COMP2003 Tutorial 5

Analysis of algorithms

Outline

- Motivation
- Analysis of algorithms
- Examples
- Practice questions

Outline

- Motivation
  - Components of a program
  - Brief introduction in algorithm analysis
  - An illustrative example
  - Motivation of algorithm analysis
  - Limit rule and growth rate of common functions
- Analysis of algorithms
- Examples
- Practice questions

Components of a program

- Program = algorithms + data structures
  - Algorithm is the way to handle the data and solve the problem
  - Data structure is the way you store the data for manipulation
- Usual way of writing a program
  - First come up with the algorithms and data structures to be used,
  - then use code to write the program

Brief introduction in algorithm analysis

- The same problem
  - May be solvable by more than one possible algorithm
    - Suppose we have 2 or more algorithms solving the same problem
    - For example, the problem is adding up a list of N consecutive integers
    - (e.g. 1,2,3,4,5…..,100)
- Which is better? How to define “better”?
  - Runs faster?
    - (Analysis the time complexity)
  - Uses less memory spaces?
    - (Analysis the space complexity)
- The general rule
  - As the input size grows, the time to run the algorithm and the amount of the memory spaces used grow
  - Thus, it is a good idea to compare the performance based on the input size

An illustrative example:

adding N consecutive integers

- Algorithm 1:
  - Using a for-loop
  - Example:

```
sum = 0
for: i=0 to N-1
   sum = sum + A[i]
return sum
```

  - Requires N additions
    - If the number of integer is 10,000, it requires 10,000 additions

- Algorithm 2:
  - Using a formula
  - Example:

```
sum = (A[0]+A[N-1])*(A[N-1]-A[N]+1)/2
return sum
```

  - Independent of the input size (N)
    - If the number of integer is 10,000, it requires only a few arithmetic operations

Motivation of algorithm analysis

- Suppose the input size is N
  - We want to roughly determine the running time or the space complexity in term of N
  - Too many terms is too clumsy
    - N^3+2N^2+4 additions V.S. $N^5+2N^{2/3}+4N$ additions
    - It is very hard to tell which one is better
- Asymptotic notations
  - Used to simplify the comparison among algorithms

Outline

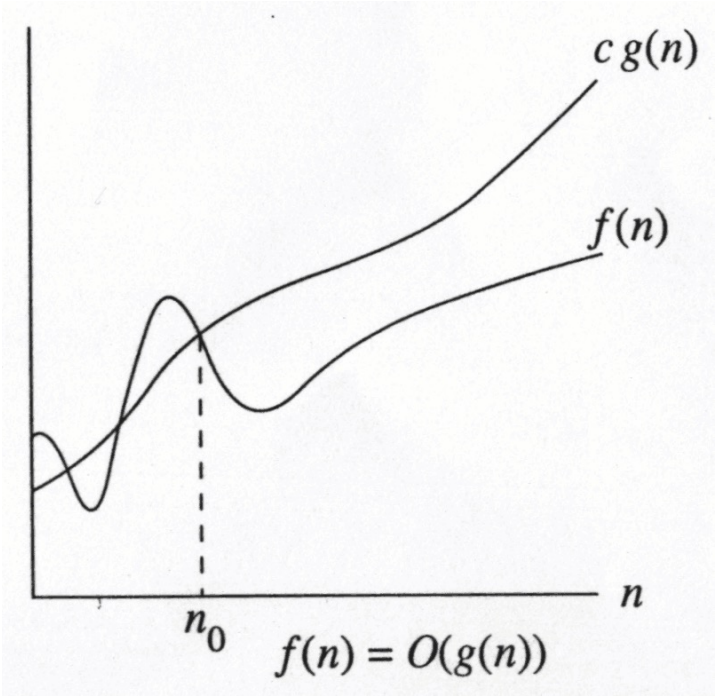
- Motivation
- Analysis of algorithms
  - Types of asymptotic notations
  - Big-Oh: Asymptotic Upper Bound
  - Big-Omega: Asymptotic Lower Bound
  - Big-Theta: Asymptotic Tight Bound
- Examples
- Practice questions

Types of asymptotic notations

- Three major types of asymptotic notations
  - Big-Oh: Asymptotic Upper Bound
  - Big-Omega: Asymptotic Lower Bound
  - Big-Theta: Asymptotic Tight Bound
- Measure the growth rate
  - A faster growth rate does not mean the algorithm always performs slower than the counterpart
  - It just means when the input size increases (e.g. N=10 becomes N=10,000), the running time grows much faster

Big-Oh: Asymptotic Upper Bound

- Definition:
  - f(N) = O(g(N)),
  - There are positive constants c and n_0 such that
    - $f(N) \leq c\, g(N)$ when $N \geq n_0$
- Example problems
  - How to prove that $2n^2 - 3n + 6 = O(n^2)$?
  - The problem is to find a pair of c and n_0 such that
    - $2n^2 - 3n + 6 \leq cn^2$ when $n \geq n_0$



To prove that mathematically:

Try some values of c and find out the corresponding n_0 which satisfies the condition.

1. $f(n) = O(n^2)$

(a) Suppose we choose c = 2:

$$2n^2 - 3n + 6 \leq 2n^2$$

$$-3n + 6 \leq 0$$

$$n \geq 2$$

So we can see that if we choose c = 2 and $n_0 = 2$, the condition is satisfied.

(b) Suppose we choose c = 3:

$$2n^2 - 3n + 6 \leq 3n^2$$

$$n^2 + 3n - 6 \geq 0$$

$$n \leq -4.37 \text{(ignored)} \text{ or } n \geq 1.37$$

So we can see that if we choose c = 3 and $n_0 = 2$, the condition is satisfied.

* There are other values of c and n_0 which satisfy the condition.

2. $f(n) = O(n^3)$ Suppose we choose c = 1:

$$2n^2 - 3n + 6 \le n^3$$

$$-n^3 + 2n^2 - 3n + 6 \le 0$$

$$n \ge 2$$

So we can see that if we choose c = 1 and $n_0 = 2$, the condition is satisfied.

* There are other values of c and n_0 which satisfy the condition.

3. $f(n) \neq O(n)$ For any given positive constant c:

$$2n^2 - 3n + 6 \le cn$$

$$2n^2 - (c+3)n + 6 \le 0$$

If $\Delta = (c+3)^2 - 48 < 0$, these is no solution. Otherwise the solution is

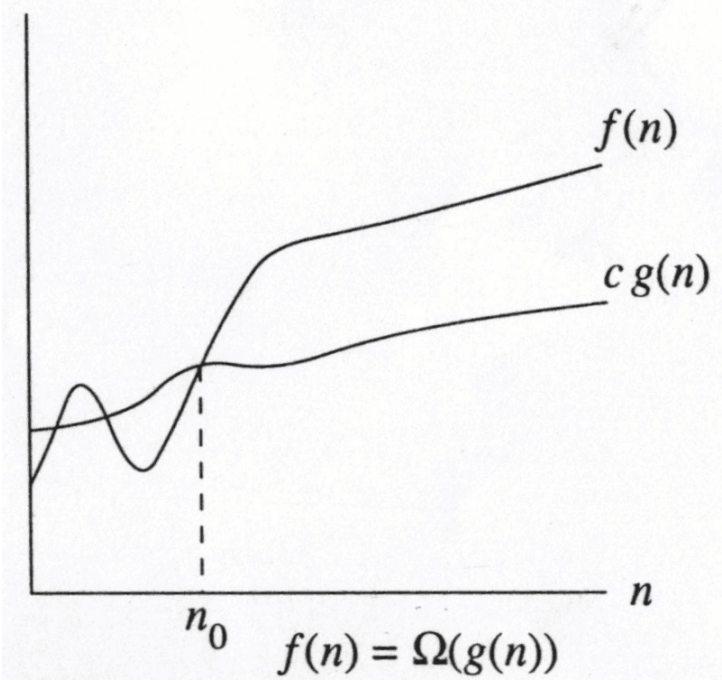
$$\frac{(c+3)-\sqrt{\Delta}}{4} \le n \le \frac{(c+3)+\sqrt{\Delta}}{4}$$

So we can see that if n is bigger than the right constant, the inequality does not hold. Hence $f(n) \neq O(n)$.

4. $f(n) \neq O(1)$: Similar to $f(n) \neq O(n)$

Big-Omega: Asymptotic Lower Bound

- Definition
  - $f(N) = \Omega(g(N))$
  - There are positive constants c and n_0 such that
    - $f(N) \geq c\, g(N)$ when $N \geq n_0$
- Example problem
  - How to prove that $2n^2 - 3n + 6 = \Omega(n^2)$?
  - The problem is to find a pair of c and n_0 such that
    - $2n^2 - 3n + 6 \geq cn^2$ when $n \geq n_0$



To prove that mathematically:

Try some values of c and find out the corresponding n_0 which satisfies the condition.

$f(n) = \Omega(n^2)$

Suppose we choose c = 1:

$$2n^2 - 3n + 6 \geq n^2$$

$$n^2 - 3n + 6 \geq 0$$

$$n > 0$$

So we can see that if we choose c = 1 and $n_0 = 1$, the condition is satisfied.

3. $f(n) \neq \Omega(n^3)$:

Assume there exists positive constants c and n_0 such that for all $n > n_0$, $2n^2 - 5n + 6 \geq cn^3$. We have

$$cn^3 \leq 2n^2 - 5n + 6 < 2n^2 + 6 \Rightarrow$$

$$n < \frac{2n^2}{cn^2} + \frac{6}{cn^2} < \frac{2}{c} + \frac{6}{cn_0^2}$$

Which means for any $n \geq \frac{2}{c} + \frac{6}{cn_0^2}$ the condition is not satisfied, contradicting our assumption. This in turn shows that no positive constants c and n_0 exist for the assumption, hence $f(n) \neq \Omega(n^3)$.

Big-Theta: Asymptotic Tight Bound

Definition:

Consider a function f(n) which is non-negative for all integers $n \geq 0$.

$f(n) = \Theta(g(n))$ (read as "f of n is big-theta of g of n") iff:

There exists positive constants c_1 , c_2 and n_0 such that

$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ for all integers $n \geq n_0$.

And we say g(n) is an asymptotic tight bound of f(n).

Generally, $f(n) = \Theta(g(n)) \Leftrightarrow \begin{cases} f(n) = O(g(n)) \\ f(n) = \Omega(g(n)) \end{cases}$

Example:

Consider $f(n) = 2n^2 - 3n + 6$. Then $f(n) = \Theta(n^2)$

To prove that mathematically: We only have to prove $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ respectively, where $g(n) = n^2$.

Estimating the growth rate of functions involving only polynomial terms

When estimating the asymptotic tight bound of f(n), there is a simple method as described in following procedure:

1. Ignore the low order terms.
2. Ignore the constant coefficient of the most significant term.
3. The remaining term is the estimation.

Proof will be given later with the limit rules.

Example:

Consider $f(n) = 2n^2 - 3n + 6$. By applying the above, we

1. Ignore all the lower order terms. Therefore, we have $2n^2$.
2. Ignore the constant coefficient of the most significant term. We have n^2.
3. The remaining term is the estimation result, i.e. $f(n) = \Theta(n^2)$.

To prove $f(n) = \Theta(n^2)$

We need two things:

1. $f(n) = 2n^2 - 3n + 6 = O(n^2)$
2. $f(n) = 2n^2 - 3n + 6 = \Omega(n^2))$

For condition 1:

$f(n) = 2n^2 - 3n + 6$

$\leq 2n^2 + n^2$ (assume $n \geq \sqrt{6}$)

$= 3n^2$

Which means we have c=3 and $n_0 \geq \sqrt{6}$ satisfying the condition.

For condition 2:

$f(n) = 2n^2 - 3n + 6$

$\geq 2n^2 - n^2$ (since $n^2 \geq 3n - 6$ always holds)

$= n^2$

Which means we have c=1 and n_0=1 satisfying the condition.

From above 2, we can choose $c_1 = 1$, $c_2 = 3$, $n_0 = \sqrt{6}$ and we have :

$n^2 \leq f(n) \leq 3n^2$ for all $n \geq \sqrt{6}$

Proof of Slide 11:

With the limit rule, we now can prove:

For any $f(n)=c_1g_1(n) + c_2g_2(n)+\ldots+ c_mg_m(n)$ where c_i is constant and $g_i(n)$ are functions such that their order of magnitudes decrease with i, we have:

$f(n)/g_1(n) = c_1 + c_2g_2(n)/g_1(n) +\ldots+ c_mg_m(n)/g_1(n)$

$= c_1$ (For $n\rightarrow\infty$)

Which means, $f(n) = \Theta(g_1(n))$, which depends on only the term of the highest order of magnitude. Note here that $g_i(n)$ is arbitrary.

Problems:(This example requires knowledge of differentiation in calculus)

- If the limit of both f(n) and g(n) approach 0 or both approach 1, we need to apply $L'Hôpital's$ rule:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{f'(n)}{g'(n)}$$

where f’ (n) and g’ (n) denote the derivatives of f(n) and g(n). Note that the rule can be applied multiple times.

In cases where the limit does not exist, we have to use the definition.

Exercise: find out the relationship between the pairs of functions. Determine whether *f* is O(g), Ω(g) or Θ(g).

1. $f(n)=\log\sqrt{n}, g(n)=\sqrt{\log n}.$ f(n)= Ω(g(n))

2. $f(n)=2n^2-n+6, g(n)=n^3+8.$ f(n)= O(g(n))

3. $f(n)=\begin{cases}10^9, \text{if } n \text{ is even},\\ n, \text{if } n \text{ is odd}.\end{cases}$ $g(n)=\begin{cases}n, \text{if } n\geq 1000,\\ n^2, \text{if } n<1000.\end{cases}$ f(n)= O(g(n))

Some common asymptotic identities

$f(n)$	Asymptotic
c	$\Theta(1)$
$\sum_{i=0}^{k} c_i n^i$	$\Theta(n^k)$
$\sum_{i=1}^{n} c_i i^k, k > 0$	$\Theta(n^{k+1})$
$\sum_{i=0}^{n} c_i r^i, r > 1$	$\Theta(r^n)$
$\sum_{i=1}^{n} (c_i / i)$	$\Theta(\log n)$
$n!$	$\Theta(\sqrt{n}(\frac{n}{e})^n)$
$\log(n!)$	$\Theta(n \log n)$

$$\sum_{i=0}^{n} i^k \cong \int_0^n x^k dx = \frac{1}{k+1} n^{k+1}$$

$$\sum_{i=1}^{n} \frac{1}{i} \cong \int_1^n \frac{1}{x} dx = \log n$$

$$n! \cong \sqrt{2\pi n}(\frac{n}{e})^n \quad (Stirling's\ formula)$$

Where $f(n) \cong g(n)$ denotes:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$$

Example: Proof of log(*n*!) = Θ(*n*log*n*).

We need to prove f(*n*) = log(*n*!) = O(*n*log*n*) = Ω(*n*log*n*)

1. Proof of log(*n*!) = O(*n*log*n*) :

$n! = n(n-1)(n-2)\ldots 3 \times 2 \times 1$

$\leq n^n$

Therefore, log(n!) ≤ nlogn, and hence log(*n*!) = O(*n*log*n*).

2. Proof of log(*n*!) = Ω(*n*log*n*) :

$$n! = n(n-1)(n-2)\ldots 3\times 2\times 1$$

$$\geq \underbrace{\frac{n}{2}\cdot\frac{n}{2}\cdot\ldots\frac{n}{2}}_{\lceil n/2\rceil \text{ times}}\cdot\underbrace{1\cdot 1\ldots\cdot 1}_{\lfloor n/2\rfloor \text{ times}}$$

$$\geq \left(\frac{n}{2}\right)^{n/2} \Rightarrow \log(n!) \geq \frac{1}{2} n \log\frac{n}{2}$$

From this we conclude that: $f(n) = \Omega(n\log\frac{n}{2}) = \Omega(n\log n - n\log 2) = \Omega(n\log n)$

Thus log(*n*!) = Ω(*n*log*n*).

Order of growth rate of some common functions

Growth rate: Slowest

- $\Theta(c)$ where c>0 is a constant.
- $\Theta(\log^k n)$ where k>0 is a constant (the larger the k, the faster the growth rate)
- $\Theta(n^p)$ where 0<p<1 is a constant (the larger the p, the faster the growth rate)
- $\Theta(n)$
- $\Theta(n \log n)$
- $\Theta(n^k)$ where k>1 is a constant (the larger the k, the faster the growth rate)
- $\Theta(k^n)$ where k>1 is a constant (the larger the k, the faster the growth rate)
- $\Theta(n!)$

Growth rate: Fastest

Exercise: Order the following functions by Big-Θ notation:

$$1/n, 4^n, 2^{100}, 6n\log n, 2^{\log n}, 2^n, \log^2 n, \sqrt{n}, 4n^{1.5}, n^2 \log n, n\log(n^2).$$

slowest

$$1/n, 2^{100}, \log^2 n, \sqrt{n}, 2^{\log n}, n\log(n^2), 6n\log n, 4n^{1.5}, n^2 \log n, 2^n, 4^n.$$

fastest

Outline

- Motivation
- Analysis of algorithms
- Examples
  - Assumptions in algorithm analysis
  - Analyze loops
  - Analyze recursive functions
  - Maximum Consecutive Sum
- Practice questions

Assumptions in algorithm analysis

- Assumptions
  - For this course, we assumed that instructions are executed one after another (sequential), without concurrent operations
  - We use RAM (Random Access Model), in which each operation (e.g. +, -, x, /,=) and each memory access take one run-time unit O(1)
  - Loops and functions can take multiple time units.

Example 1: Analyze the bubble sort

- Analyze the bubble sort algorithm on an array of size N:

```
for: i=0 to N-1
   for: j=N-1 to i+1
    if A[j]<A[j-1]                  <-- constant time O(1)
       swap(A[j], A[j-1])           <-- constant time O(1)
```

- It takes at most (N-1)+(N-2)+…+1 = N(N-1)/2 swaps
- Assume each swap takes O(1) time, we have an $O(N^2)$ algorithm

Example 2: Analyze a loop

- Loops (in C++ code format)

```
int sum(int n)
  int partialSum;
  partialSum = 0;
  for(int i=0;i<n;i++)
     partialSum += i*i*i;
  return partialSum
}
```

constant time O(1)

O(1)+(n+1)*O(1)+n*O(1) = O(n)

O(1)+2*O(1)+O(1) = O(1)

constant time O(1)

- Time complexity
  = 1st + 2nd*3rd + 4th
  = O(1) + O(n)*O(1) + O(1)
  = O(n)

Example 3: Analyze nested loops

- Nested loops (in pseudo-code format)

```
sum = 0;                  <-- O(1)
for i=0 to n              <-- O(n)
    for j=0 to n          <-- O(n)
        for k=0 to n      <-- O(n)
            sum++;        <-- O(1)
```

- Time complexity
  = 1st + 2nd*3rd*4th*5th
  = O(1)+O(n)*O(n)*O(n)*O(1)
  = $O(n^3)$

Example 4: Analyze recursion (1/3)

- Recursion consists of 2 main parts:
  - Base case -- directly returns something
  - Recursive case -- calls itself again with a smaller input
- Example: factorial

```
int factorial(int n) {
 if(n<=0)
   return 1;          <-- base case
 else
   return n*factorial(n-1);   <-- recursive case
}
```

Example 4: Analyze recursion (2/3)

- Recursion analysis
  - Base case typically takes constant time O(1)
  - Recursive case takes similar time with smaller input:
    - Suppose for input size = n it takes T(n) time, then for input size = n-1 it takes T(n-1) time
- We have this set of recurrence:

$$T(n) = \begin{cases} O(1) & \text{for } n \leq 0 \\ T(n-1) + O(1) & \text{otherwise} \end{cases}$$

Example 4: Analyze recursion (3/3)

- Recursion derivation

$$
\begin{aligned}
T(n) &= T(n-1) + O(1) \\
&= \left[T(n-2) + O(1)\right] + O(1) \\
&= \left[T(n-3) + O(1)\right] + O(1) + O(1) \\
&= \ldots \\
&= T(0) + (n+1) \times O(1) \\
&= O(1) + (n+1) \times O(1) \\
&= O(n)
\end{aligned}
$$

Example 5: Analyze recursion (1/4)

- Recursion derivation
- The mergesort algorithm (described in later tutorials) is a recursion and has the following recurrence:

$$T(n) = \begin{cases} O(1) & \text{for } n = 1 \\ 2T(\frac{n}{2}) + O(n) & \text{for } n > 1 \end{cases}$$

Example 5: Analyze recursion (2/4)

- Usually we re-write the recurrence to the following for simplicity in derivation:

$$T(n) = \begin{cases} c & \text{for } n = 1 \\ 2T(\frac{n}{2}) + cn & \text{for } n > 1 \end{cases}$$

Example 5: Analyze recursion (3/4)

$$T(n) = 2T(\frac{n}{2}) + cn$$

$$= 2\left[2T(\frac{n}{2^2}) + c\frac{n}{2}\right] + cn$$

$$= 2^2 T(\frac{n}{2^2}) + cn + cn$$

$$= 2^2\left[2T(\frac{n}{2^3}) + c\frac{n}{2^2}\right] + cn + cn$$

$$= \ldots$$

$$= 2^k T(\frac{n}{2^k}) + k \times cn$$

What's next??

Example 5: Analyze recursion (4/4)

- Let n = 2^k, then we can continue like this:

$$T(n) = 2^k T(\frac{n}{2^k}) + kcn$$

$$= nT(\frac{n}{n}) + cn\log_2 n$$

$$= nT(1) + cn\log_2 n$$

$$= n \times O(1) + \frac{c}{\log 2} n\log n$$

$$= O(n) + O(n\log n)$$

$$= O(n\log n)$$

$log_2 n = k log_2 2$ => $k = log_2 n$

Example 6: Analyze recursion (1/4)

- In the lecture…
  - ```
    int fun(int n) {
      int a,b;
      if(n<=4)
        return 0;
      else {
        a = fun(n/2);
        b = fun(n/4);
        return a+b;
      }
    }
    ```
- The recurrence is therefore:

$$T(n) = \begin{cases} O(1) & \text{for } n \le 4 \\ T(\frac{n}{2}) + T(\frac{n}{4}) + O(1) & \text{otherwise} \end{cases}$$
    ```

Example 6: Analyze recursion (2/4)

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + O(1)$$

$$= \left[T(\frac{n}{2^2}) + T(\frac{n}{2^3}) + O(1) \right] + T(\frac{n}{2^2}) + O(1)$$

$$= 2T(\frac{n}{2^2}) + T(\frac{n}{2^3}) + (1+1) \times O(1)$$

$$= 2\left[T(\frac{n}{2^3}) + T(\frac{n}{2^4}) + O(1) \right] + T(\frac{n}{2^3}) + (1+1) \times O(1)$$

$$= 3T(\frac{n}{2^3}) + 2T(\frac{n}{2^4}) + (1+1+2) \times O(1)$$

$$= 3\left[T(\frac{n}{2^4}) + T(\frac{n}{2^5}) + O(1) \right] + 2T(\frac{n}{2^4}) + (1+1+2) \times O(1)$$

$$= 5T(\frac{n}{2^4}) + 3T(\frac{n}{2^5}) + (1+1+2+3) \times O(1)$$

$$= \ldots$$

The coefficients of the first 2 terms are in the Fibonacci sequence, and the third term is the sum of such sequence… so…

Example 6: Analyze recursion (3/4)

- Let $n = 2^k$, and F_i be the i-th term of the Fibonacci sequence, we have:

$$T(n) = 5T(\frac{n}{2^4}) + 3T(\frac{n}{2^5}) + (1+1+2+3) \times O(1)$$

$$= \ldots$$

$$= F_{k-1}T(\frac{n}{2^{k-2}}) + F_{k-2}T(\frac{n}{2^{k-1}}) + \sum_{i=1}^{k-2} F_i \times O(1)$$

$$= F_{k-1}T(4) + F_{k-2}T(2) + \sum_{i=1}^{k-2} F_i \times O(1)$$

What's left is how to determine those F_i's…

Example 6: Analyze recursion (4/4)

- Properties of Fibonacci sequence (proof omitted):

$$F_0 + F_1 + F_{2+} \ldots + F_n = F_{n+2} - 1$$

$$F_k = \frac{\phi^k - (1-\phi)^k}{\sqrt{5}}, \text{ where } \phi = \frac{1+\sqrt{5}}{2}$$

$$F_k < (\frac{5}{3})^k$$

- We then have:

$$T(n) = F_{k-1}T(4) + F_{k-2}T(2) + \sum_{i=1}^{k-2} F_i \times O(1)$$

$$= F_{k-1}O(1) + F_{k-2}O(1) + F_{k-2}O(1) + F_{k-3}O(1) + F_{k-4}O(1) + \ldots + F_1O(1)$$

$$= F_{k-2}O(1) + (F_{k+1} - 1)O(1)$$

$$< (\frac{5}{3})^{k-2}O(1) + (\frac{5}{3})^{k+1}O(1) - O(1) < (2^{k-2} + 2^{k+1})O(1)$$

$$= 2^k \frac{9}{4}O(1) = O(2^k) = O(n)$$

Maximum Consecutive Sum

- Two algorithms presented:
  - Brute forcing every possible consecutive sub-array and get the largest sum. $O(n^3)$ complexity
  - Divide and Conquer. Solve the left and right half of the array and then solve the middle sub-array and compare them to get the max-sum. $O(n\log n)$ time.
    - You can apply the above techniques to analysis the time complexities of these two algorithms
  - Refer to lecture notes for more details.
- Actually, an O(n) algorithm exists for this problem

Outline

- Motivation
- Analysis of algorithms
- Examples
- Practice questions

Practice Question 1

1. Suppose $T_1(n) = O(f(n))$, $T_2(n) = O(f(n))$. Which of the followings are TRUE:

   $(a)\ T_1(n) + T_2(n) = O(f(n))$

   $(b)\ \frac{T_1(n)}{T_2(n)} = O(1)$

   $(c)\ T_1(n) = O(T_2(n))$

2. $f(n) = n + n\log n =$ _____.

   $(a)\ O(n)$

   $(b)\ O(n^2)$

   $(c)\ \Omega(n^2)$

   $(d)\ \Omega(n\log n)$

   $(e)\ \Theta(n\log n)$

   $(f)\ \Theta(n)$

Practice Question 2

- Given the recurrence:

$$T(n) = \begin{cases} O(1) & \text{for } n = 1 \\ 9T(\frac{n}{3}) + n & \text{for } n > 1 \end{cases}$$

- Solve the recurrence and give the tightest possible bound for the Big-O complexity analysis of it.

Summary

- Motivation
  - Components of a program
  - Brief introduction in algorithm analysis
  - An illustrative example
  - Motivation of algorithm analysis
  - Limit rule and growth rate of common functions
- Analysis of algorithms
  - Types of asymptotic notations
  - Big-Oh: Asymptotic Upper Bound
  - Big-Omega: Asymptotic Lower Bound
  - Big-Theta: Asymptotic Tight Bound
- Examples
  - Assumptions in algorithm analysis
  - Analyze loops
  - Analyze recursive functions
  - Maximum Consecutive Sum
- Practice questions

Thank you!