why you have decided on the basic operation you choose. (1-3 sentences)

Since basic operation (4), which is y = y+1, is in the both part of if-else statement, it executes for any member in the array. Also, since it's in the most inner loop, it is the most executed operation in the algorithm. That's why it is the most convenient one for investigating the execution time of the algorithm.

Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

REAL EXECUTION

Best Case

N Size	Time Elapsed in Seconds
1	9.8967431640625e-07
10	1.4066696166992188e-05
50	0.0011229515075683594
100	0.0016970634460449219
200	0.0071239471435546875
300	0.01915121078491211
400	0.033731937408447266
500	0.05326986312866211
600	0.08579785919189453
700	0.11757605285644531

Worst Case

N Size	Time Elapsed in Seconds
1	2.9457672119140625e-06
10	0.00011873245239257812
50	0.0476129150390625
100	0.21271395683288574
200	1.203434944152832
300	4.439490079879761
400	11.213199100494385
500	22.745553827285767
600	40.85364890098572
700	66.96775097846985

Average Case

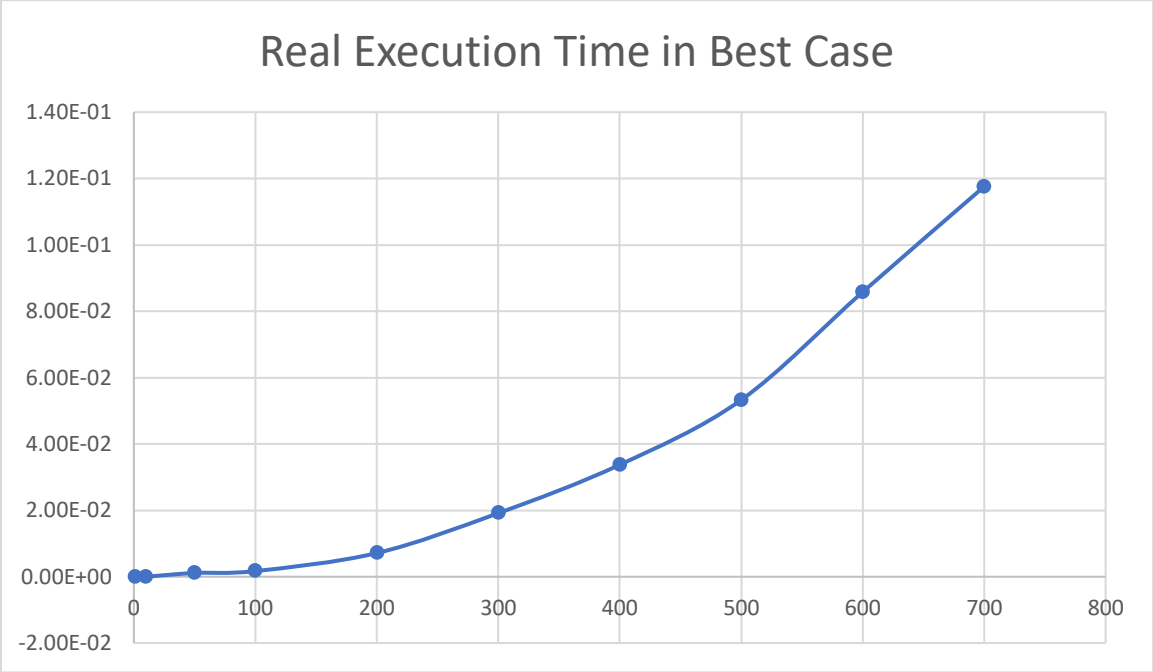
N Size	Time Elapsed in Seconds
1	2.1073486328125e-06
10	9.703636169433594e-05
50	0.016276836395263672
100	0.1217582130432129
200	0.9279918766021729
300	3.041199207305908
400	7.241055011749268
500	15.209041833877563
600	27.34042205810547
700	43.420200872421265

Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

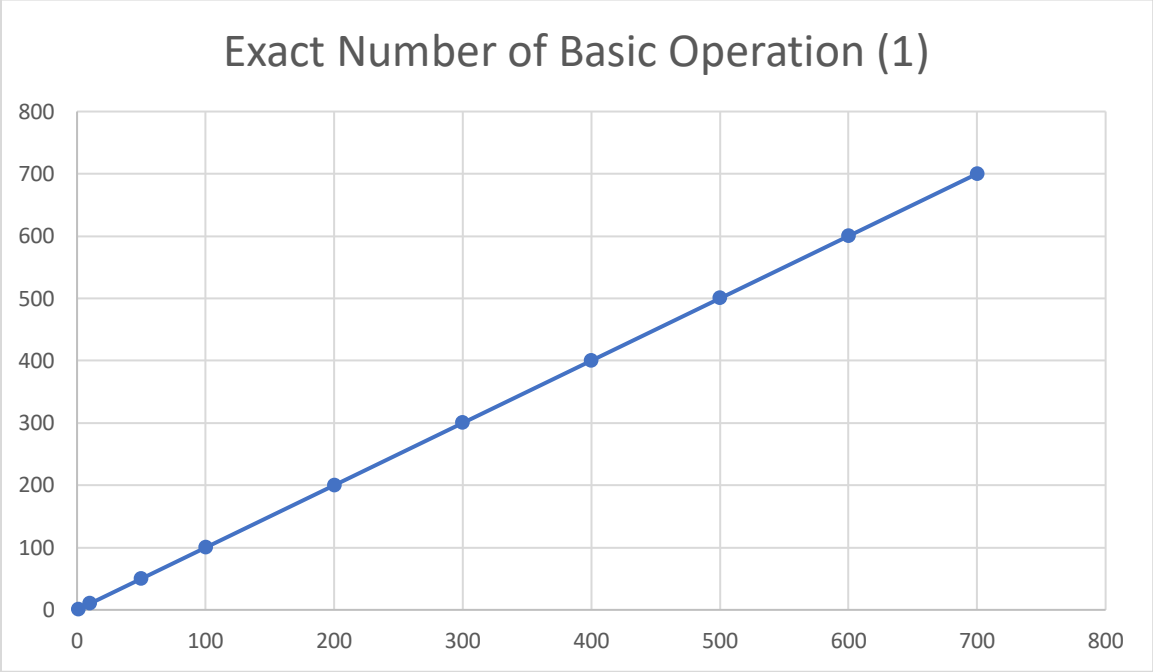
COMPARISON

Best Case

Graph of the real execution time of the algorithm

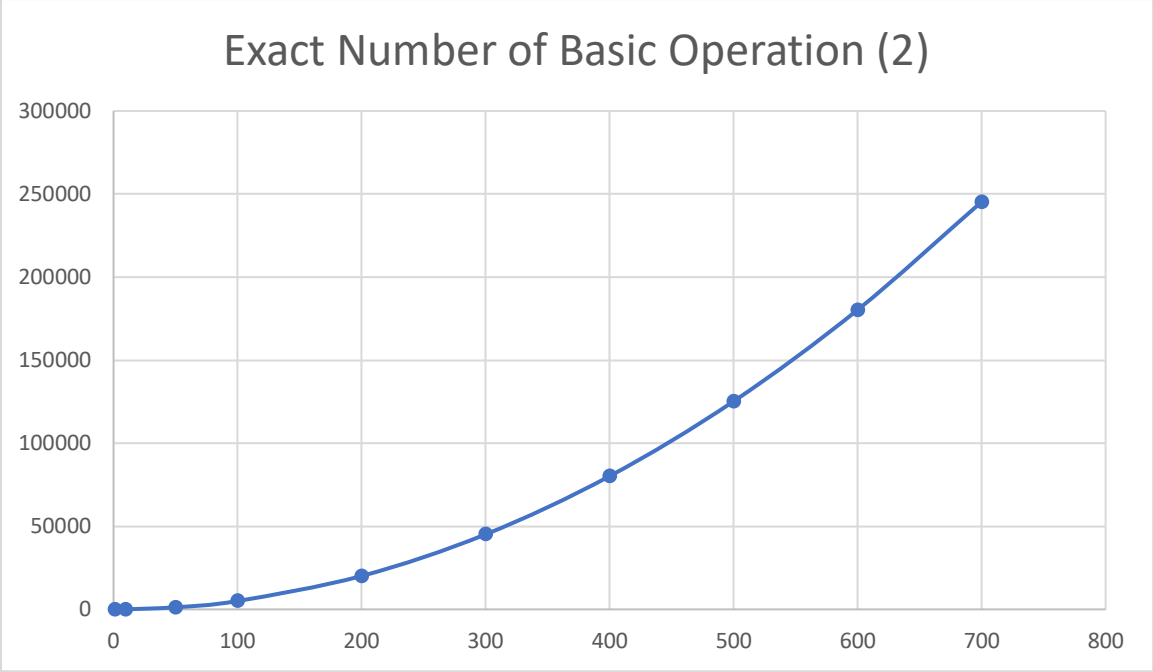


Graph of the theoretical analysis when basic operation is the operation marked as (1)

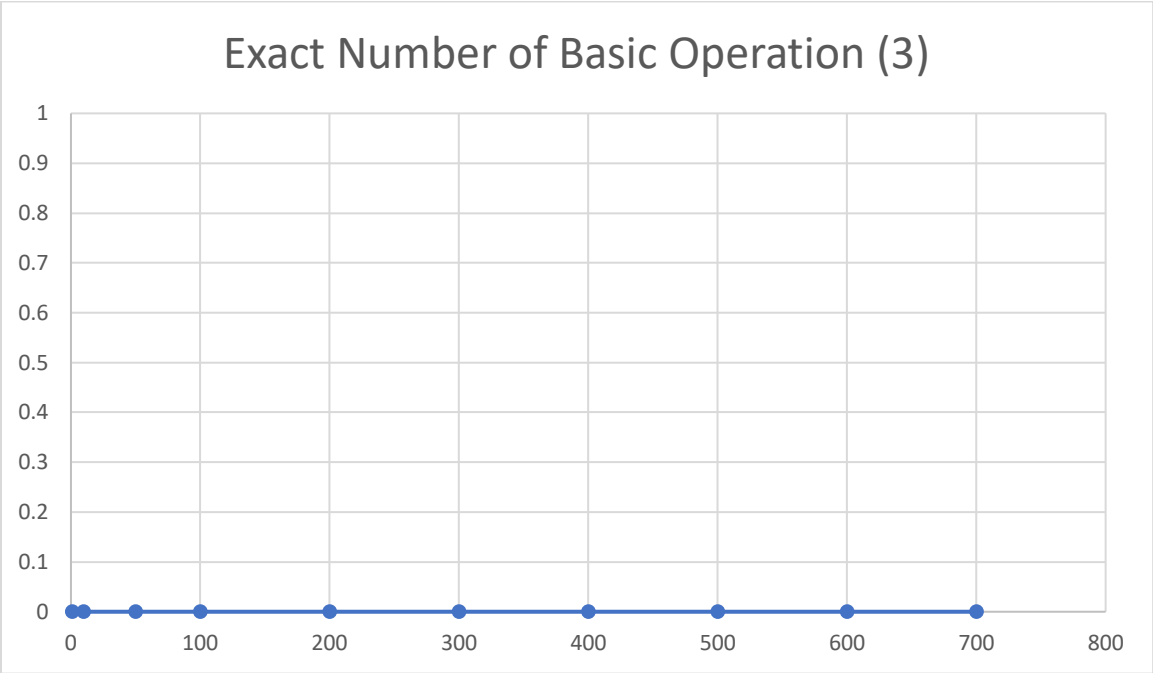


Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

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

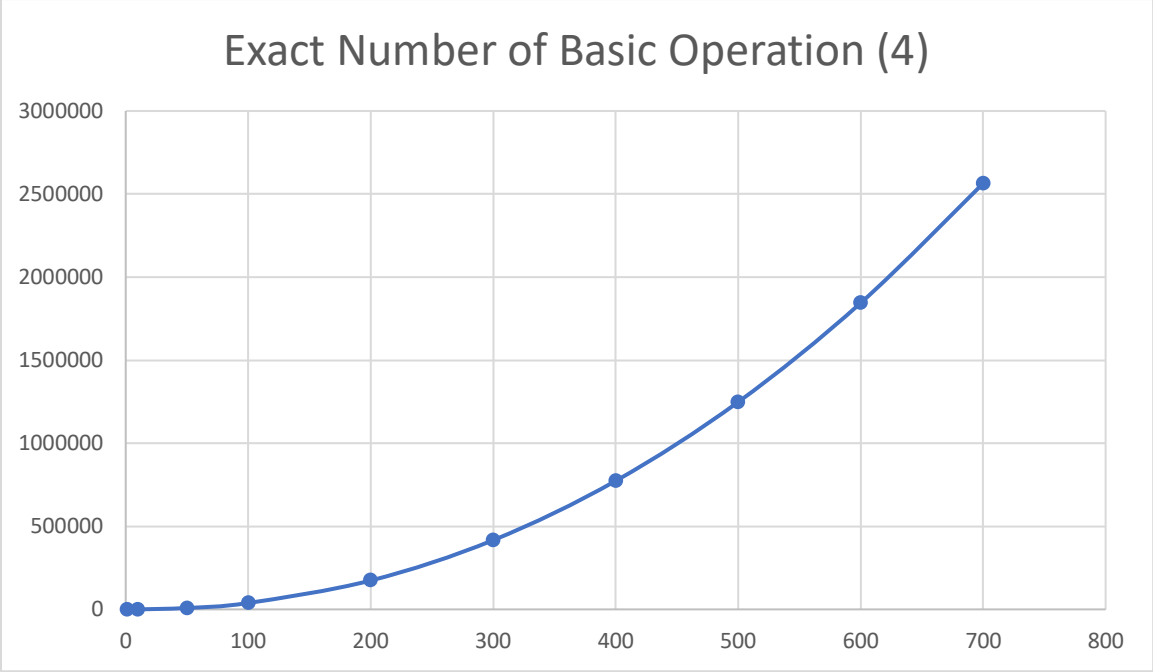


Graph of the theoretical analysis when basic operation is the operation marked as (3)



Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

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



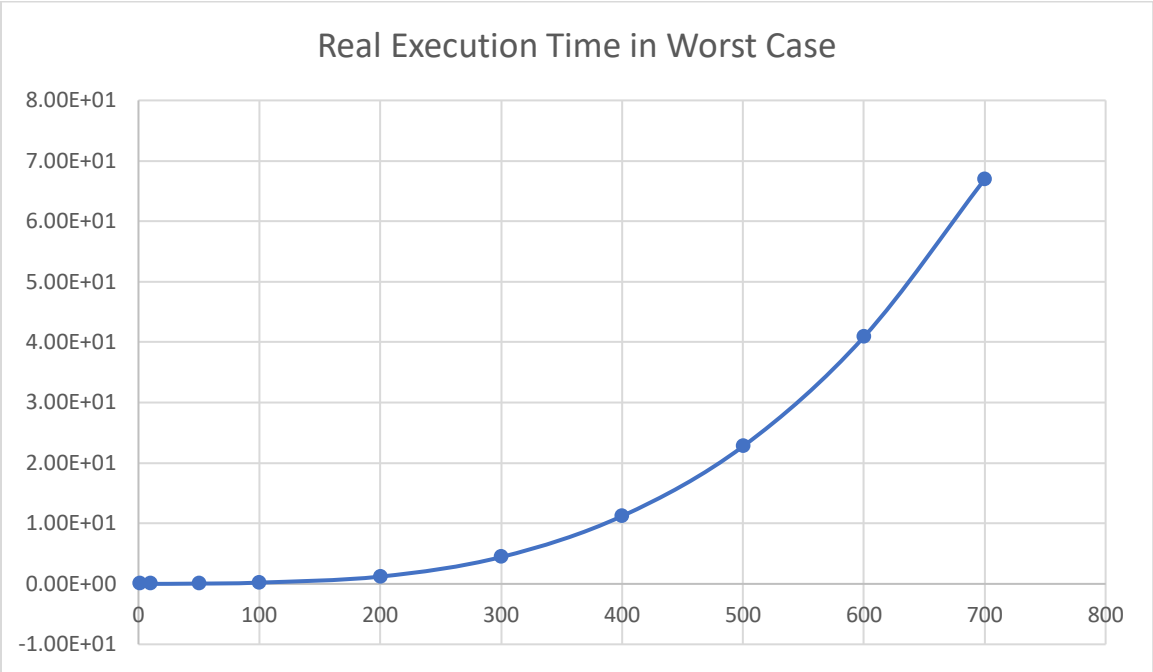
Comments

- For operation (1), we think that it doesn't matter ratio of 0 or 1 in the array, since the comparison operation will be done for each element regardless of whether it is 0 or 1. Therefore, complexity is the same for best, worst, and average case.
- For operation (2), we think that it doesn't matter ratio of 0 or 1 in the array, since the operation (2) is in the both part of if-else statement. So, this operation will be done for each element regardless of whether it is 0 or 1. Therefore, complexity is the same for best, worst, and average case.
- If we fill array with all 1's, operation (3) will be never executed. So, it will be the case in the best-case scenario.
- We have 2 possible complexities in operation (4). If we execute the if part, complexity is ((n*(n+1))/2) *$\lfloor \log_2(2n) \rfloor$, on the other hand, complexity is $(\sum_{x=1}^{n} \lceil n/x \rceil)$*((n*(n+1))/2) in the else part. Since the latter one higher complexity, in the best-case scenario, we need to execute always if part. Therefore, we fill the array with all 0's.

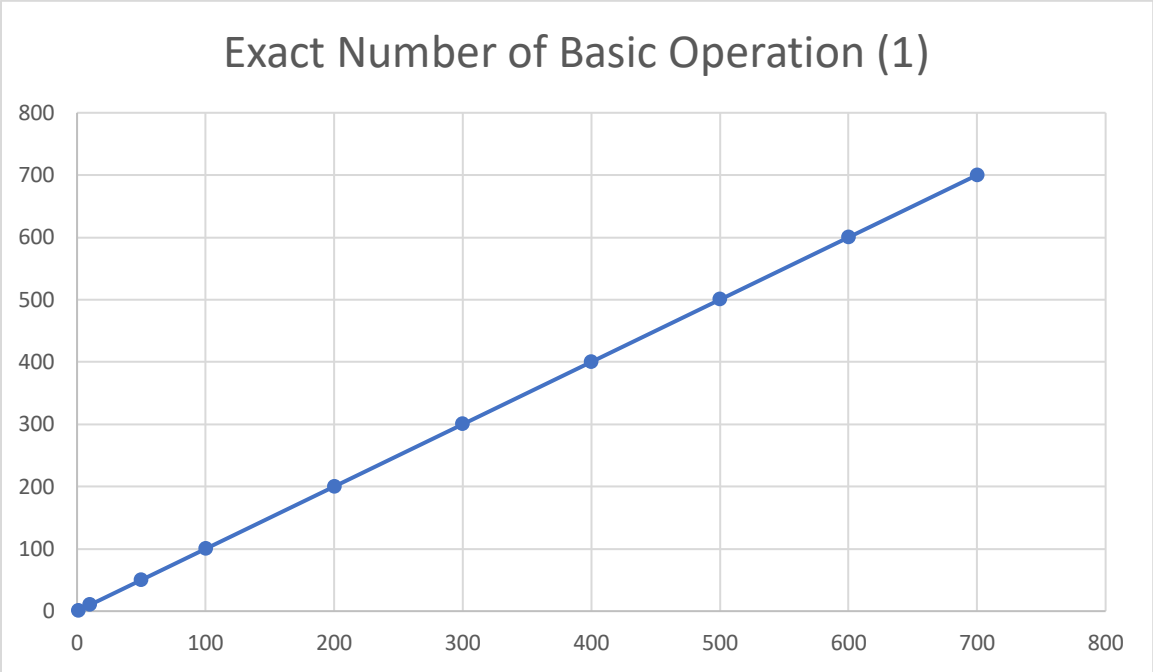
Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

Worst Case

Graph of the real execution time of the algorithm

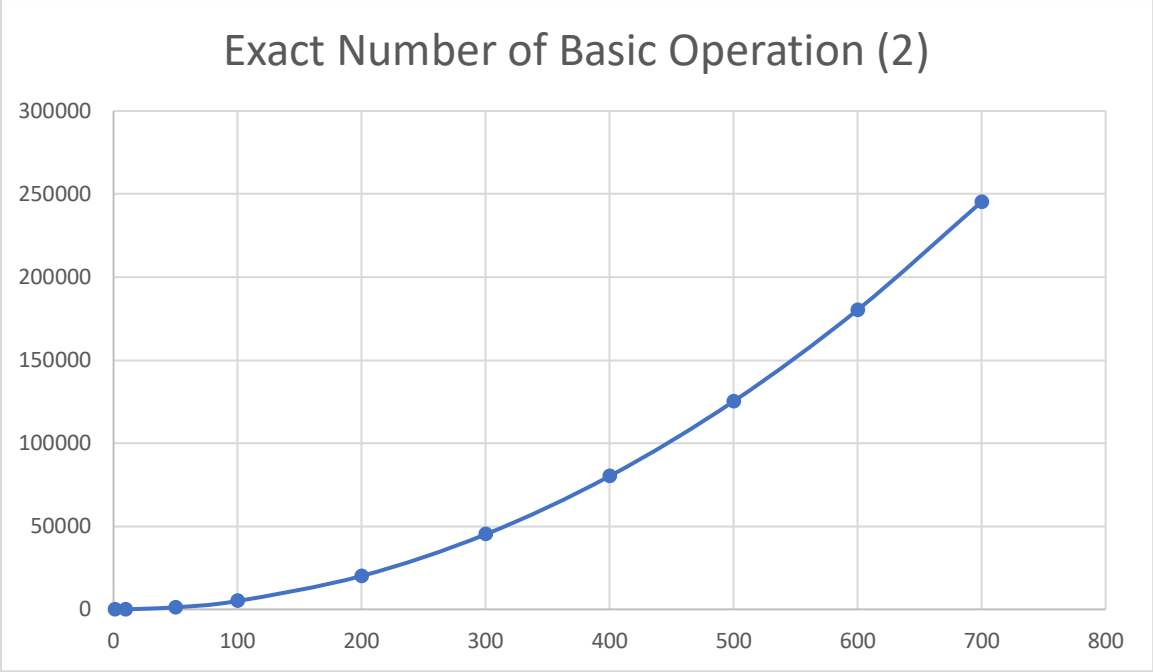


Graph of the theoretical analysis when basic operation is the operation marked as (1)

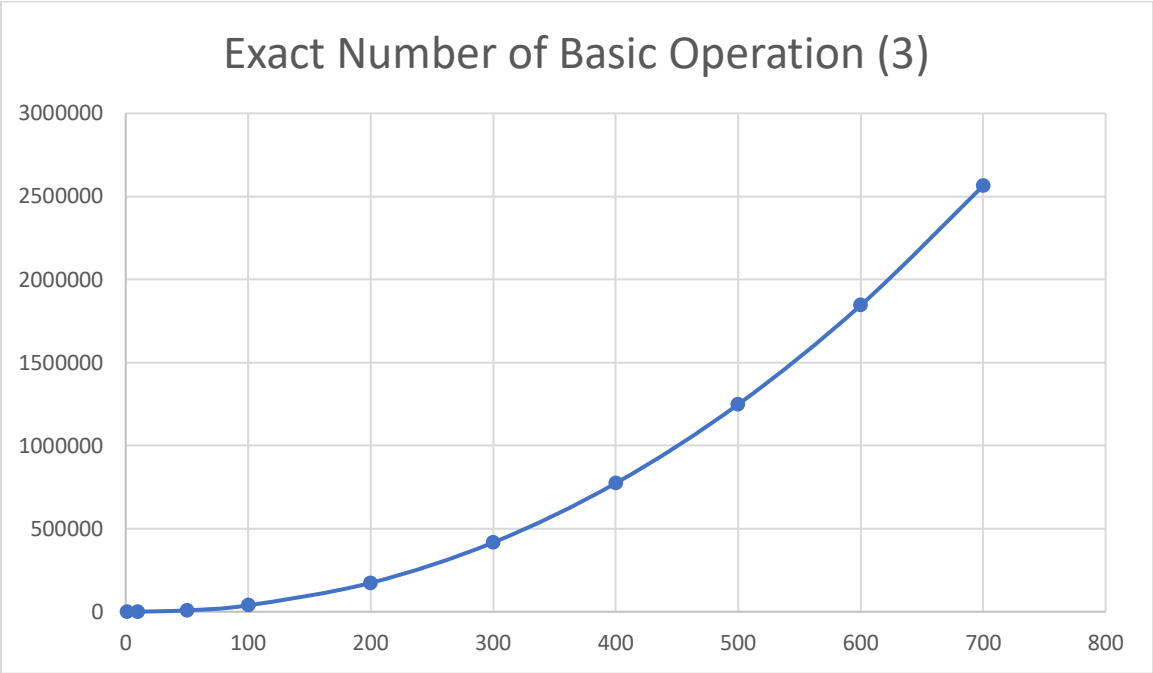


Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

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

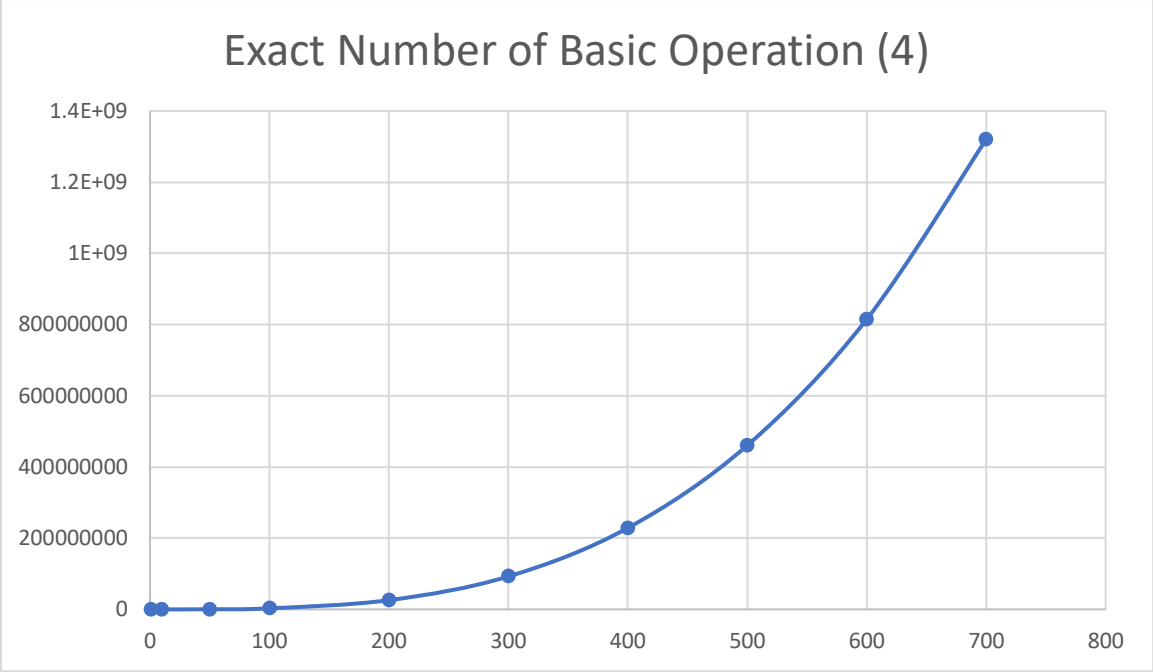


Graph of the theoretical analysis when basic operation is the operation marked as (3)



Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

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



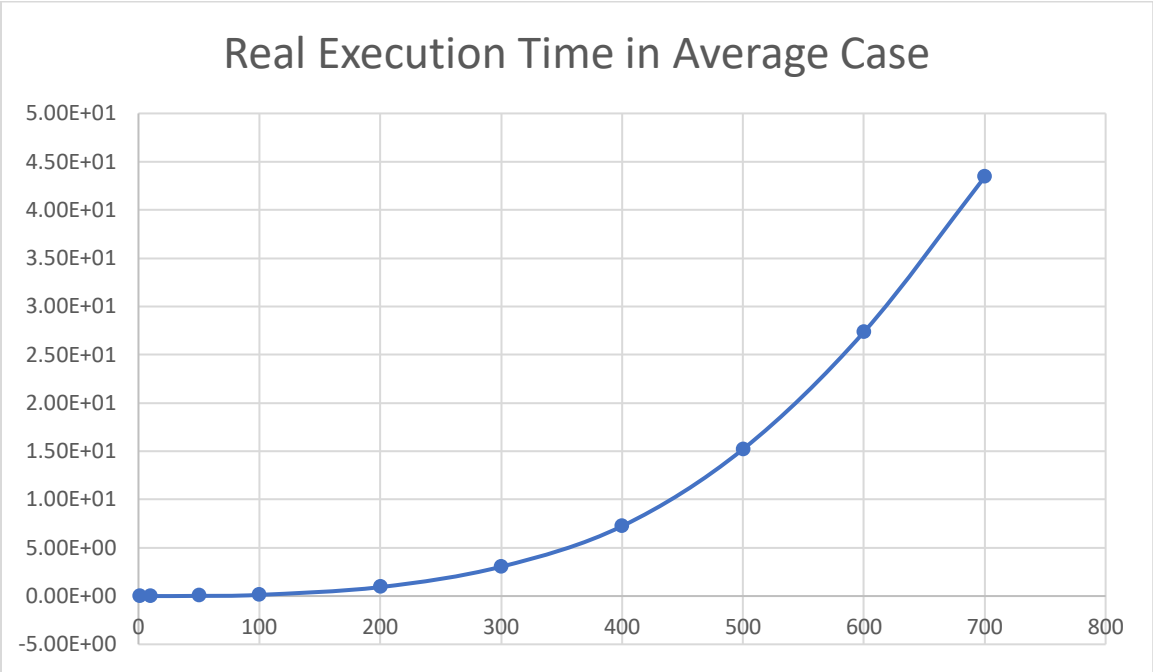
Comments

- For operation (1), we think that it doesn't matter ratio of 0 or 1 in the array, since the comparison operation will be done for each element regardless of whether it is 0 or 1. Therefore, complexity is the same for best, worst, and average case.
- For operation (2), we think that it doesn't matter ratio of 0 or 1 in the array, since the operation (2) is in the both part of if-else statement. So, this operation will be done for each element regardless of whether it is 0 or 1. Therefore, complexity is the same for best, worst, and average case.
- If we fill array with all 0's, operation (3) will be executed for each element of array, in other words as much as possible, with complexity ((n*(n+1))/2) *⌊$\log_2(2n)$⌋. So, it will be case in the worst-case scenario.
- We have 2 possible complexities in operation (4). If we execute the if part, complexity is ((n*(n+1))/2) *⌊$\log_2(2n)$⌋, on the other hand, complexity is $(\sum_{x=1}^{n}\lceil n/x \rceil)$*((n*(n+1))/2) in the else part. Since the latter one higher complexity, in the worst-case scenario, we need to execute always else part. Therefore, we fill the array with all 1's.

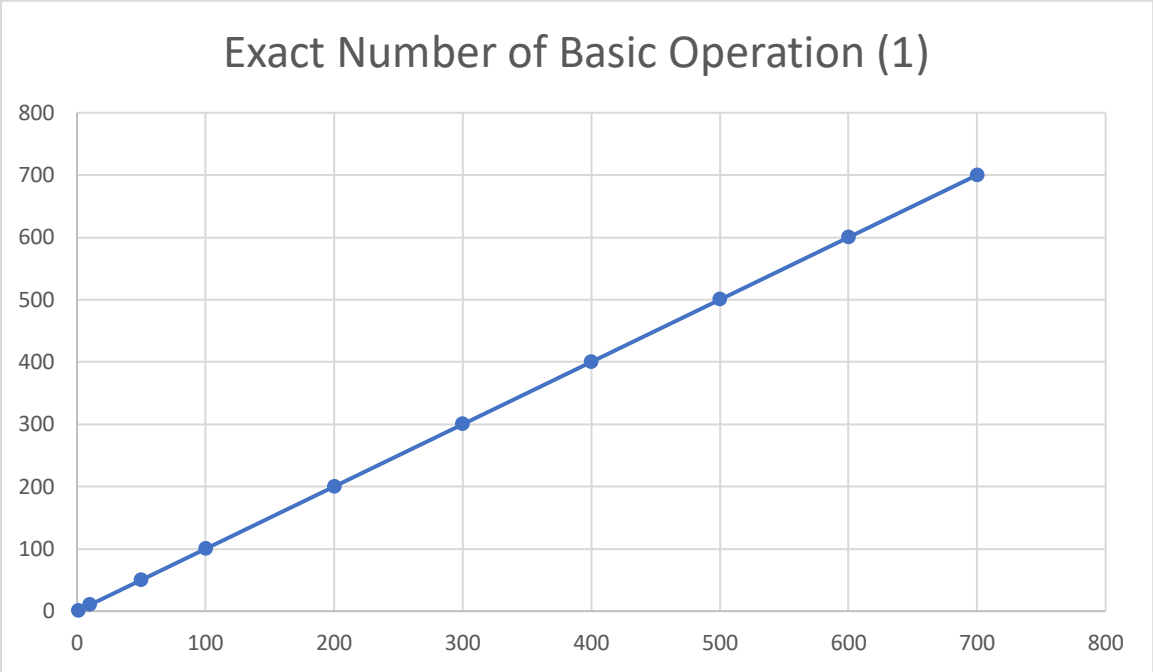
Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

Average Case

Graph of the real execution time of the algorithm

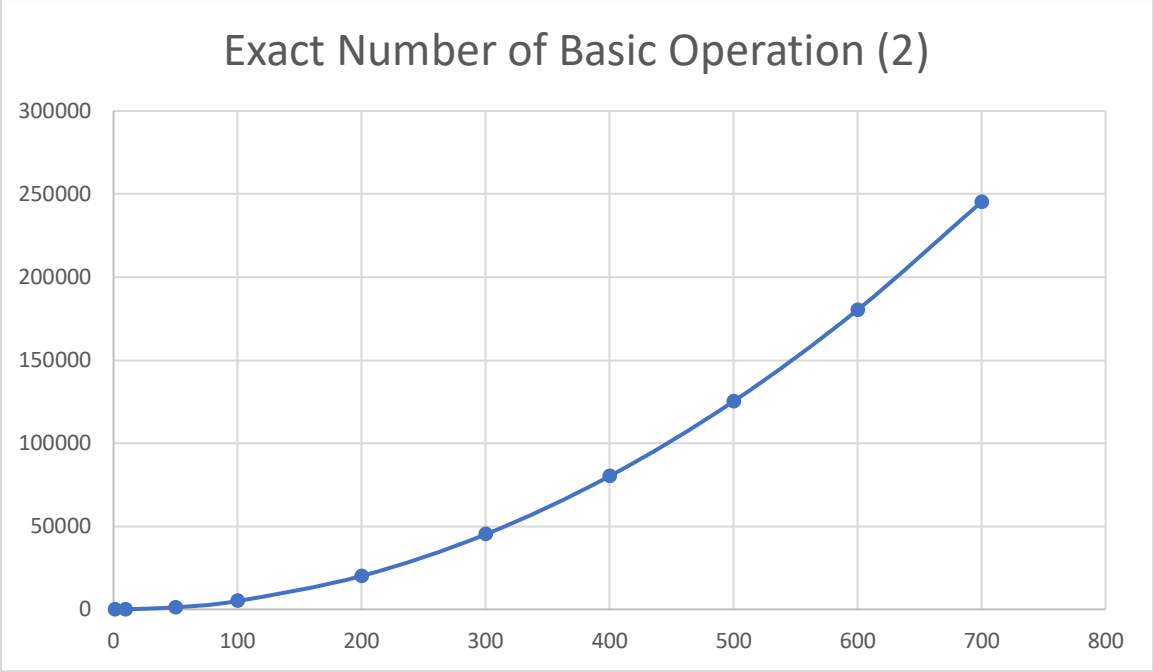


Graph of the theoretical analysis when basic operation is the operation marked as (1)

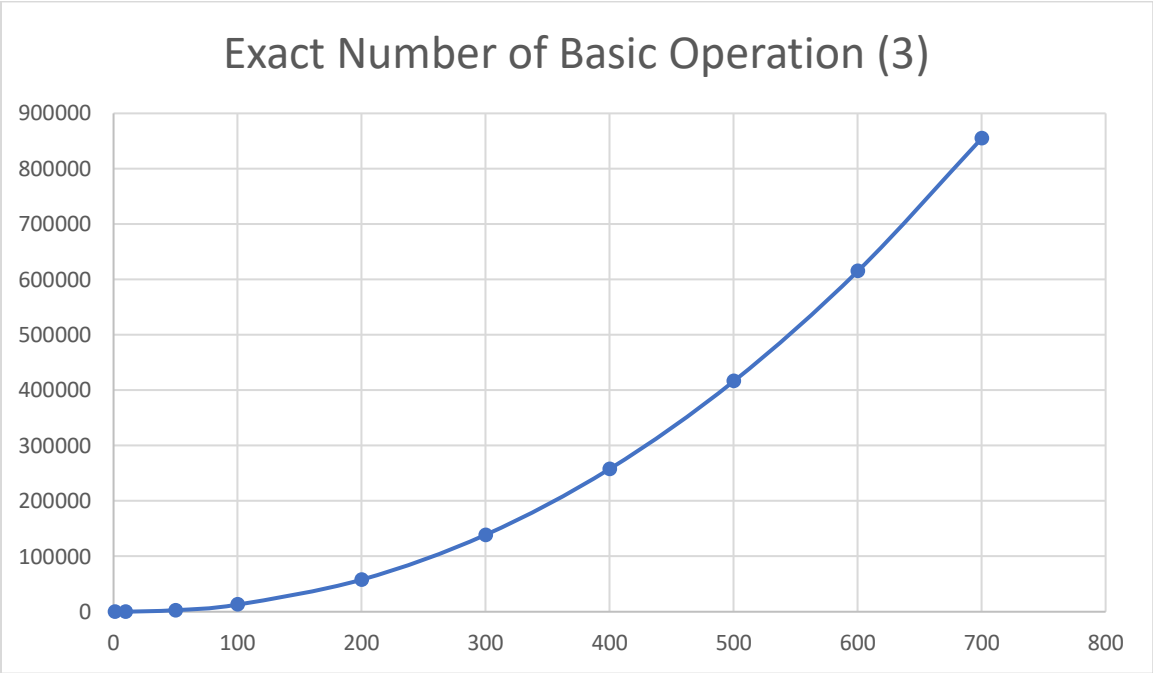


Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

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

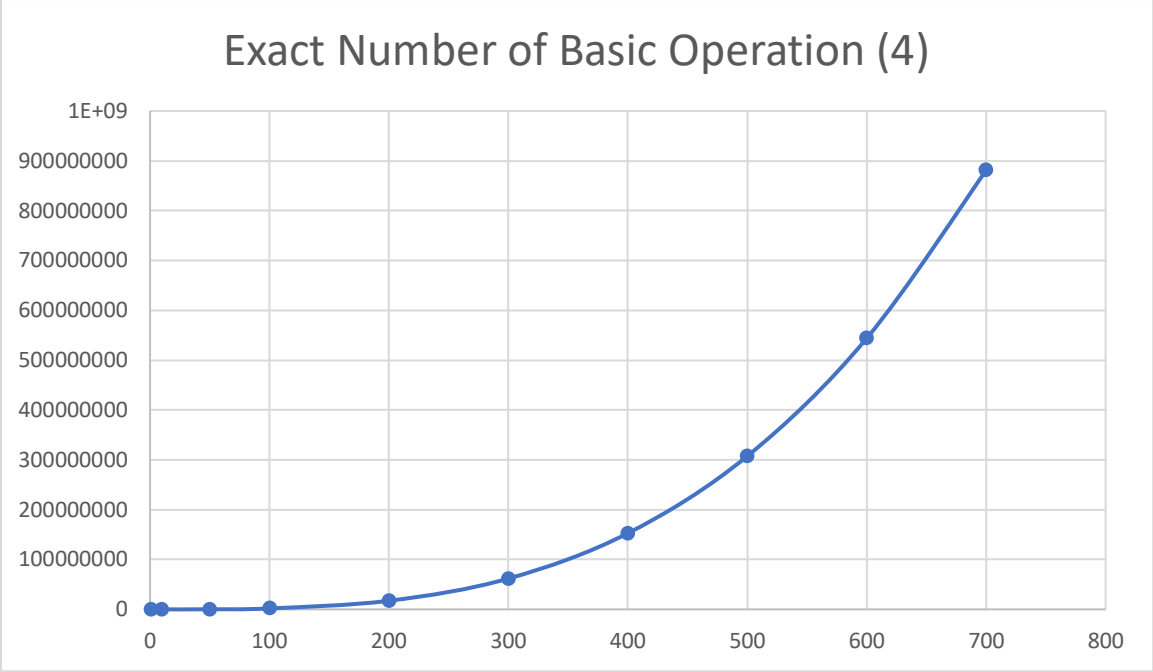


Graph of the theoretical analysis when basic operation is the operation marked as (3)



Leyla Yayladere – 2018400216
Umut Deniz Şener – 2018400225

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



Comments

- For operation (1), we think that it doesn't matter ratio of 0 or 1 in the array, since the comparison operation will be done for each element regardless of whether it is 0 or 1. Therefore, complexity is the same for best, worst, and average case.
- For operation (2), we think that it doesn't matter ratio of 0 or 1 in the array, since the operation (2) is in the both part of if-else statement. So, this operation will be done for each element regardless of whether it is 0 or 1. Therefore, complexity is the same for best, worst, and average case.
- In the average case, we should fill the list such that one-third of the elements are zero and two-thirds are one according to given information. So, we execute if part 1/3 times and else part 2/3 times. According to this, operation (3) will be executed 1/3 times in the worst-case complexity, because in the best-case scenario execution time is equal to 0. -> W(n)/3
- Operation (4) will be executed 1/3 times in the if part and 2/3 times in else part so the average complexity must be (1*B(n) + 2* W(n))/3.