Big-O

Zakir Hossain, Summit Pradhan

September 29, 2021

1 Big-O Proofs

Use the formal definition of Big-O notation to prove the following statements:
1. $8n^3 + 7n^2 + 12$ is $\mathcal{O}(n^3)$
2. $6n^2 - n + 4$ is $\mathcal{O}(n^2)$

Definition of Big-O :

$f(n)$ is $\mathcal{O}(g(n))$ if:

there exists a c > 0 and $n_0 \geq$ such that:

$f(n) \leq c * g(n)$ for all n $\geq n_0$

1. $8n^3 + 7n^2 + 12$ is $\mathcal{O}(n^3)$

By substitution:

$$8n^3 + 7n^2 + 12 \leq 8n^3 + 7n^3 + 12n^3$$

Simplifying the terms on the right-hand side:

$$8n^3 + 7n^2 + 12 \leq 27n^3$$

Thus, as long as $c \geq 27$ and $n_0 \geq 1$ then we can say that$f(n)$ is $\mathcal{O}(n^3)$

2. $6n^2 - n + 4$ is $\mathcal{O}(n^2)$

By substitution:

$$6n^2 - n + 4 \leq 6n^2 - n^2 + 4n^2$$

Simplifying the terms on the right-hand side:

$$6n^2 - n + 4 \leq 9n^2$$

Thus, as long as $c \geq 9$ and $n_0 \geq 1$ then we can say that$f(n)$ is $\mathcal{O}(n^3)$

2 Big-O Analysis of Mystery Functions Pseudo-codes

1. **fnA**

```
function fnA(n)
    for $i$ in 1 to $N/2$ do
        set a To i
    end for
end function
```

time complexity: $\mathcal{O}(n)$

Since the only loop in fnA(n) runs from 1 to n/2, there will be n/2 iterations. Therefore, the time complexity of the function will be O(n) time.

2. **fnB**

```
function fnB(n)
    for $i$ in 1 to $N$ do
        for $j$ in 1 to $N$ do
            set a To i
        end for
    end for
end function
```

time complexity: $\mathcal{O}(n^2)$

The outer loop has n iterations, and the inner loop has n iterations. This means that the function has n*n iterations total. Therefore, the time complexity of the function will be O(n^2)*time*.

3. **fnC**

```
function fnC(n)
    for $i$ in 1 to $N$ do
        for $j$ in 1 to 4 do
            set a To i
        end for
    end for
end function
```

time complexity: $\mathcal{O}(n)$

The outer loop has n iterations, and the inner loop has a constant 4 iterations. This means that the function has 4n iterations total. Therefore, the time complexity of the function will be O(n) time.

4. **fnD**

```
function fnD(n)
    set i To 0
    while $i < N * N * N$ do
        set a To i + 1
    end while
end function
```

time complexity: $\mathcal{O}(n^3)$

The only loop in the function iterates from 0 to n^3, therefore the function will have n^3 iterations. Therefore, the time complexity of the function will be O(n^3)*time*

5. **fnE**

```
function fnE(n)
    for $i$ in 1 to $N$ do
        Set j To 1
        while $j < N$ do
            Set j To j*2
        end while
    end for
end function
```

time complexity: $\mathcal{O}(nlog(n))$

The outer loop of the function iterates n times. Since the inner loop doubles values on every iteration while less than n ($2^j < n$), then the number of iterations, j, is equal to log2(n). The number of iterations is nlog2(n), therefore the time complexity of the function is O(nlog(n)) time.

6. **fnF**

```
function fnF(n)
    for $i$ in 1 to $N * N$ do
        for $j$ in 1 to $N * N$ do
            Set a To j
        end for
    end for
end function
```

time complexity: $\mathcal{O}(n^4)$

The outer loop iterates n^2 times, and the inner loop iterates n^2 times. This means that the total number of iterations in the function is n^4 iterations ($n^2 * n^2 = n^4$). Therefore, the time complexity of the function is O(n^4)*time*.

Time complexity from shortest to longest:
fnA(n), *fnC*(n), *fnE*(n), *fnB*(n), *fnD*(n), *fnF*(n)

3 Matching Mystery Functions with Pseudo-codes

1. **Matching**

   a. **f1 is fnF(n)**
   b. **f2 is fnB(n)**
   c. **f3 is fnD(n)**
   d. **f4 is fnE(n)**
   e. **f5 is fnA(n)**
   f. **f6 is fnC(n)**

2. **Order of Graphs from longest to shortest run times:**

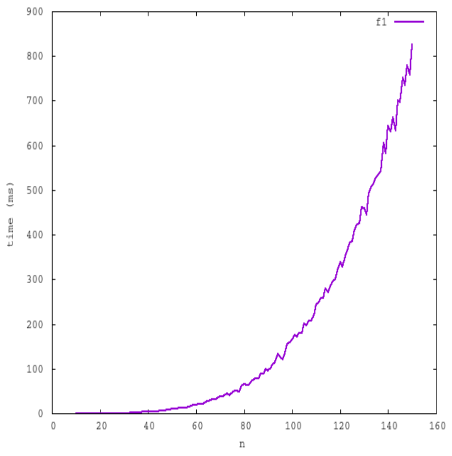
   $f1 : (O(n^4)),\ f3 : (O(n^3)),\ f2 : (O(n^2)),\ f4 : (O(nlog(n))),\ f6 : (O(n)),\ f5 : (O(n))$

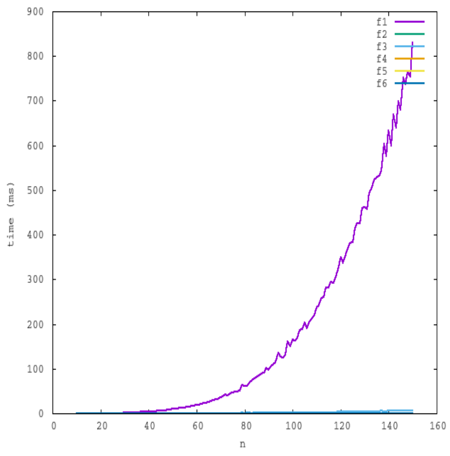
3. **General Argument:**

   Since we know the relative order of the Big-O time complexity of the mystery functions, and we can figure out the relative order of run times for each of the graphing functions, we can figure out which mystery function relates to which graph function. As an additional support, the shape of the graphs should match with the Big-O time complexities of the mystery functions (linear vs nonlinear).

4. **The Graphs (longest to shortest run times):**

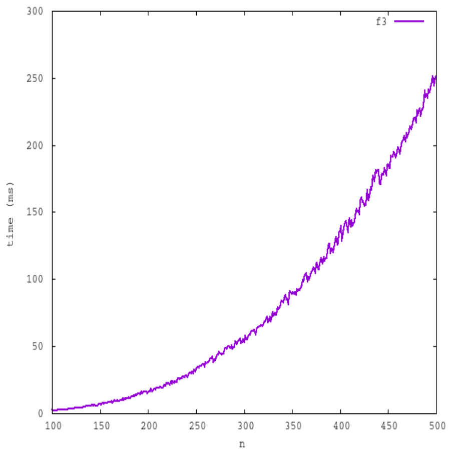
   a. **Function 1: n = 10 m = 150**

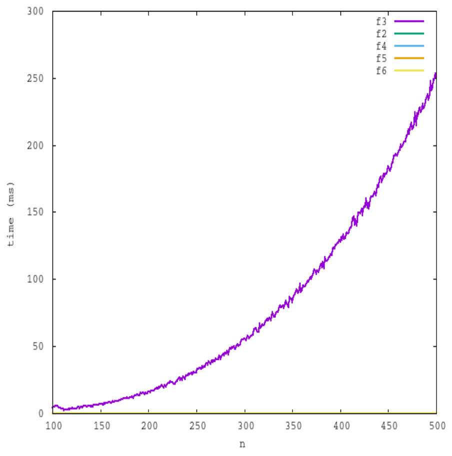




The graph of f1 from n = 10 to m = 150 looks to be in a nonlinear shape. When compared to all of the other graphs at this range, it is shown to have a significantly higher runtime. Since the The The mystery function fnF(n) has the longest run time and is $O(n^4)$, which is nonlinear, it is likely that f1 is fnF(n)

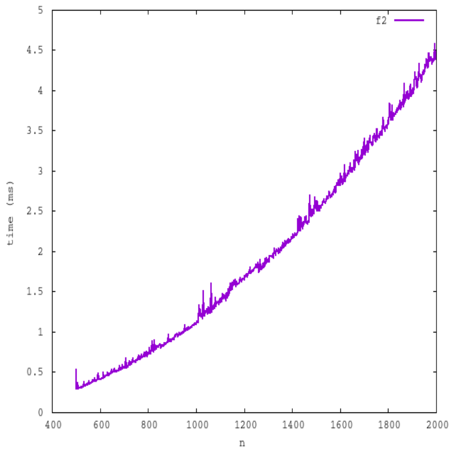
b. **Function 3: n = 100 m = 500**

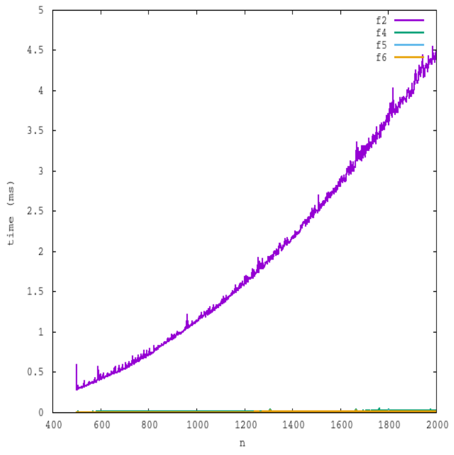




The graph of f3 from n = 100 to m = 500 looks to be in a nonlinear shape. When compared to f2, f4, ff5, and f6 at this range, it is shown to have a significantly higher runtime. Since the The The mystery function fnD(n) has the second longest run time and is $O(n^3)$, which is nonlinear, it is likely that f3 is fnD(n)

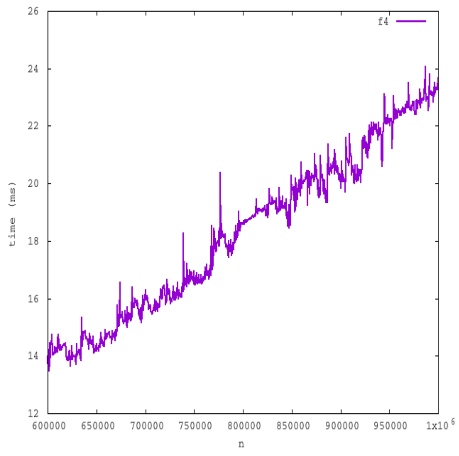
c. **Function 2: n = 400 m = 2000**

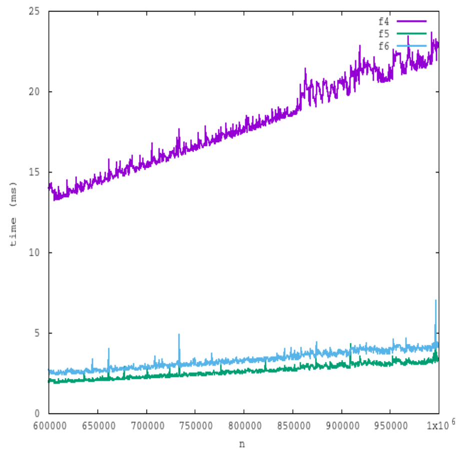




The graph of f2 from n = 400 to m = 2000 looks to be in a nonlinear shape. When compared to f4, f5, and f6 at this range, it is shown to have a significantly higher runtime. Since the The The mystery function fnB(n) has the third longest run time and is $O(n^3)$, which is nonlinear, it is likely that f2 is fnB(n)

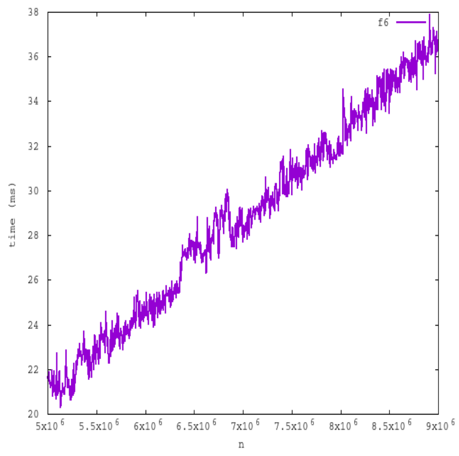
d. **Function 4: n = 600000 m = 1000000**

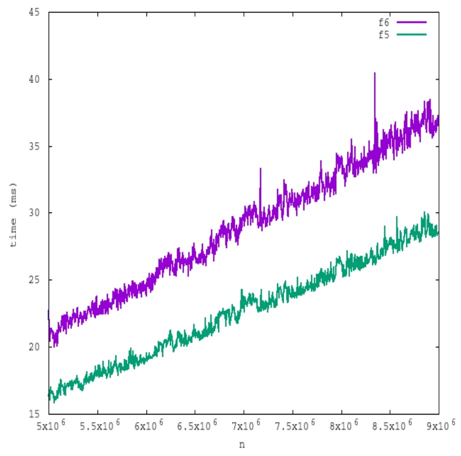




The graph of f4 from n = 600000 to m = 1000000 looks to be in a somewhat linear shape. When compared to f5 and f6 at this range, it is shown to have a significantly higher runtime. The mystery function fnE(n) also has the fourth longest run time. Although fnE(n) is O(nlog(n)) time, which is nonlinear, it would make sense that for very large values of n, a function of nlog(n) would begin to look linear as the rate of change for log(n) decreases significantly as n increases. Therefore, f4 is likely fnE(n)

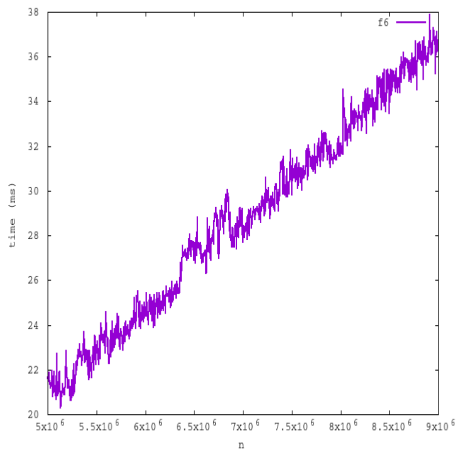
e. **Function 6: n = 5000000 m = 9000000**





The graph of f6 from n = 5000000 to m = 9000000 looks to be in a linear shape. When compared to f5 at this range, it is shown to have a significantly higher runtime. Since the mystery function fnC(n) has the fifth longest run time and is O(n), it is likely that f6 is fnC(n)

f. **Function 5: n = 5000000 m = 9000000**



The graph of f5 from n = 5000000 to m = 9000000 looks to be in a linear shape. Since all of the previous functions were shown to run longer than f5, f5 must have the shortest run time. The mystery function fnA(n) has the lowest time complexity of the mystery functions and is O(n). Therefore, f5 is likely fnA(n)