香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

INTRODUCTION TO COMPUTER SCIENCE: PROGRAMMING METHODOLOGY

TUTORIAL 10

ALGORITHM ANALYSIS

Frederick Khasanto
122040014

16 November 2023

Primitive operations

Counting primitive operations:

~ Assigning an identifier to an object
~ Performing an arithmetic operation
~ Comparing two numbers
~ Calling a function
…

Measuring operations as a function of input size.

To capture the order of growth of an algorithm's running time, we will associate, with each algorithm, a function f(n) that characterize the number of primitive operations as a function of input size.

An quadratic-time algorithm

Example:

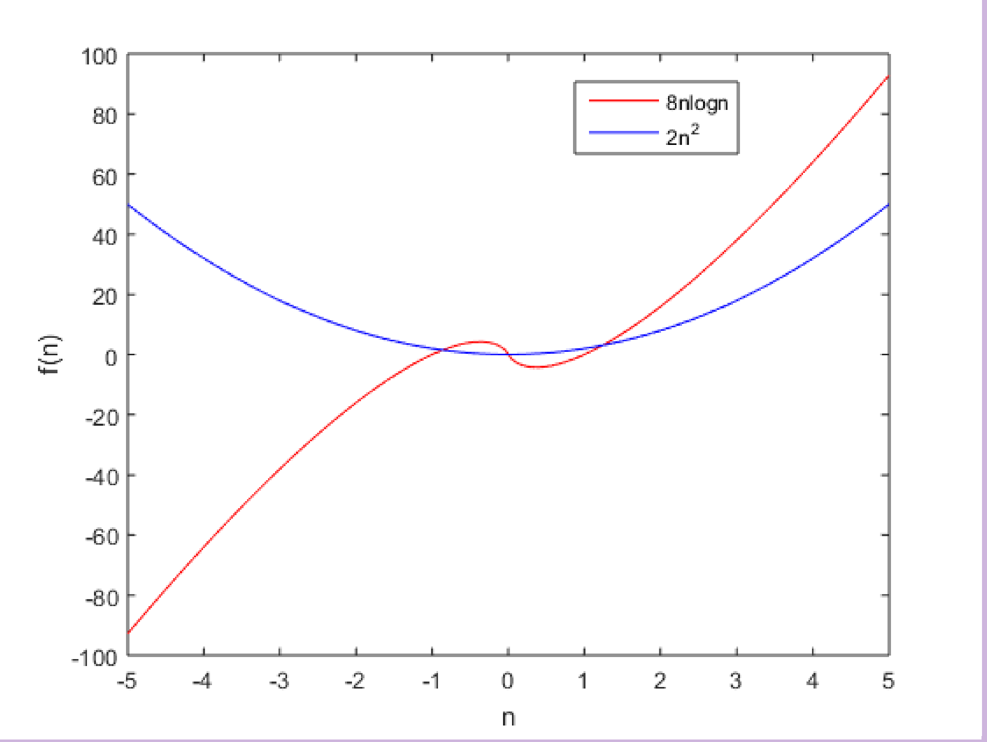
```
def prefix_average1(S):
  """Return list such that, for all j, A[j] equals average of S[0], ..., S[j]."""
  n = len(S)                         O(1)
  A = [0] * n                        O(n)      # create new list of n zeros
  for j in range(n):
    total = 0                        O(n)      # begin computing S[0] + ... + S[j]
    for i in range(j + 1):
      total += S[i]                  n(n+1)/2->O(n^2)
    A[j] = total / (j+1)             O(n)      # record the average
  return A
```

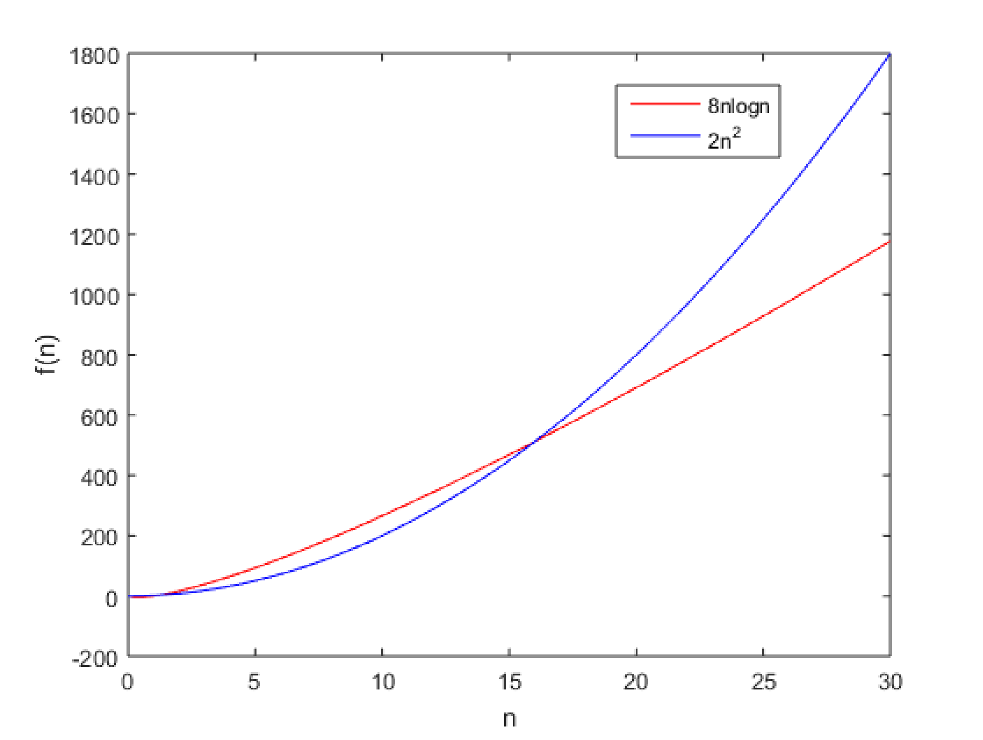
The running time of this program is $O(n^2)$: quadratic-time algorithm

Q1: Verify 8nlogn better than $2n^2$

The number of operations executed by algorithms A and B is $8n\log n$ and $2n^2$, respectively. Determine n_0 such that A is better than B for $n \geq n_0$.

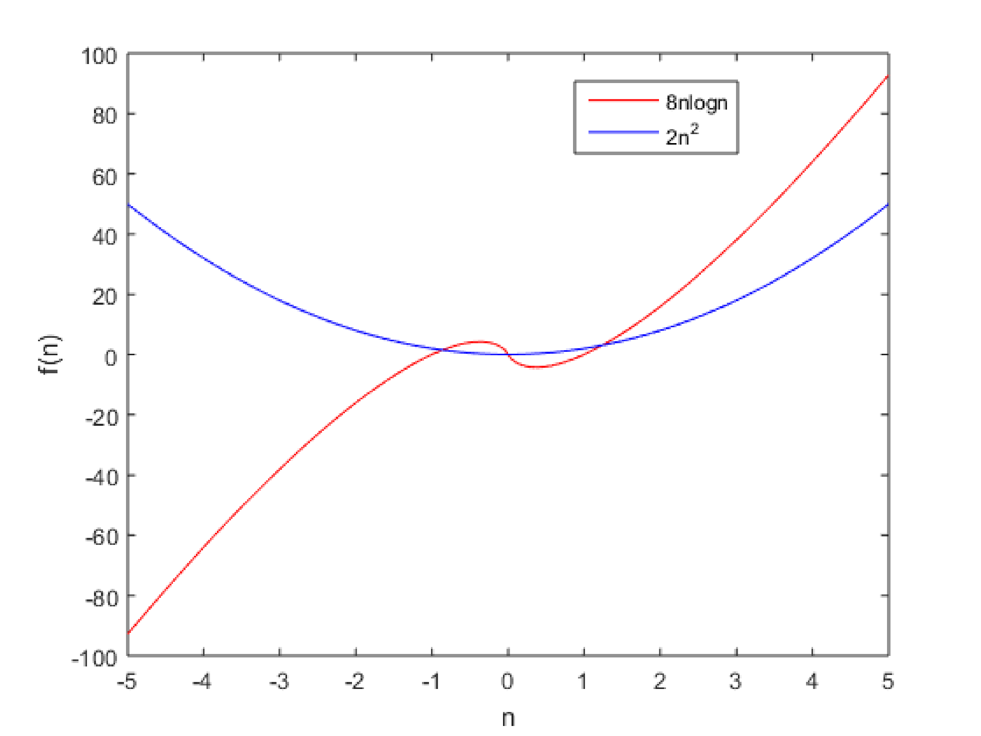
In this tutorial, logn means $\log_2 n$.

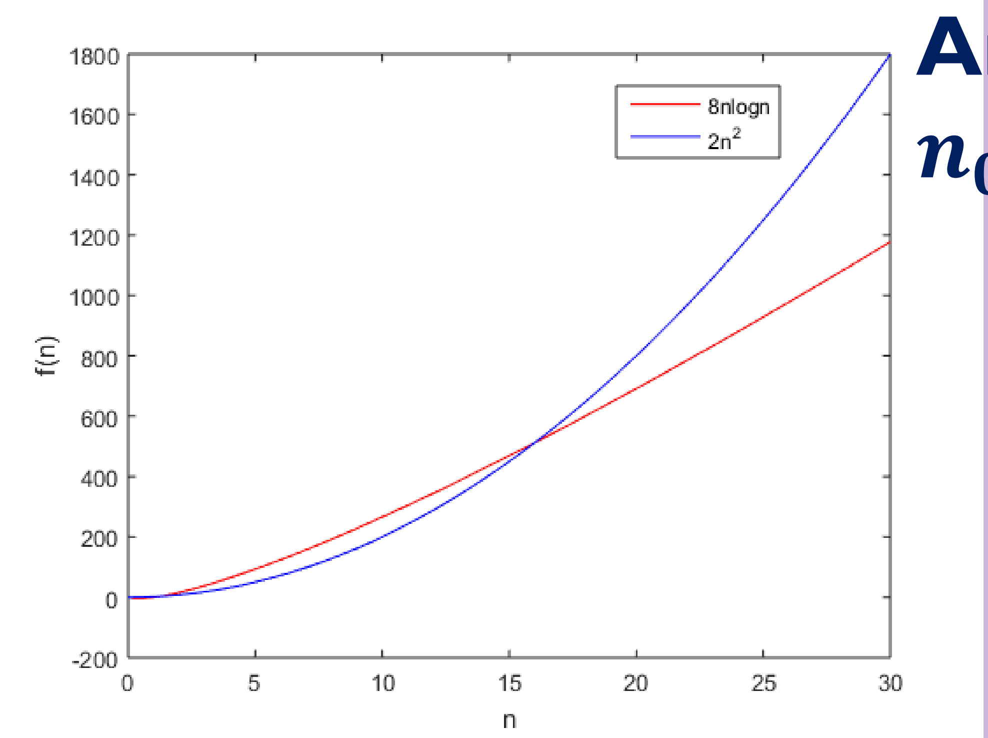




Q1: Answer

The number of operations executed by algorithms A and B is $8n \log n$ and $2n^2$, respectively. Determine n_0 such that A is better than B for $n \geq n_0$.

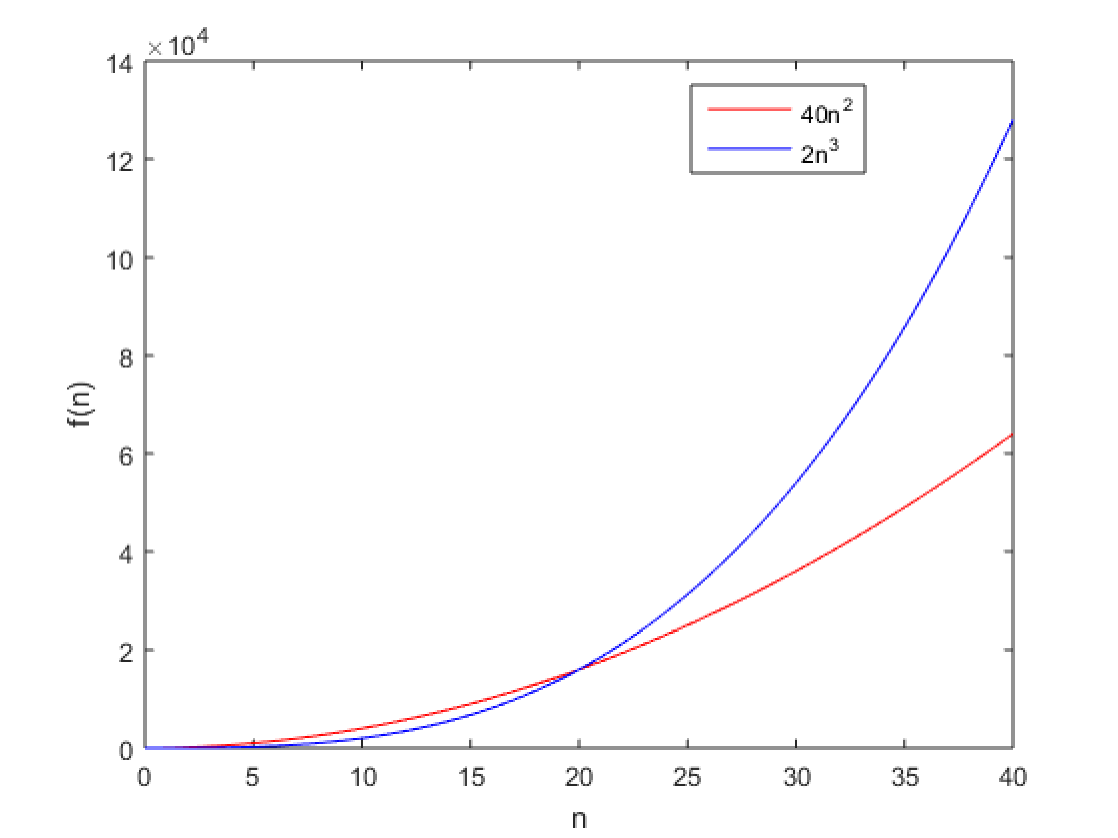




Answer: $\boldsymbol{n_0 = 16}$.

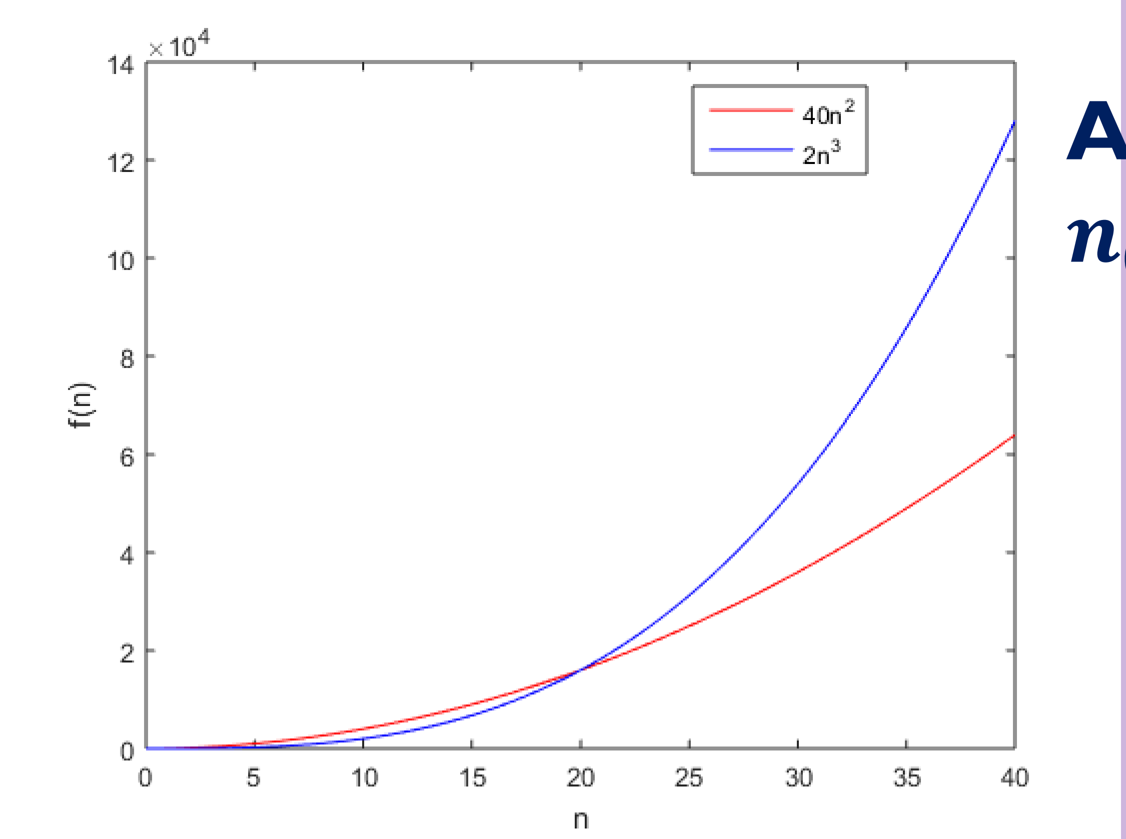
Q2: Verify $40n^2$ better than $2n^3$

The number of operations executed by algorithms A and B is $40n^2$ and $2n^3$, respectively. Determine n_0 such that A is better than B for $n \geq n_0$.



Q2:Answer

The number of operations executed by algorithms A and B is $40n^2$ and $2n^3$, respectively. Determine n_0 such that A is better than B for $n \geq n_0$.



Answer:

$n_0 = 20.$

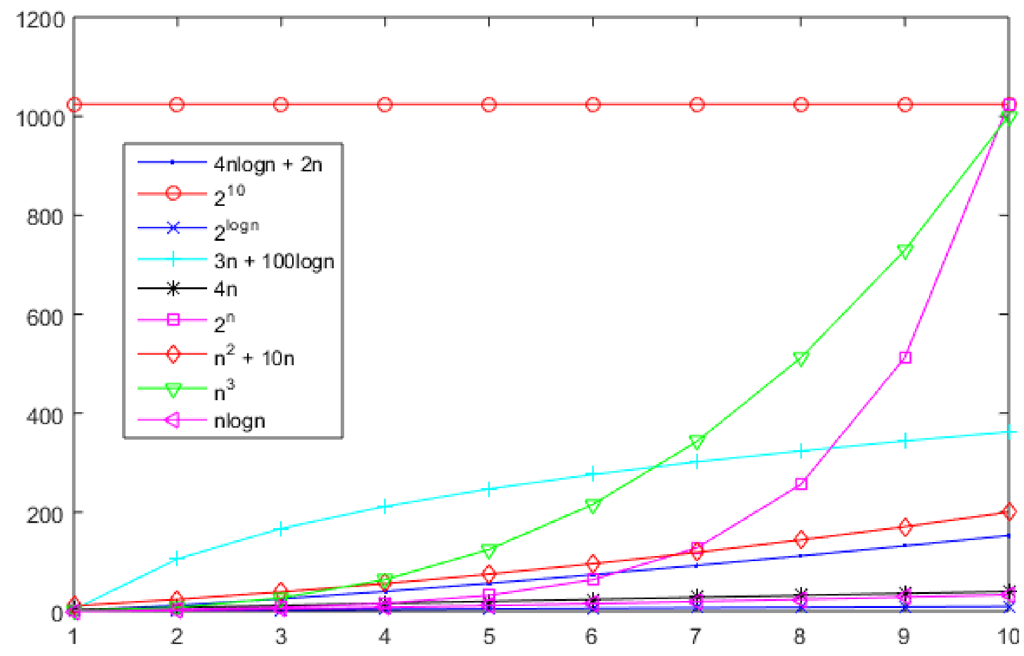
Q3: Ordering functions asymptotically

Order the following functions by asymptotic growth rate.

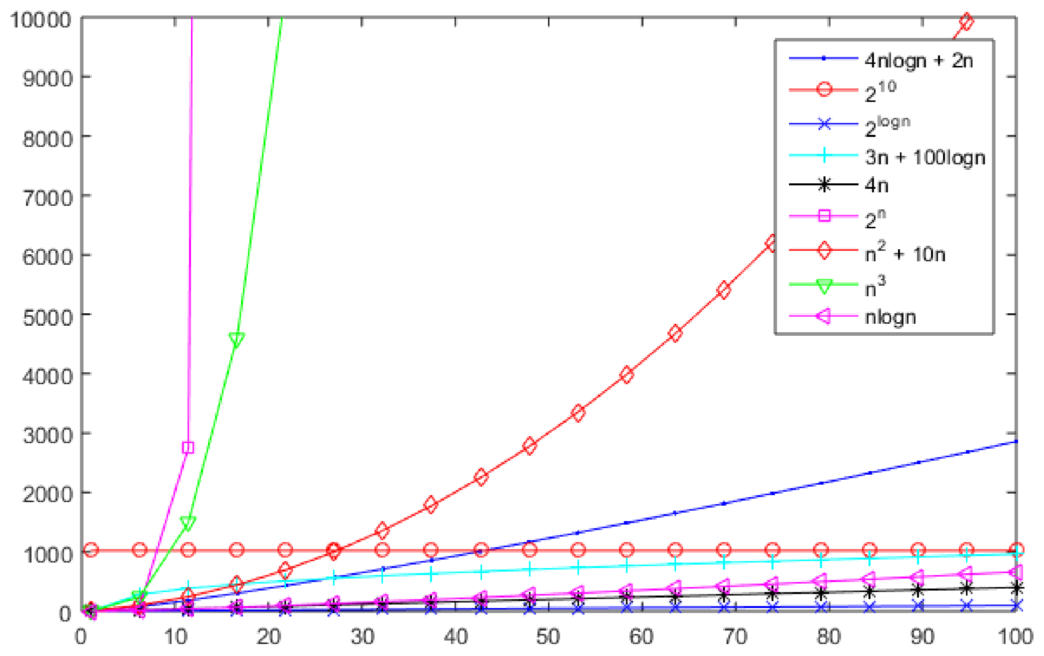
$$\begin{array}{ccc} 4n\log n + 2n & 2^{10} & 2^{\log n} \\ 3n + 100\log n & 4n & 2^n \\ n^2 + 10n & n^3 & n\log n \end{array}$$

Hints: Using Matlab, Python(download matplotlib module) or Excel to plot the graph of each function for n in the range [0,10], [0,100], [0,1000] respectively.

Q3: Answer-I



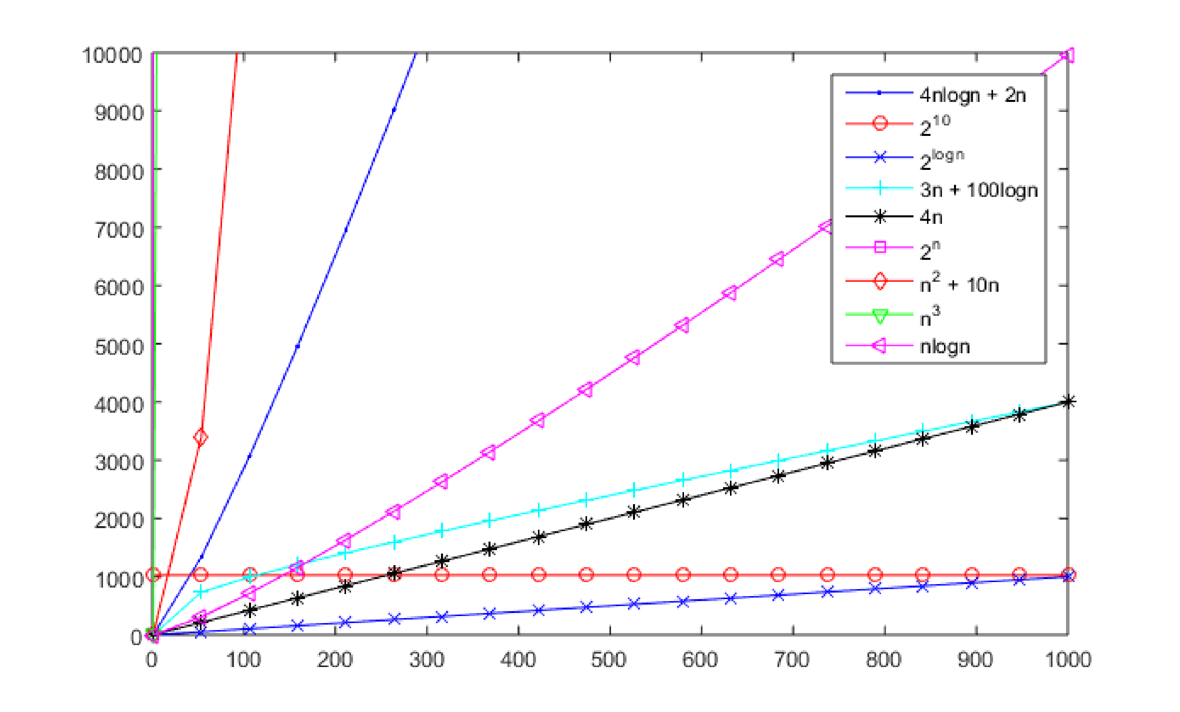
4nlogn + 2n
2^{10}
2^{logn}
3n + 100logn
4n
2^n
n^2 + 10n
n^3
nlogn

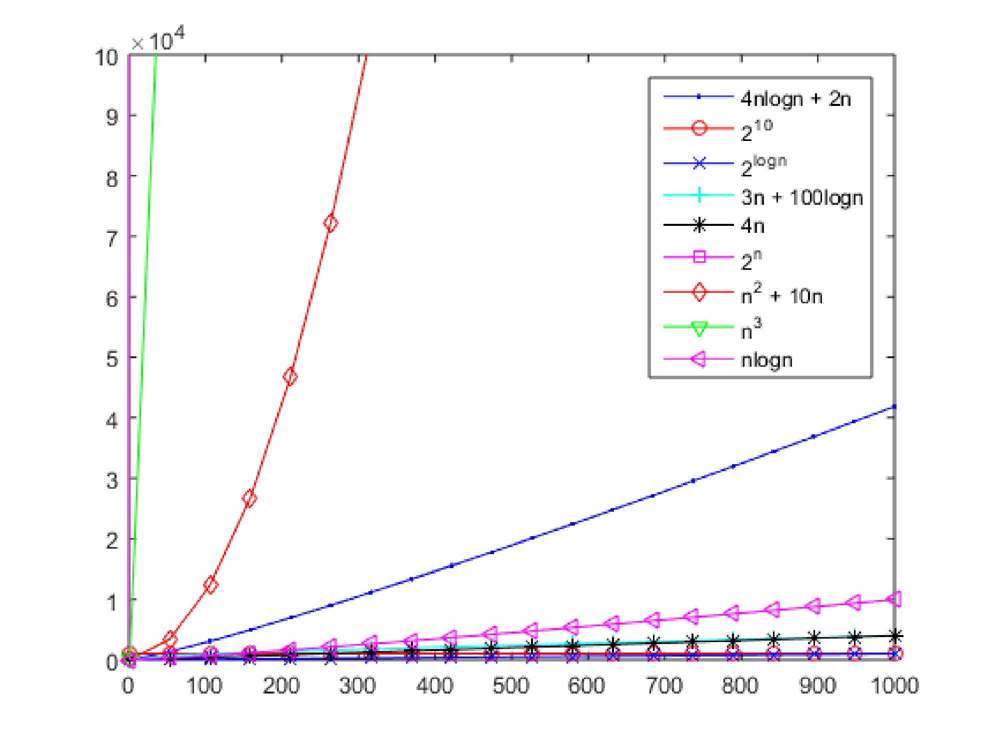


4nlogn + 2n
2^{10}
2^{logn}
3n + 100logn
4n
2^n
n^2 + 10n
n^3
nlogn

Q3: Answer-II

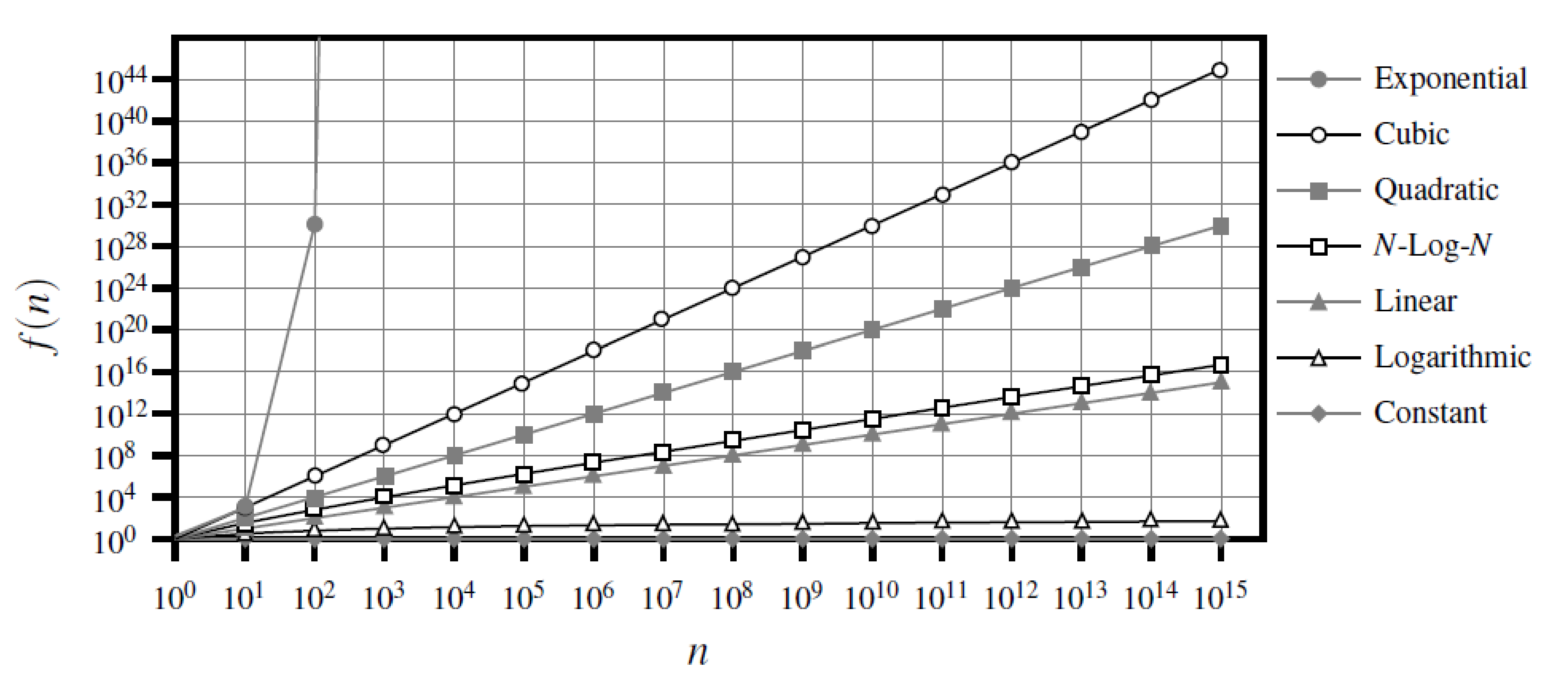
Answer: Asymptotically $2^n > n^3 > n^2+10n > 4n\log n+2n > 3n+100\log n$





Conclusion

constant	logarithm	linear	n-log-n	quadratic	cubic	exponential
1	$\log n$	n	$n \log n$	n^2	n^3	a^n

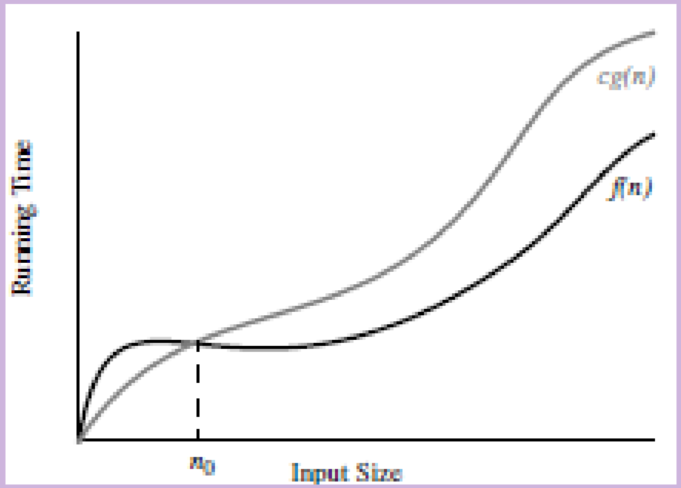


Definition of Big-Oh Notation

Definition: Let f(n) and g(n) be functions mapping positive integers to positive real numbers. We say that f(n) is O(g(n)) if there is a real constant c>0 and an integer constant $n_0 \geq 1$ such that

$$f(n) \leq cg(n), \text{ for } n \geq n_0$$

This definition is often referred to as the 'big-Oh' notation, for it is sometimes pronounced as 'f(n) is big-Oh of g(n).' Or you can say 'f(n) is order of g(n)'.



Q4: Prove Big Oh of a function

Use the definition of Big Oh notation shown before to

i) Show that 8n+5 is O(n).

ii) Show that $5n^4+3n^3+2n^2+4n+1$ is $O(n^4)$.

iii) Show that $5n^2+3n\log n+2n+5$ is $O(n^2)$.

iv) Show that 16nlogn+n is O(nlogn).

Rules: Characterizing functions in simplest terms.

Use the 7 functions. Our 7 functions are ordered by increasing growth rate in the following sequence.

constant	logarithm	linear	*n*-log-*n*	quadratic	cubic	exponential
1	$\log n$	n	$n \log n$	n^2	n^3	a^n

better → **worse**

Q4: Answer

Use the definition of Big Oh notation shown before to

i) Show that 8n+5 is O(n).

Proof: $8n + 5 \leq cn \; for \; c = 9 \; \forall n \geq 5$ or $c = 13 \; \forall n \geq 1$

ii) Show that $5n^4+3n^3+2n^2+4n+1$ is $O(n^4)$.

Proof: $5n^4 + 3n^3 + 2n^2 + 4n + 1 \leq cn^4 \;\; for \; c = 15 \;\; \forall n \geq 1$

iii) Show that $5n^2+3n\log n+2n+5$ is $O(n^2)$.

Proof: $5n^2 + 3nlogn + 2n + 5 \leq cn^2 \;\; for \; c = 15 \;\; \forall n \geq 1 (Because \; (logn)' = \frac{1}{(ln2)n} < \frac{5}{3} \Rightarrow growth \; rate \; of \frac{5}{3}n > that \; of logn \; and \frac{5}{3}n > logn \; when \; n = 1)$

iv) Show that 16nlogn+n is O(nlogn).

Proof: $16nlogn + n \leq cnlogn \; for \; c = 17 \; \forall n \geq 24 (optional) (Because \; (logn)' = \frac{1}{(ln2)n} < \frac{1}{16} \Rightarrow n > 23.08 \ldots \Rightarrow growth \; rate \; of \frac{1}{16}n > that \; of \; logn \; \forall n \geq 24$ and $\frac{9}{2}n > logn \; when \; n = 24)$

Q5: Time Complexity

13) What is time complexity of fun(n)?

```
def fun(n):
    count = 0
    m = n//2
    for i in range(n, 0, -m):
        m = m//2
        for j in range(0, i, 1):
            count += 1
    return count
```

A. $O(n^2)$
B. $O(n * \log n)$
C. $O(n)$
D. $O(n * \log n * \log n)$

Q5:Answer

13) What is time complexity of fun(n)?

```
def fun(n):
    count = 0
    m = n//2
    for i in range(n, 0, -m):
        m = m//2
        for j in range(0, i, 1):
            count += 1
    return count
```

A. $O(n^2)$
B. $O(n * \log n)$
C. $O(n)$
D. $O(n * \log n * \log n)$

Answer: C

Reason: given an n, you can try to print out count, then you will find count~n, which means time complexity is O(n).

Q6:Time complexity

i)What is the functionality of the function product()?
ii)What is the time complexity of this function?

```
def product(n, m):
    if n==0:
        return 0
    elif n==1:
        return m
    else:
        if n%2==1:
            return product(n//2, m)*2+m
        else:
            return product(n//2, m)*2
```

Q6: Answer-I

i)The functionality of the function product(n,m) is to calculate the product of n and m, that is n*m.

ii)The time complexity of this function is O(logn). The reason is on the next ppt, in which we design an algorithm to count the number of primitive operations in the function.

Q6:Answer-II

```
def product(n,m,count):
    if n==0:
        count+=1
        print(count)
        return 0
    elif n==1:
        print(count)
        count+=1
        return m
    else:
        if n%2==1:
            count+=1
            return product(n//2,m,count)*2+m
        else:
            count+=1
            return product(n//2,m,count)*2
product(8,4,0)
product(64,4,0)
product(256,4,0)
```

```
3
6
8
>>>
```

Reason: count is used to count the primitive operation in the function. For n=8, count=3; for n=64, count=6; for n=256, count=8, and so on. So the time complexity of the function product is O(logn).