Complexity Analysis: Asymptotic Analysis

Recall

- What is the measurement of algorithm?
- How to compare two algorithms?
- Definition of Asymptotic Notation

Today Topic

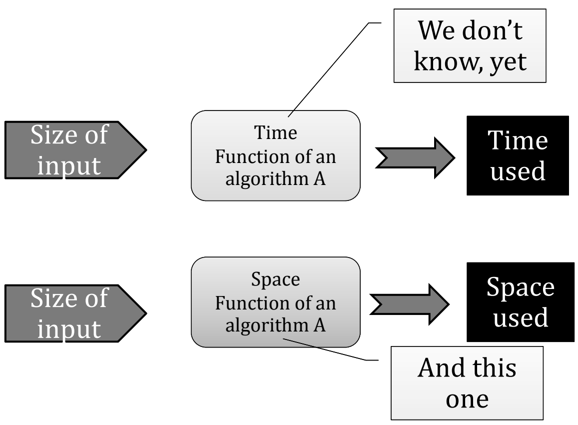
- Finding the asymptotic **upper** bound of the algorithm

WAIT…

Did we miss something?

The resource function!

Resource Function



From Experiment

- We count the number of instructions executed
- Only count some instruction
  - One that's promising

Why instruction count?

- Does instruction count == time?
  - Probably
  - But not always
  - But…
    - Usually, there is a strong relation between instruction count and time if we count the most occurred instruction

Why count just some?

- Remember the findMaxCount()?
  - Why we count just the max assignment?
  - Why don't we count everything?
- What if, each max assignment takes N instruction
  - That starts to make sense

Time Function = instruction count

- Time Function = instruction count
- Time Function = instruction count
- Time Function = instruction count

COMPUTE O()

Interesting Topics of Upper Bound

- Rule of thumb!
- We neglect
  - Lower order terms from addition
    - E.g. $n^3+n^2 = O(n^3)$
  - Constant
    - E.g. $3n^3 = O(n^3)$

Remember that we use = instead of (more correctly) $\in$

Why Discard Constant?

- From the definition
  - We can scale by adjusting the constant c
- E.g. $3n = O(n)$
  - Because
    - When we let c >= 3, the condition is satisfied

Why Discard Lower Order Term?

- Consider
  - $f(n) = n^3+n^2$
  - $g(n) = n^3$
- If $f(n) = O(g(n))$
  - Then, for some c and n_0
    - $c * g(n)-f(n) > 0$
    - Definitely, just use any c >1

Why Discard Lower Order Term?

- Try c = 1.1

$1.1 * g(n)-f(n) = 0.1n^3-n^2$

- Does $0.1n^3-n^2 > 0$
- It is when
  - $0.1n > 1$
  - E.g., n > 10

$$\begin{aligned} 0.1n^3-n^2 &> 0 \\ 0.1n^3 &> n^2 \\ 0.1n^3/n^2 &> 1 \\ 0.1n &> 1 \end{aligned}$$

Lower Order only?

- In fact,
  - It's only the dominant term that count
- Which one is dominating term?
  - The one that grows faster
- Why?
  - Eventually, it is $g^*(n)/f^*(n)$
    - If g(n) grows faster,
      - $g(n)/f^*(n)$ > some constant
      - E.g, $\lim g(n)/f^*(n) \rightarrow$ infinity

The non-dominant term

The dominant term

What dominating what?

Left side dominates

n^a	n^b (when a > b)
n log n	n
$n^2 \log n$	$n \log^2 n$
c^n	n^c
Log n	1
n	log n

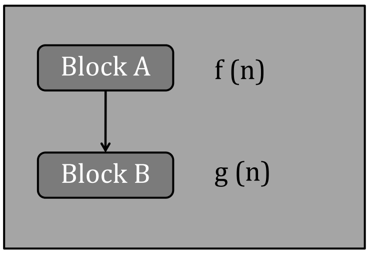
Putting into Practice

- What is the asymptotic class of
  - $0.5n^3+n^4-5(n-3)(n-5)+n^3\log^8 n+25+n^{1.5}$ **$O(n^4)$**
  - $(n-5)(n^2+3)+\log(n^{20})$ **$O(n^3)$**
  - $20n^5+58n^4+15n^{3.2}*3n^2$ **$O(n^{5.2})$**

Asymptotic Notation from Program Flow

- Sequence
- Conditions
- Loops
- Recursive Call

Sequence

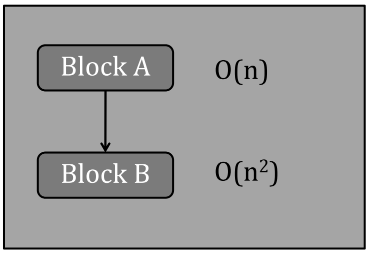


$f(n) + g(n) =$

$O(\max(f(n),g(n)))$

Example

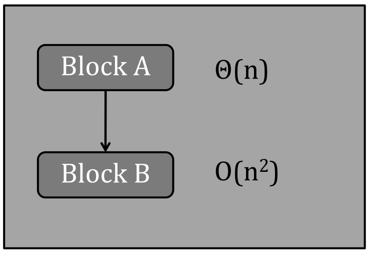
$\boxed{f(n) + g(n) = O(\max (f(n),g(n))}$



$O(n^2)$

Example

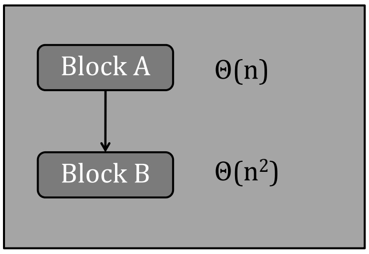
$\boxed{f(n) + g(n) = O(\max (f(n),g(n))}$



$O(n^2)$

Example

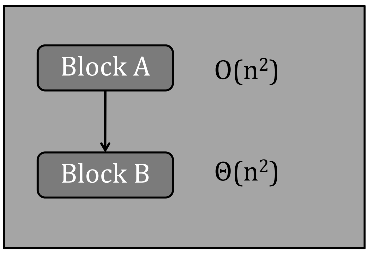
$\boxed{f(n) + g(n) = O(\max (f(n),g(n))}$



$\Theta(n^2)$

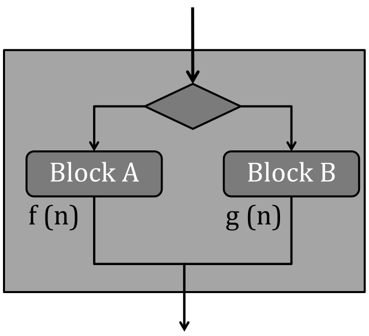
Example

$\boxed{f(n) + g(n) = O(\max (f(n),g(n))}$



$\Theta(n^2)$

Condition



$O(\max (f(n),g(n)))$

Loops

```
for (i = 1;i <= n;i++) {
  P(i)
}
```

$$\sum_{i=1}^{n} t_i$$

Let P(i) takes time t_i

Example

```
for (i = 1;i <= n;i++) {
  sum += i;
}
```

$$\sum_{i=1}^{n} \Theta(1) = \Theta(n)$$

sum += i ➔ $\Theta(1)$

Why don't we use max(t_i)?

- Because the number of terms is not constant

```
for (i = 1;i <= n;i++) {
  sum += i;
}
```

$\Theta(n)$

```
for (i = 1;i <= 100000;i++) {
  sum += i;
}
```

$\Theta(1)$

With large constant

Example

```
for (j = 1;j <= n;j++) {
  for (i = 1;i <= n;i++) {
    sum += i;
  }
}
```

sum += i ➔ $\Theta(1)$

$$\begin{aligned} \sum_{j=1}^{n}\sum_{i=1}^{n} \Theta(1) &= \sum_{j=1}^{n} \Theta(n) \\ &= \Theta(n) + \Theta(n) + \ldots + \Theta(n) \\ &= \Theta(n^2) \end{aligned}$$

Example

```
for (j = 1;j <= n;j++) {
  for (i = 1;i <= j;i++) {
    sum += i;
  }
}
```

sum += i ➔ $\Theta(1)$

$$\begin{aligned} \sum_{j=1}^{n}\sum_{i=1}^{j} \Theta(1) &= \sum_{j=1}^{n} \Theta(j) \\ &= \sum_{j=1}^{n} cj \\ &= c\sum_{j=1}^{n} j \\ &= c\left(\frac{n(n+1)}{2}\right) \\ &= \Theta(n^2) \end{aligned}$$

Example : Another way

```
for (j = 1;j <= n;j++) {
  for (i = 1;i <= j;i++) {
    sum += i;
  }
}
```

sum += i ➔ $\Theta(1)$

$$\begin{aligned} \sum_{j=1}^{n}\sum_{i=1}^{j} \Theta(1) &= \sum_{j=1}^{n} \Theta(j) \\ &= \sum_{j=1}^{n} O(n) \\ &= O(n^2) \end{aligned}$$

Example

Your turn

```
for (j = 2;j <= n-1;j++) {
  for (i = 3;i <= j;i++) {
    sum += i;
  }
}
```

sum += i ➔ $\Theta(1)$

Example : While loops

```
While (n > 0) {
  n = n - 1;
}
```

$\Theta(n)$

Example : While loops

```
While (n > 0) {
  n = n - 10;
}
```

$\Theta(n/10) = \Theta(n)$

Example : While loops

```
While (n > 0) {
  n = n / 2;
}
```

$\Theta(\log n)$

Example : Euclid's GCD

```
 function gcd(a, b) {
     while (b > 0) {
         tmp = b
         b   = a mod b
         a   = tmp
     }
     return a
}
```

Example : Euclid's GCD

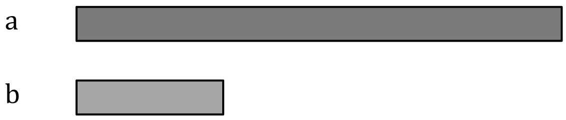
Until the modding one is zero

```
 function gcd(a, b) {
     while (b > 0) {
         tmp = b
         b   = a mod b
         a   = tmp
     }
     return a
}
```

How many iteration?

Compute mod and swap

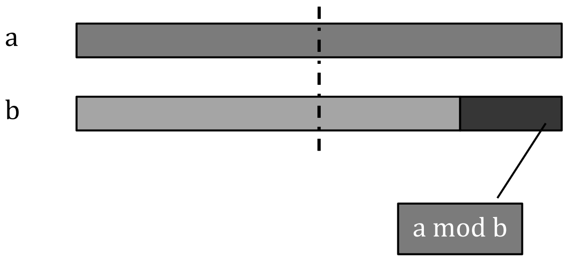
Example : Euclid's GCD



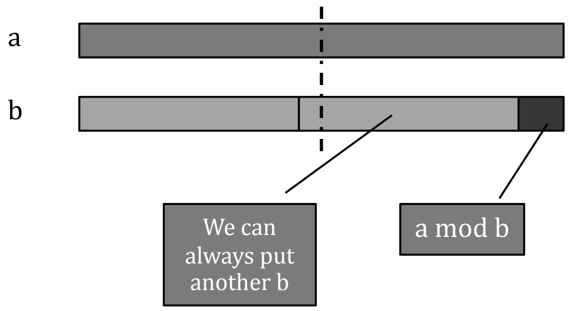
Example : Euclid's GCD

If a > b
a mod b < a / 2

Case 1: b > a / 2



Case 1: b ≤ a / 2



Example : Euclid's GCD

```
function gcd(a, b) {
    while (b > 0) {
        tmp = b
        b   = a mod b
        a   = tmp
    }
    return a
}
```

O(log n)

B always reduces at least half

RECURSIVE PROGRAM

Code that calls itself

```
void recur(int i) {
  // checking termination

  // do something

  // call itself
}
```

Find summation from 1 to i

```
int sum(int i) {
  //terminating condition
  if (i == 1) //return 1;

  // recursive call
  int result;
  result = i + sum(i-1);
  return result;
}

void main() {  printf("%d\n",sum()); }
```

Order of Call: Printing Example

```
void seq(int i) {
  if (i == 0)
  return;


  printf("%d ",i);
  seq(i - 1);


}
```

```
void seq(int i) {
  if (i == 0)
  return;


  seq(i - 1);
  printf("%d ",i);


}
```

Draw Triangle

```
void drawtri(int start,int i,int n) {
  if (i <= n) {
    for (int j = start;j <= i+start-1;j++) {
      printf("%d ",j);
    }
    printf("\n");
    drawtri(start,i + 1,n);

  }
}
```

Programming Recursive Function

- First, define what will the function do
  - Usually, it is related to "smaller" instant of problem
- Write the function to do that in the trivial case
- Call itself to do the defined things

Analyzing Recursive Programming

- Use the same method, count the most occurred instruction
- Needs to consider every calls to the function

Example

```
void sum(int i) {
  if (i == 0) return 0;

  int result;
  result = i + sum(i - 1);
  return result;
}
```

$\Theta(n)$

Example 2

```
void sum(int i) {
  if (i == 0) return 0;

  int result;
  result = i + sum(i - 1);
  int count = 0;
  for (int j = 0;j < i;j++) {
    count++;
  }
  return result;
}
```

$\Theta(n^2)$

Theorem

- $T(n)= \Sigma T(a_i n) + O(N)$
  - If $\Sigma a_i < 1$ then
    - $T(n) = O(n)$

$T(n) = T(0.7n) + T(0.2n) + T(0.01)n + 3n$
$= O(n)$

Recursion

```
void try( n ){

  if ( n <= 0 ) return 0;

  for ( j = 1; j <= n ; j++)
    sum += j;

  try (n * 0.7)
  try (n * 0.2)
}
```

Recursion

```
void try( n ){
  if ( n <= 0 ) return 0;
  for ( j = 1; j <= n ; j++)
    sum += j;
  try (n * 0.7)
  try (n * 0.2)
}
```

terminating — $\Theta(1)$

process — $\Theta(n)$

recursion — $T(0.7n) + T(0.2n)$

$T(n)$

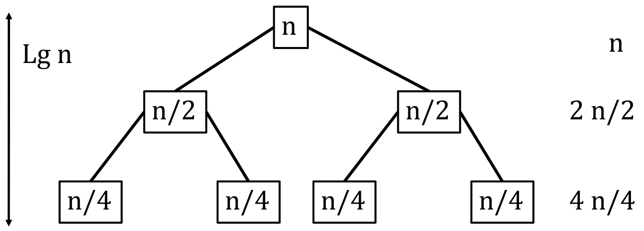
$T(n) = T(0.7n) + T(0.2n) + O(n)$

Guessing and proof by induction

- $T(n) = T(0.7n) + T(0.2n) + O(n)$
- Guess: $T(n) = O(n)$, $T(n) \le cn$
- Proof:
- Basis: obvious
- Induction:
  - Assume $T(i < n) = O(i)$
  - $T(n) \le 0.7cn + 0.2cn + O(n)$
  - $= 0.9cn + O(n)$
  - $= O(n)$ <<< dominating rule

Using Recursion Tree

- $T(n) = 2\,T(n/2) + n$



$T(n) = O(n \lg n)$

Master Method

- $T(n) = aT(n/b) + f(n)\quad a \geq 1, b > 1$
- Let $c = \log_b(a)$
- $f(n) = O(n^{c-\varepsilon}) \rightarrow T(n) = \Theta(n^c)$
- $f(n) = \Theta(n^c) \rightarrow T(n) = \Theta(n^c \log n)$
  - $f(n) = \Theta(n^c \log^k n) \rightarrow T(n) = \Theta(n^c \log^{k+1} n)$
- $f(n) = \Omega(n^{c+\varepsilon}) \rightarrow T(n) = \Theta(f(n))$

Master Method : Example

- $T(n) = 9T(n/3) + n$
- $a = 9, b = 3, c = \log_3 9 = 2, n^c = n^2$
- $f(n) = n = O(n^{2-0.1})$
- $T(n) = \Theta(n^c) = \Theta(n^2)$

The case of $f(n) = O(n^{c-\varepsilon})$

Master Method : Example

- $T(n) = T(n/3) + 1$
- $a = 1, b = 3, c = \log_3 1 = 0, n^c = 1$
- $f(n) = 1 = \Theta(n^c) = \Theta(1)$
- $T(n) = \Theta(n^c \log n) = \Theta(\log n)$

The case of $f(n) = \Theta(n^c)$

Master Method : Example

- $T(n) = 3T(n/4) + n \log n$
- $a = 3, b = 4, c = \log_4 3 < 0.793, n^c < n^{0.793}$
- $f(n) = n \log n = \Omega(n^{0.793})$
- $a\,f(n/b) = 3\,((n/4) \log (n/4)) \leq (3/4)\, n \log n = d\, f(n)$
- $T(n) = \Theta(f(n)) = \Theta(n \log n)$

The case of $f(n) = \Omega(n^{c+\varepsilon})$

Conclusion

- Asymptotic Bound is, in fact, very simple
  - Use the rule of thumbs
    - Discard non dominant term
    - Discard constant
- For recursive
  - Make recurrent relation
    - Use master method
    - Guessing and proof
    - Recursion Tree