Project 1

CmpE 300, Analysis of Algorithms, Fall 2022

Bahadır Gezer - 2020400039
Muhammet Batuhan Ilhan - 2019400243

November 2022

Contents

1 Theoretical Analysis

Require: $n = 2^k - 1$, $k \in \mathbb{Z}^+$
Require: $arr[i] \in \{0, 1, 2\}$, $0 \leq i \leq n - 1$
1: **Input:** $arr[0 : n - 1]$
2: **Output:** $arr2[0 : 4]$
3: **function** $Example(arr[0 : n - 1])$
4:
5: $arr2 \leftarrow [0, 0, 0, 0, 0]$
6: **for** $i \leftarrow 0$ **to** $n - 1$ **do**
7: **if** $arr[i] = 0$ **then**
8: **for** $t1 \leftarrow i$ **to** $n - 1$ **do**
9: $p1 \leftarrow t1^{1/2}$
10: $x1 \leftarrow n + 1$
11: **while** $x1 \geq 1$ **do**
12: $x1 \leftarrow \lfloor x1/2 \rfloor$
13: $arr2[i \bmod 5] \leftarrow arr2[i \bmod 5] + 1$
14: **end while**
15: **end for**
16: **else if** $arr[i] = 1$ **then**
17: **for** $t2 \leftarrow n$ **down to** 1 **do**
18: **for** $p2 \leftarrow 1$ **to** n **do**
19: $x2 \leftarrow n + 1$
20: **while** $x2 > 0$ **do**
21: $x2 \leftarrow \lfloor x2/2 \rfloor$
22: $arr2[i \bmod 5] \leftarrow arr2[i \bmod 5] + 1$
23: **end while**
24: **end for**
25: **end for**
26: **else if** $arr[i] = 2$ **then**
27: **for** $t3 \leftarrow 1$ **to** n **do**
28: $x3 \leftarrow t3 + 1$
29: **for** $p3 \leftarrow 0$ **to** $t3^2 - 1$ **do**
30: $arr2[i \bmod 5] \leftarrow arr2[i \bmod 5] + 1$
31: **end for**
32: **end for**
33: **end if**
34: **end for**
35: **end**

for loop 1 - line 6: This loop will iterate **from** 0 **to** $\mathbf{n - 1}$. Which is a total of **n** iterations.

for loop 2 - line 8: This loop will iterate **from i to** $\mathbf{n - 1}$. Which is a total of $\mathbf{n - i}$ iterations.

$$
\begin{aligned}
&i = 0 \quad \text{loop iterates n times} \\
&i = 1 \quad \text{loop iterates n - 1 times} \\
&i = 2 \quad \text{loop iterates n - 2 times} \\
&\vdots \\
&i = n - 1 \quad \text{loop iterates 1 time}
\end{aligned}
$$

Therefore, this loop iterates $\frac{\mathbf{n}\cdot(\mathbf{n}+\mathbf{1})}{\mathbf{2}}$ times. This calculation takes for loop 1 into account.

while loop 1 - line 11: This loop will iterate **from $\mathbf{n}+\mathbf{1}$ to 1** by halving at each iteration. For the sake of simplicity, lets consider $\mathbf{n}+\mathbf{1}$ as $\mathbf{2^z}$ where $\mathbf{z}$ is some positive integer. It will be generalized using interpolation later.

$$
\begin{aligned}
&\textit{Iteration 1 -} && k = 2^z \\
&\textit{Iteration 2 -} && k = 2^{z-1} \\
&\textit{Iteration 3 -} && k = 2^{z-2} \\
&\quad\vdots && \quad\vdots \\
&\textit{Iteration m -} && k = 2^{z-m-1} = 1
\end{aligned}
$$

As seen above, the loop will iterate $\mathbf{m}$ times. Solving the equation $\mathbf{2^{z-m-1}=1}$ for $\mathbf{m}$ we get $\mathbf{m=z-1}$. Substituting $\mathbf{z=\log_2(n+1)}$ into the equation we get $\mathbf{m=\log_2(n+1)-1}$. Thus, the loop will have $\mathbf{\log_2(n+1)-1}$ iterations. The last iteration does not count, because of the termination statement of the while loop, so the real number of iterations is $\mathbf{\log_2(n+1)-2}$.

for loop 3 - line 17: This loop will iterate **from n to 1**. The loop is not interrupted mid iteration. Thus the total number of iterations come out to be $\mathbf{n}$.

for loop 4 - line 18: This loop will iterate **from 1 to n**. The loop is not interrupted mid iteration. Thus the total number of iterations come out to be $\mathbf{n}$.

while loop 2 - line 20: This loop will iterate **from $\mathbf{n}+\mathbf{1}$ to 1** by halving at each iteration. For the sake of simplicity, lets consider $\mathbf{n}+\mathbf{1}$ as $\mathbf{2^z}$ where $\mathbf{z}$ is some positive integer.

$$
\begin{aligned}
&\textit{Iteration 1 -} && k = 2^z \\
&\textit{Iteration 2 -} && k = 2^{z-1} \\
&\textit{Iteration 3 -} && k = 2^{z-2} \\
&\quad\vdots && \quad\vdots \\
&\textit{Iteration m -} && k = 2^{z-m-1} = 1
\end{aligned}
$$

As seen above, the loop will iterate $\mathbf{m}$ times. Solving the equation $\mathbf{2^{z-m-1}=1}$ for $\mathbf{m}$ we get $\mathbf{m=z-1}$. Substituting $\mathbf{z=\log_2(n+1)}$ into the equation we get $\mathbf{m=\log_2(n+1)-1}$. Thus, the loop will have $\mathbf{\log_2(n+1)-1}$ iterations.

for loop 5 - line 27: This loop will iterate **from 1 to n**. The loop is not interrupted mid iteration. Thus the total number of iterations come out to be $\mathbf{n}$.

for loop 6 - line 29: This loop will iterate **from 0 to $\mathbf{t3^2-1}$**.

$$
\begin{aligned}
&t3 = 0 && \text{loop iterates } \mathbf{0^2} \text{ times} \\
&t3 = 1 && \text{loop iterates } \mathbf{1^2} \text{ times} \\
&t3 = 2 && \text{loop iterates } \mathbf{2^2} \text{ times} \\
&\quad\vdots \\
&t3 = n-1 && \text{loop iterates } \mathbf{(n-1)^2} \text{ time}
\end{aligned}
$$

Since there is no interruption in the loop, it will always iterate $(\mathbf{0^2+1^2+2^2+...+(n-1)^2})$ times. This sum is equal to $\frac{\mathbf{(n-1)\cdot n\cdot(2n-1)}}{\mathbf{6}}$. This value is the total number of times the statements inside this loop will iterate, it takes the outer loop into account.

1.1 Comparison as Basic Operation - (1)

Since for loop 1 runs without interruption - we do not have any break statements. So the the comparison will run regardless of the input. This comparison will be executed once for each iteration of the for loop. This loop will iterate **from 0 to n − 1**, which is **n** iterations.

1.1.1 Analyze *B(n)*

The complexity does not matter on the input. The loop iterates **n** times, and the comparison is executed once for each iteration of the loop. Thus, $\mathbf{B(n) = 1 \cdot n}$. We can prove that this complexity belongs to the $\mathbf{\Theta(n)}$ complexity class by using the definitions of $\boldsymbol{\mathcal{O}}$ and $\mathbf{\Omega}$

$$
\begin{aligned}
B(n) = n \leq n \\
&\implies B(n) \in \mathcal{O}(n) \\
B(n) = n \geq n \\
&\implies B(n) \in \Omega(n) \\
B(n) \in \Omega(n) \land B(n) \in \mathcal{O}(n) &\implies \mathbf{B(n) \in \Theta(n)}
\end{aligned}
$$

□

1.1.2 Analyze *W(n)*

The complexity does not matter on the input. The loop iterates **n** times, and the comparison is executed once for each iteration of the loop. Thus, $\mathbf{W(n) = 1 \cdot n}$. We can prove that this complexity belongs to the $\mathbf{\Theta(n)}$ complexity class by using the definitions of $\boldsymbol{\mathcal{O}}$ and $\mathbf{\Omega}$

$$
\begin{aligned}
W(n) = n \leq n \\
&\implies W(n) \in \mathcal{O}(n) \\
W(n) = n \geq n \\
&\implies W(n) \in \Omega(n) \\
W(n) \in \Omega(n) \land W(n) \in \mathcal{O}(n) &\implies \mathbf{W(n) \in \Theta(n)}
\end{aligned}
$$

□

1.1.3 Analyze *A(n)*

The complexity does not matter on the input. The loop iterates **n** times, and the comparison is executed once for each iteration of the loop. Thus, $\mathbf{A(n) = 1 \cdot n}$. We can prove that this complexity belongs to the $\mathbf{\Theta(n)}$ complexity class by using the definitions of $\boldsymbol{\mathcal{O}}$ and $\mathbf{\Omega}$

$$
\begin{aligned}
A(n) = n \leq n \\
&\implies A(n) \in \mathcal{O}(n) \\
A(n) = n \geq n \\
&\implies A(n) \in \Omega(n) \\
A(n) \in \Omega(n) \land A(n) \in \mathcal{O}(n) &\implies \mathbf{A(n) \in \Theta(n)}
\end{aligned}
$$

□

1.2 Three Assignments as Basic Operation - (2)

There are three possible entries in the input set, one for each if statement. The first if statement has one basic operation inside of it. This basic operation runs for loops 1 and 2, and also while loop 1. These are all executing inside of each other, so we will multiply their iteration counts. The basic operation is running once for all of these iterations. The second if statement -branch- also has one basic operation inside of it. This second basic operation runs for loops 1, 3 and 4, and also while loop 2. These are all executing inside of each other, so we will multiply their iteration counts. This second basic operation line runs once for all of these iterations. The third if statement -branch- also has one basic operation inside of it. This third basic operation runs for loop 1, 5 and 6. These are all executing inside of each other, so we will multiply their iteration counts. This third basic operation line runs once for all of these iterations. These executions come out to three entries in the input set, with each of them having a probability of $\frac{1}{3}$. Doing the operations above, we get the input set below.

I_1 - $\rho(I_1) = \frac{1}{3}$, $\tau(I_1) = \frac{1}{2} \cdot (n^2+1)(log_2(1+n)-2)$

I_2 - $\rho(I_2) = \frac{1}{3}$, $\tau(I_2) = n^3 \cdot (log_2(n+1)-1)$

I_3 - $\rho(I_3) = \frac{1}{3}$, $\tau(I_3) = \frac{n^2 \cdot (n-1) \cdot (2n-1)}{6}$

$T_n = \{I_1, I_2, I_3\}$

1.2.1 Analyze *B(n)*

$$B(n) = min\{\tau(I) \mid I \in T_n\}$$
$$= \frac{1}{2} \cdot (n^2-1) \cdot (log_2(n+1)-2)$$

Some positive constants $\mathbf{c_1}$, $\mathbf{c_2}$ *and* $\mathbf{n_0}$ *can be found such that*
$\mathbf{W(n)}$ *is squeezed by the function above when* $\mathbf{n > n_0}$.

$$\implies W(n) \in \Theta(\frac{1}{2} \cdot (n^2-1) \cdot (log_2(n+1)-2))$$

Constants and terms of lower order can be omitted from the complexity class.

$$\implies \mathbf{A(n) \in \Theta(n^2 \cdot log_2(n))}$$

1.2.2 Analyze *W(n)*

$$W(n) = max\{\tau(I) \mid I \in T_n\}$$
$$= \frac{n^2 \cdot (n-1) \cdot (2n-1)}{6}$$
$$= \frac{n^2}{6} - \frac{n^3}{2} + \frac{n^4}{3}$$

Some positive constants $\mathbf{c_1}$, $\mathbf{c_2}$ *and* $\mathbf{n_0}$ *can be found such that*
$\mathbf{W(n)}$ *is squeezed by the function above when* $\mathbf{n > n_0}$.

$$\implies W(n) \in \Theta(\frac{n^2}{6} - \frac{n^3}{2} + \frac{n^4}{3})$$

Constants and terms of lower order can be omitted from the complexity class.

$$\implies \mathbf{A(n) \in \Theta(n^4)}$$

1.2.3 Analyze *A(n)*

$$A(n) = \sum_{I \in T_n} \tau(I) \cdot \rho(I)$$

$$A(n) = \frac{1}{3} \cdot \frac{1}{2} \cdot (n^2 - 1)(\log_2(1+n) - 2) + \frac{1}{3} \cdot n^3(\log_2(n+1) - 1) + \frac{1}{3} \cdot \frac{1}{6} \cdot n^2 \cdot (n-1) \cdot (2n-1)$$

$$= \frac{n^4}{9} - \frac{n^3}{2} + \frac{n^3 \log_2(n+1))}{3} - \frac{5 \cdot n^2}{18} + \frac{n^2 \log_2(n+1)}{6} - \frac{\log_2(n+1)}{6} + \frac{1}{3}$$

Some positive constants $\mathbf{c_1}$, $\mathbf{c_2}$ *and* $\mathbf{n_0}$ *can be found such that*

$\mathbf{A(n)}$ *is squeezed by the function above when* $\mathbf{n > n_0}$.

$$\implies A(n) \in \Theta(\frac{n^4}{9} - \frac{n^3}{2} + \frac{n^3 \log_2(n+1))}{3} - \frac{5 \cdot n^2}{18} + \frac{n^2 \log_2(n+1)}{6} - \frac{\log_2(n+1)}{6} + \frac{1}{3})$$

Constants and terms of lower order can be omitted from the complexity class.

$$\implies \mathbf{A(n) \in \Theta(n^4)}$$

1.3 Two Assignments as Basic Operation - (3)

Only second and third if block contains basic operation. The basic operation inside the second if block iterates in while loop 2. Therefore, it has same iteration number with while loop 2 which is $\mathbf{n^3 \cdot (\log_2(n+1) - 1)}$. The basic operation inside the third if block iterates in for loop 6. Therefore, it has same iteration number with for loop 6 which is $\frac{\mathbf{n^2 \cdot (n-1) \cdot (2n-1)}}{\mathbf{6}}$.

1.3.1 Analyze *B(n)*

Best case is where the input array is filled with zeros. In that case the program does not enter the if statements -branches- with our basic operations. Thus, no basic operation is executed when in the best case. So, $\mathbf{B(n) = 0 \cdot n = 0}$, the complexity class of which can be found trivially. $\mathbf{B(n) = \Theta(1)}$.

1.3.2 Analyze *W(n)*

Worst case is where the input array is filled with twos. In that case the program only enters the third if branch. In this branch our basic operation is inside of for loops 1, 5 and 6. This same complexity was calculated in *Section 1.2.2*. So, taking the same steps as before we get $\mathbf{W(n)} = \frac{\mathbf{n^2}}{\mathbf{6}} - \frac{\mathbf{n^3}}{\mathbf{2}} + \frac{\mathbf{n^4}}{\mathbf{3}}$.

$$W(n) = \frac{n^2}{6} - \frac{n^3}{2} + \frac{n^4}{3}$$

Some positive constants $\mathbf{c_1}$, $\mathbf{c_2}$ *and* $\mathbf{n_0}$ *can be found such that*

$\mathbf{W(n)}$ *is squeezed by the function above when* $\mathbf{n > n_0}$.

$$\implies W(n) \in \Theta(\frac{n^2}{6} - \frac{n^3}{2} + \frac{n^4}{3})$$

Constants and terms of lower order can be omitted from the complexity class.

$$\implies \mathbf{W(n) \in \Theta(n^4)}$$

1.3.3 Analyze *A(n)*

Probabilities of $\mathbf{arr[i] = 0}, \mathbf{arr[i] = 1}$ and $\mathbf{arr[i] = 2}$ are $\frac{\mathbf{1}}{\mathbf{3}}$. Therefore, each if blocks have same execution probability. This makes the set $\mathbf{T_n}$ have 3 elements. Which each mapping out to one of the three conditional branches the function might enter.

I_1 - $\rho(I_1) = \frac{1}{3}$, $\tau(I_1) = 0$

I_2 - $\rho(I_2) = \frac{1}{3}$, $\tau(I_2) = n^3 \cdot (log_2(n+1) - 1)$

I_3 - $\rho(I_3) = \frac{1}{3}$, $\tau(I_3) = \frac{n^2 \cdot (n-1) \cdot (2n-1)}{6}$

$$
\begin{aligned}
A(n) &= \sum_{I \in T_n} \tau(I) \cdot \rho(I) \\
&= (\frac{1}{3} \cdot 0 + \frac{1}{3} \cdot (n^3 \cdot (log_2(n+1) - 1)) + \frac{1}{3} \cdot \frac{n^2 \cdot (n-1) \cdot (2n-1)}{6} \\
&= \frac{n^4}{9} - \frac{n^3}{2} + \frac{n^2}{18} + \frac{n^3 log_2(n+1)}{3}
\end{aligned}
$$

Some positive constants $\mathbf{c_1}$, $\mathbf{c_2}$ *and* $\mathbf{n_0}$ *can be found such that* $\mathbf{A(n)}$ *is squeezed by the function above when* $\mathbf{n > n_0}$.

$\implies A(n) \in \Theta(\frac{n^4}{9} - \frac{n^3}{2} + \frac{n^2}{18} + \frac{n^3 log_2(n+1)}{3})$

Terms of lower order -$\mathbf{n^3}, \mathbf{n^2}, \mathbf{n^3 log_2(n)}$- *can be omitted from the complexity class.*

$\implies \mathbf{A(n) \in \Theta(n^4)}$

1.4 Two Loop Increments as Basic Operation - (4)

Only second and third if block contains basic operation. The first basic operation line iterates inside of for loops 1, 3, and 4. These iterations are multiplied together since the loops are all inside of each other. This comes out to a total of $\mathbf{n^3}$ iterations. Our first basic operation is executed once for each iteration. So the first basic operation -and the second if block- has a complexity of $\mathbf{n^3}$. Our second basic operation line iterates inside of loops 1, and 5. These iterations are multiplied together since the two loops are inside of each other. This comes out to a total of $\mathbf{n^2}$ iterations. Our second basic operation is executed once for each iteration. So the second basic operation - and the third if block- has a complexity of $\mathbf{n^3}$. The first if block does not contain any basic operations so it has a basic operation count of $\mathbf{0}$.

1.4.1 Analyze *B(n)*

Best case is where the input array is filled with zeros. In that case the program does not enter the if statements -branches- with our basic operations. Thus, no basic operation is executed when in the best case. So, $\mathbf{B(n) = 0 \cdot n = 0}$, the complexity class of which can be found trivially. $\mathbf{B(n) = \Theta(1)}$.

1.4.2 Analyze *W(n)*

Worst case is where the input array is filled with ones. In that case the program only enters the second if branch. In this branch our basic operation is inside of for loops 1, 3 and 4. The basic operation count was calculated as $\mathbf{W(n) = n^3}$ above.

$$
\begin{aligned}
W(n) = n^3 \leq n^3 \\
\implies W(n) \in \mathcal{O}(n^3) \\
W(n) = n^3 \geq n^3 \\
\implies W(n^3) \in \Omega(n^3) \\
W(n) \in \Omega(n^3) \wedge W(n) \in \mathcal{O}(n^3) \implies \mathbf{W(n) \in \Theta(n^3)}
\end{aligned}
$$

1.4.3 Analyze *A(n)*

Probabilities of $\mathbf{i}$ in the input array $\mathbf{arr[i] = 0}, \mathbf{arr[i] = 1}$ and $\mathbf{arr[i] = 2}$ are $\frac{\mathbf{1}}{\mathbf{3}}$. Therefore, each if blocks have same execution probability. This makes the set $\mathbf{T_n}$ have 3 elements. Which each mapping out to one of the three conditional branches the function might enter.

I_1 - $\rho(I_1) = \frac{1}{3}$, $\tau(I_1) = 0$

I_2 - $\rho(I_2) = \frac{1}{3}$, $\tau(I_2) = n^3$

I_3 - $\rho(I_3) = \frac{1}{3}$, $\tau(I_3) = n^2$

$$
\begin{aligned}
A(n) &= \sum_{I \in T_n} \tau(I) \cdot \rho(I) \\
&= \frac{1}{3} \cdot 0 + \frac{1}{3} \cdot n^3 + \frac{1}{3} \cdot n^2 \\
&= \frac{n^3 + n^2}{3}
\end{aligned}
$$

Some positive constants $\mathbf{c_1}$, $\mathbf{c_2}$ *and* $\mathbf{n_0}$ *can be found such that* $\mathbf{A(n)}$ *is squeezed by the function above when* $\mathbf{n > n_0}$.

$\implies A(n) \in \Theta(\frac{n^3}{3} + \frac{n^2}{3})$

Constants and terms of lower order can be omitted from the complexity class.

$\implies \mathbf{A(n) \in \Theta(n^3)}$

1.5 Assignment as Basic Operation - (5)

Only the first if block contains a basic operation. So the basic operation count for the second and third if block is zero. The basic operation statement in the first if block is iterated inside of for loops 1, and 2. The second for loop iterates without interruption inside of the first one, so the total number of iterations inside of for loop 2 is $\frac{\mathbf{n \cdot (n+1)}}{\mathbf{2}}$. The basic operation is executed once for each iteration so the total number of basic operations for the first if block is $\frac{\mathbf{n \cdot (n+1)}}{\mathbf{2}}$.

1.5.1 Analyze *B(n)*

Best case is where the input array is either filled with ones or twos. In this case the program does not enter the if statement -branches- with our basic operation, which is the first if block. Thus, no basic operation is executed in the best case. So, $\mathbf{B(n) = 0 \cdot n = 0}$, the complexity class of which can be found trivially. $\mathbf{B(n) = \Theta(1)}$.

1.5.2 Analyze *W(n)*

Worst case is where the input array is filled with zeros. In that case the program only enters the first if branch. In this branch our basic operation is inside of for loops 1, and 2. The basic operation count was calculated as $\frac{\mathbf{n \cdot (n+1)}}{\mathbf{2}}$ above.

$$W(n) = \frac{n \cdot (n+1)}{2}$$
$$= \frac{n^2}{2} + \frac{n}{2}$$

Some positive constants $\mathbf{c_1}$, $\mathbf{c_2}$ *and* $\mathbf{n_0}$ *can be found such that*
$\mathbf{W(n)}$ *is squeezed by the function above when* $\mathbf{n > n_0}$.

$\implies W(n) \in \Theta(\frac{n^2}{2} + \frac{n}{2})$

Constants and terms of lower order can be omitted from the complexity class.

$\implies \mathbf{W(n) \in \Theta(n^2)}$

1.5.3 Analyze *A(n)*

Probabilities of $\mathbf{arr[i] = 0}, \mathbf{arr[i] = 1}$ and $\mathbf{arr[i] = 2}$ are $\frac{\mathbf{1}}{\mathbf{3}}$. Therefore, each if blocks have same execution probability. This makes the set $\mathbf{T_n}$ have 3 elements. Which each mapping out to one of the three conditional branches the function might enter.

I_1 - $\rho(I_1) = \frac{1}{3}$, $\tau(I_1) = \frac{n(n-1)}{2}$

I_2 - $\rho(I_2) = \frac{1}{3}$, $\tau(I_2) = 0$

I_3 - $\rho(I_3) = \frac{1}{3}$, $\tau(I_3) = 0$

$$A(n) = \sum_{I \in T_n} \tau(I) \cdot \rho(I)$$
$$= \frac{1}{3} \cdot \frac{n(n-1)}{2} + \frac{1}{3} \cdot 0 + \frac{1}{3} \cdot 0$$
$$= \frac{n^2}{6} + \frac{n}{6}$$

Some positive constants $\mathbf{c_1}$, $\mathbf{c_2}$ *and* $\mathbf{n_0}$ *can be found such that*
$\mathbf{A(n)}$ *is squeezed by the function above when* $\mathbf{n > n_0}$.

$\implies A(n) \in \Theta(\frac{n^2}{6} + \frac{n}{6})$

Constants and terms of lower order can be omitted from the complexity class.

$\implies \mathbf{A(n) \in \Theta(n^2)}$

2 Identification of Basic Operation(s)

The basic operation for this function are the assignment operations to $arr2$. Analysis of this basic operation is already done in *Section 1.2*.

3 Real Execution

The real basic operations for this program is identified in *Section 2*. Looking at the theoretical analysis -and the results from the real analysis support this- we can say that the best case is where the input array is filled with zeros, the worst case is where the array is filled with twos. The average case has theoretical complexity that is the same as the worst case, but in real analysis we will see lower execution times compared to the worst case.

3.1 Best Case

N Size	Time Elapsed (ms)
1	0.004768372
5	0.010967255
10	0.031948090
25	0.216007233
50	1.003980637
75	2.583026886
100	4.234075546
150	10.351896286
200	17.199993134
250	26.273965836

3.2 Worst Case

N Size	Time Elapsed (ms)
1	0.002861023
5	0.026941299
10	0.254154205
25	8.384943008
50	133.820772171
75	671.449661255
100	2112.159013748
150	10967.132091522
200	33765.886068344
250	83805.630207062

3.3 Average Case

N Size	Time Elapsed (ms)
1	0.006437302
5	0.029881795
10	0.379641851
25	7.709662120
50	66.747744878
75	334.980964661
100	1006.102959315
150	4331.564426422
200	13257.391373316
250	31049.947659175

4 Comparison

4.1 Best Case

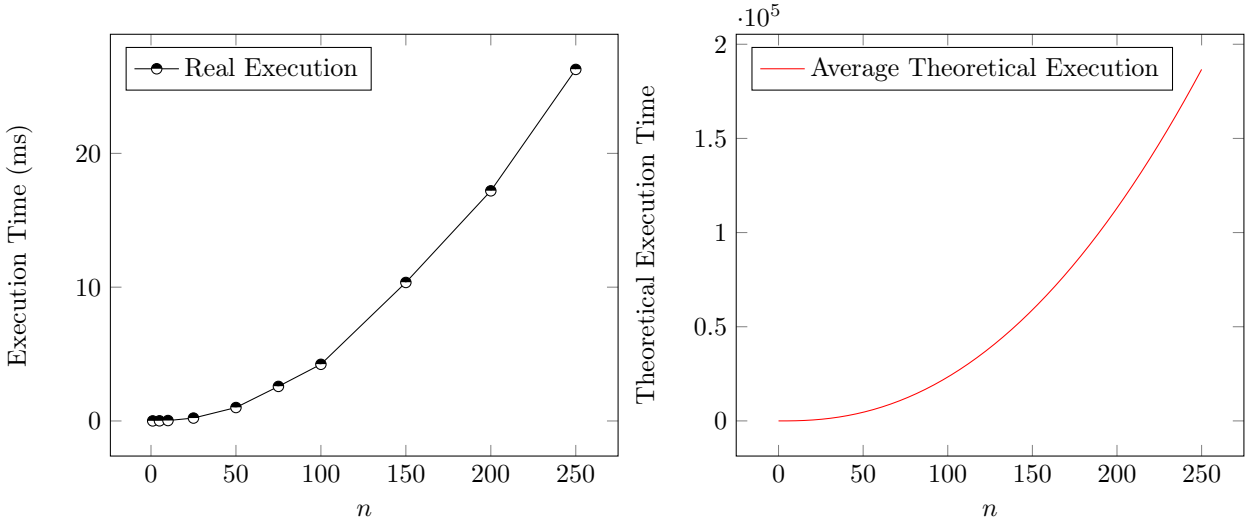

Figure 1: Side-by-side comparison of the real execution times and the correct theoretical complexity for the best case.

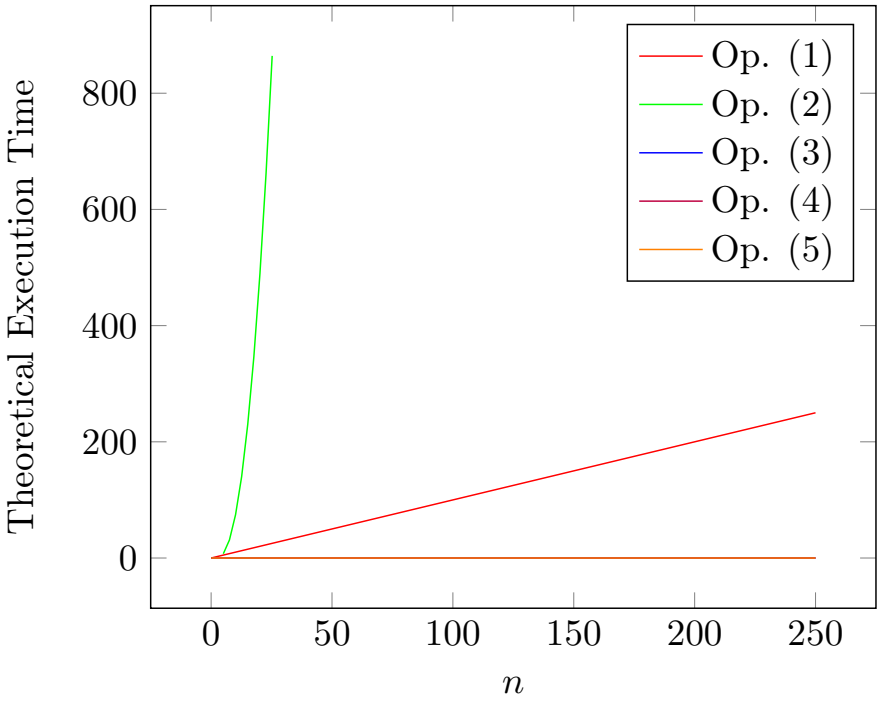

Figure 2: Graph of theoretical best complexities for different basic operations.

4.2 Worst Case

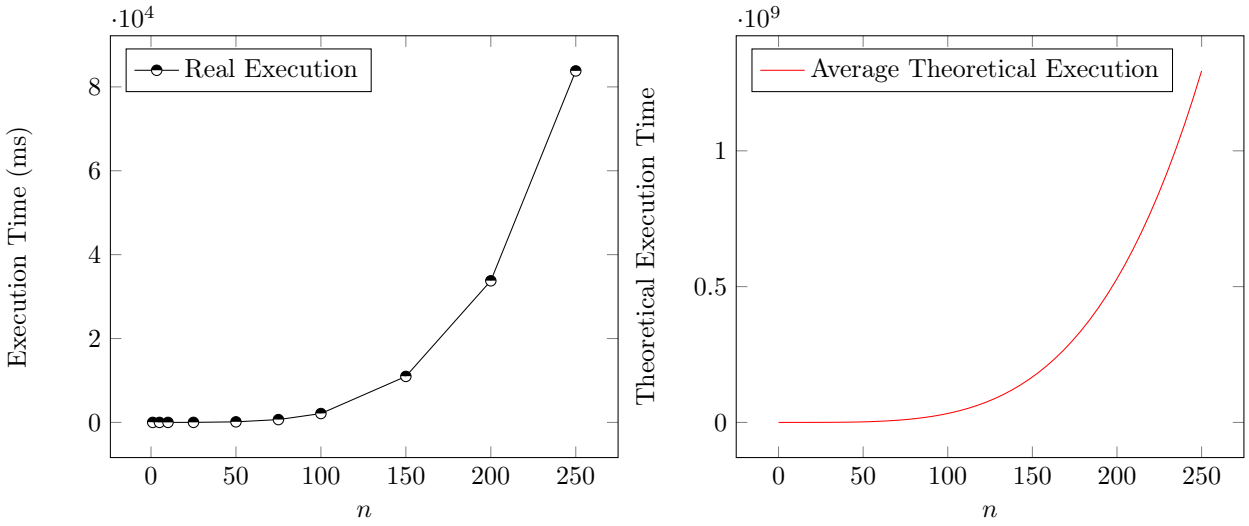

Figure 3: Side-by-side comparison of the real execution times and the correct theoretical complexity for the worst case.

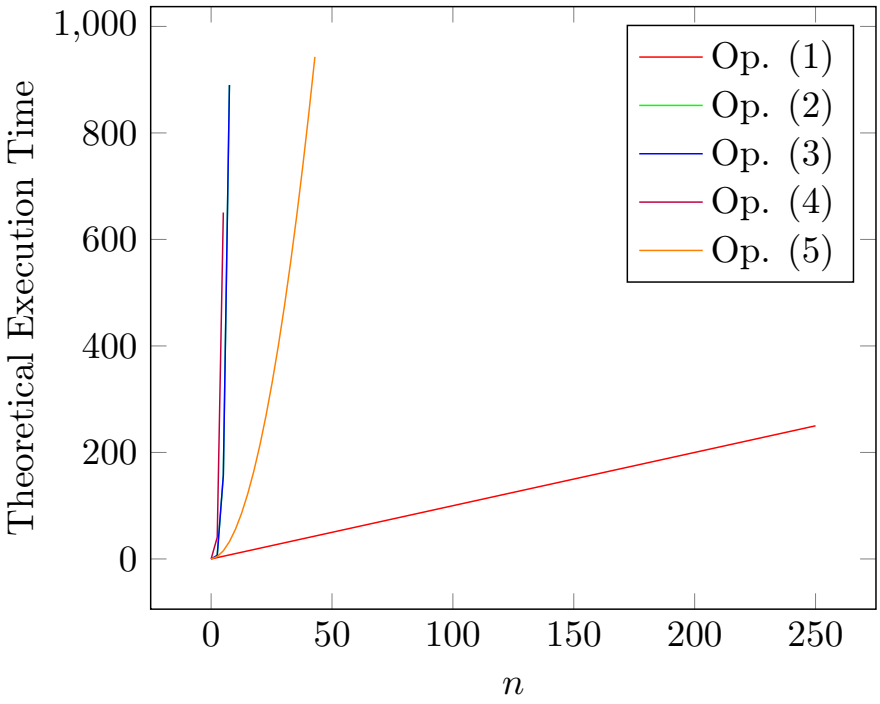

Figure 4: Graph of theoretical worst complexities for different basic operations.

4.3 Average Case

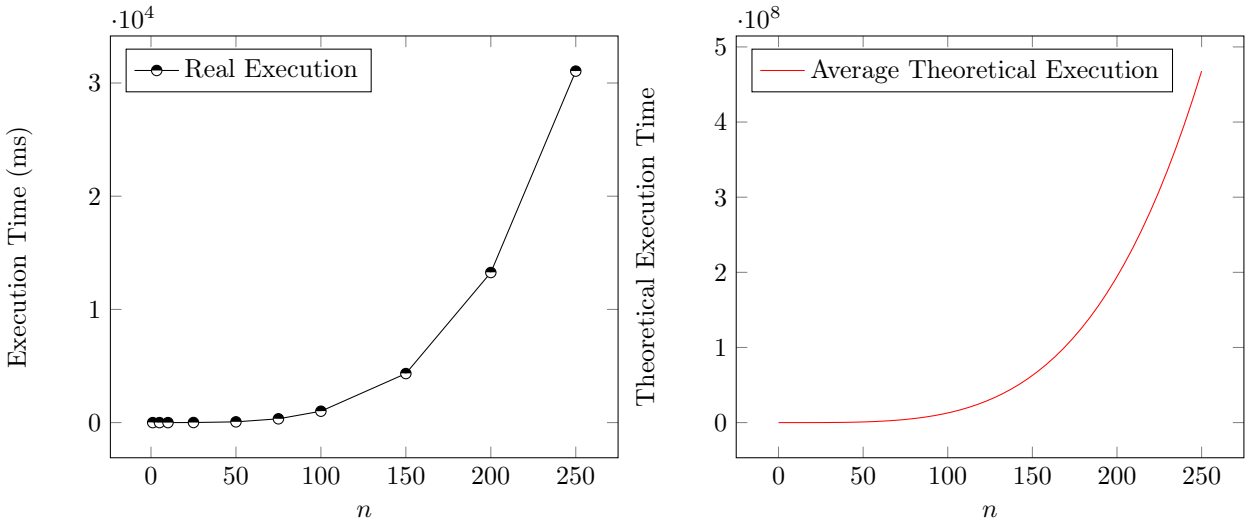

Figure 5: Side-by-side comparison of the real execution times and the correct theoretical complexity for the average case.

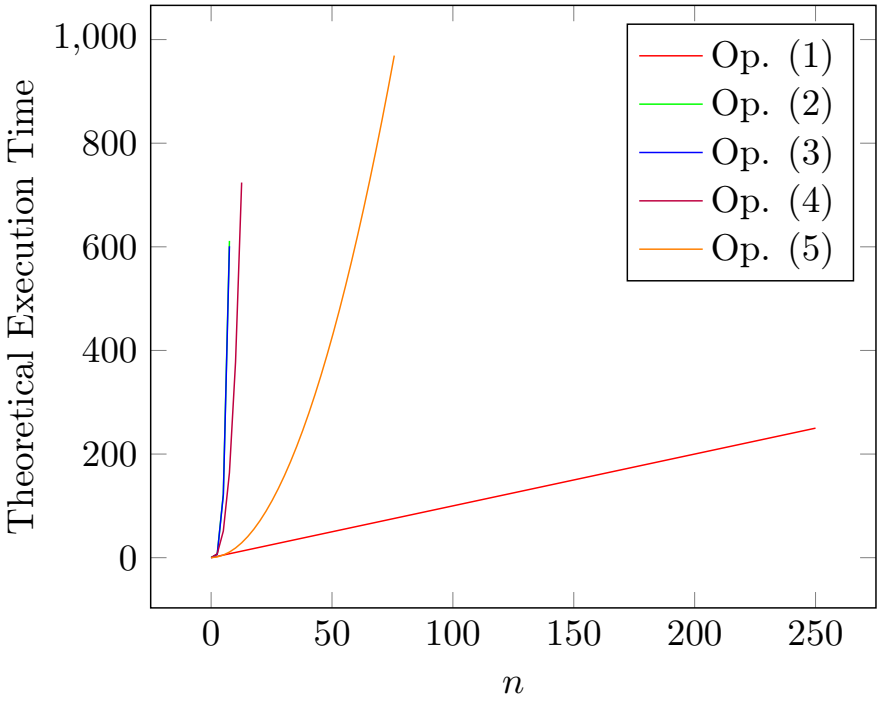

Figure 6: Graph of theoretical average complexities for different basic operations.

4.4 Comments on Graphs

All real execution time graphs and correct theoretical complexity graphs match up. The only difference is the constant scaling between the two. The theoretical graph is about 10^4 to 10^5 greater in the y-axis than the real execution time graphs. Which is normal since they have different units. The scale being the same between the three cases is also a indication that our analysis was correct.

The graphs of different basic operations use the complexity functions calculated in the related subsections. The actual basic operation count is used, not the complexity class. This created more variance in out graphs.

Best Case

The best case real execution time graph rises in the same way as the correct theoretical complexity. Which indicates that our analysis was correct. The graphs for the different basic operations are not too interesting for the best case. We have one linear graph and a quadratic one, the other three graphs are just constant zero.

Worst Case

The worst case real execution time graph rises in the same way as the correct theoretical complexity. Which indicates that our analysis was correct. Worst case graphs for different basic operations are more varied with one linear, one quadratic and three quartic graphs. This graph shows the actual complexity class difference between different polynomials. The different growth rates can be seen clearly.

Average Case The average case real execution time graph rises in the same way as the correct theoretical complexity. Which indicates that our analysis was correct. Average case graphs for different basic operations is the most varied of all. There is one linear, one quadratic, one cubic and two quartic graphs. The average case graphs, especially the quartic ones, have more terms in them and result in a set of graphs which are more varied than the other cases.