Overview

- Big-O notation
- Analysis cases
- Critical section
Big-O Notation

- Represents
  - Upper bound on number of steps in algorithm
  - Intrinsic efficiency of algorithm for large inputs

Formal Definition of Big-O

- Function $f(n)$ is $O(\ g(n)\ )$ if
  - For some positive constants $M$, $N_0$
  - $M \times g(N_0) \geq f(n)$, for all $n \geq N_0$

- Intuitively
  - For some coefficient $M$ & all data sizes $\geq N_0$
  - $M \times g(n)$ is always greater than $f(n)$
Big-O Examples

5n + 1000 ⇒ O(n)
- Select M = 6, N₀ = 1000
- For n ≥ 1000
  - 6n ≥ 5n+1000 is always true
- Example ⇒ for n = 1000
  - 6000 ≥ 5000 +1000

Big-O Examples

2n² + 10n + 1000 ⇒ O(n²)
- Select M = 4, N₀ = 100
- For n ≥ 100
  - 4n² ≥ 2n² + 10n + 1000 is always true
- Example ⇒ for n = 100
  - 40000 ≥ 20000 + 1000 + 1000
Types of Case Analysis

- Can analyze different types (cases) of algorithm behavior
- Types of analysis
  - Best case
  - Worst case
  - Average case

Types of Case Analysis

- Best case
  - Smallest number of steps required
  - Not very useful
  - Example: Find item in first place checked
Types of Case Analysis

- **Worst case**
  - Largest number of steps required
  - Useful for upper bound on worst performance
  - Real-time applications (e.g., multimedia)
  - Quality of service guarantee
  - Example ⇒ Find item in last place checked

Types of Case Analysis

- **Average case**
  - Number of steps required for “typical” case
  - Most useful metric in practice
  - Different approaches
    1. Average case
    2. Expected case
    3. Amortized
Approaches to Average Case

1. Average case
   - Average over all possible inputs
   - Assumes uniform probability distribution

2. Expected case
   - Weighted average over all inputs
   - Weighted by likelihood of input

3. Amortized
   - Examine common sequence of operations
   - Average number of steps over sequence

Quicksort Example

- Quicksort
  - One of the fastest comparison sorts
  - Frequently used in practice

- Quicksort algorithm
  - Pick pivot value from list
  - Partition list into values smaller & bigger than pivot
  - Recursively sort both lists
Quicksort Example

- Quicksort properties
  - Average case = $O(n\log(n))$
  - Worst case = $O(n^2)$
    - Pivot ≈ smallest / largest value in list
    - Picking from front of nearly sorted list
- Can avoid worst-case behavior
  - Attempt to select random pivot value

Amortization Example

- Adding numbers to end of array of size $k$
  - If array is full, allocate new array
    - Allocation cost is $O($size of new array$)$
    - Copy over contents of existing array
- Two approaches
  - Non-amortized
    - If array is full, allocate new array of size $k+1$
  - Amortized
    - If array is full, allocate new array of size $2k$
    - Compare their allocation cost
Amortization Example

- **Non-amortized approach**
  - Allocation cost as table grows from 1..n
  - Total cost \( \Rightarrow \frac{1}{2} (n+1)^2 \)
  - Case analysis
    - Best case \( \Rightarrow \) allocation cost = k
    - Worse case \( \Rightarrow \) allocation cost = k
    - Average case \( \Rightarrow \) allocation cost = k = n/2

- **Amortized approach**
  - Allocation cost as table grows from 1..n
  - Total cost \( \Rightarrow 2(n-1) \)
  - Case analysis
    - Best case \( \Rightarrow \) allocation cost = 0
    - Worse case \( \Rightarrow \) allocation cost = 2(k – 1)
    - Average case \( \Rightarrow \) allocation cost = 2
  - Worse case takes more steps, but faster overall
Analyzing Algorithms

Goal
- Find asymptotic complexity of algorithm

Approach
- Ignore less frequently executed parts of algorithm
- Find critical section of algorithm
- Determine how many times critical section is executed as function of problem size

Critical Section of Algorithm

Heart of algorithm
- Dominates overall execution time

Characteristics
- Operation central to functioning of program
- Contained inside deeply nested loops
- Executed as often as any other part of algorithm

Sources
- Loops
- Recursion
Critical Section Example 1

- Code (for input size $n$)
  1. A
  2. for (int $i = 0; i < n; i++$)
  3. B
  4. C

- Code execution
  - A $\Rightarrow$
  - B $\Rightarrow$
  - C $\Rightarrow$

- Time $\Rightarrow$

Critical Section Example 1

- Code (for input size $n$)
  1. A
  2. for (int $i = 0; i < n; i++$)
  3. B
  4. C

- Code execution
  - A $\Rightarrow$ once
  - B $\Rightarrow$ $n$ times
  - C $\Rightarrow$ once

- Time $\Rightarrow 1 + n + 1 = O(n)$
Critical Section Example 2

Code (for input size n)
1. A
2. for (int i = 0; i < n; i++)
3. B
4. for (int j = 0; j < n; j++)
5. C
6. D

Code execution
- A ⇒
- B ⇒
- C ⇒
- D ⇒

Time ⇒

Critical Section Example 2

Code (for input size n)
1. A
2. for (int i = 0; i < n; i++)
3. B
4. for (int j = 0; j < n; j++)
5. C
6. D

Code execution
- A ⇒ once
- B ⇒ n times
- C ⇒ n^2 times
- D ⇒ once

Time ⇒ 1 + n + n^2 + 1 = O(n^2)
Critical Section Example 3

Code (for input size n)

1. A
2. for (int i = 0; i < n; i++)
3. for (int j = i+1; j < n; j++)
4. B

Code execution

A ⇒
B ⇒
Time ⇒

Time ⇒ 1 + ½ n² = O(n²)
Critical Section Example 4

Code (for input size n)
1. A
2. for (int i = 0; i < n; i++)
3. for (int j = 0; j < 10000; j++)
4. B

Code execution
- A ⇒
- B ⇒
- Time ⇒

Time ⇒ 1 + 10000 n = O(n)
Critical Section Example 5

Code (for input size \( n \))
1. for (int \( i \) = 0; \( i < n \); \( i++ \))
2. for (int \( j \) = 0; \( j < n \); \( j++ \))
3. A
4. for (int \( i \) = 0; \( i < n \); \( i++ \))
5. for (int \( j \) = 0; \( j < n \); \( j++ \))
6. B

Code execution
- \( A \Rightarrow \)
- \( B \Rightarrow \)
- Time \( \Rightarrow \)

Critical Section Example 5

Code (for input size \( n \))
1. for (int \( i \) = 0; \( i < n \); \( i++ \))
2. for (int \( j \) = 0; \( j < n \); \( j++ \))
3. A
4. for (int \( i \) = 0; \( i < n \); \( i++ \))
5. for (int \( j \) = 0; \( j < n \); \( j++ \))
6. B

Code execution
- \( A \Rightarrow n^2 \) times
- \( B \Rightarrow n^2 \) times
- Time \( \Rightarrow n^2 + n^2 = O(n^2) \)
Critical Section Example 6

Code (for input size n)
1. \( i = 1 \)
2. \( \text{while} \ (i < n) \)
3. \( A \)
4. \( i = 2 \times i \)
5. \( B \)

Code execution
- \( A \rightarrow \)
- \( B \rightarrow \)
- \( \text{Time} \rightarrow \)

\[ \text{Time} \rightarrow \log(n) + 1 = O(\log(n)) \]
Critical Section Example 7

- **Code (for input size n)**
  1. DoWork (int n)
  2. if (n == 1)
  3.   A
  4. else
  5.   DoWork(n/2)
  6.   DoWork(n/2)

- **Code execution**
  - A ➞
  - DoWork(n/2) ➞
  - Time(1) ➞ Time(n) =

Critical Section Example 7

- **Code (for input size n)**
  1. DoWork (int n)
  2. if (n == 1)
  3.   A
  4. else
  5.   DoWork(n/2)
  6.   DoWork(n/2)

- **Code execution**
  - A ➞ 1 times
  - DoWork(n/2) ➞ 2 times
  - Time(1) ➞ 1  Time(n) = 2 × Time(n/2) + 1