Data Structure

Lecture#6: Algorithm Analysis 2 (Chapter 3)

U Kang
Seoul National University

In This Lecture

- Learn the examples of asymptotic analysis
- Learn the concepts of space complexity, and time/space tradeoff
- Learn how to analyze algorithms with multiple parameters

Asymptotic Analysis: Big-oh

- Definition: For $\mathbf{T}(n)$ a non-negatively valued function, $\mathbf{T}(n)$ is in the set $O(f(n))$ if there exist two positive constants c and n_0 such that $\mathbf{T}(n) <= cf(n)$ for all $n > n_0$.
- Use: The algorithm is in $O(n^2)$ in [best, average, worst] case.
- Meaning: For all data sets big enough (i.e., $n>n_0$), the algorithm always executes in less than $cf(n)$ steps in [best, average, worst] case.

Big-Omega

- Definition: For $\mathbf{T}(n)$ a non-negatively valued function, $\mathbf{T}(n)$ is in the set $\Omega(g(n))$ if there exist two positive constants c and n_0 such that $\mathbf{T}(n) >= cg(n)$ for all $n > n_0$.

- Meaning: For all data sets big enough (i.e., $n > n_0$), the algorithm always requires more than $cg(n)$ steps.

- Lower bound.

Theta Notation

- When big-Oh and Ω coincide, we indicate this by using Θ (big-Theta) notation.

- Definition: An algorithm is said to be in $\Theta(h(n))$ if it is in $O(h(n))$ and it is in $\Omega(h(n))$.

Simplifying Rules

1. If $f(n)$ is in $O(g(n))$ and $g(n)$ is in $O(h(n))$, then $f(n)$ is in $O(h(n))$.
2. If $f(n)$ is in $O(kg(n))$ for some constant $k > 0$, then $f(n)$ is in $O(g(n))$.
3. If $f_1(n)$ is in $O(g_1(n))$ and $f_2(n)$ is in $O(g_2(n))$, then $(f_1 + f_2)(n)$ is in $O(\max(g_1(n), g_2(n)))$.
   - What about Ω ? What about Θ ?
4. If $f_1(n)$ is in $O(g_1(n))$ and $f_2(n)$ is in $O(g_2(n))$ then $f_1(n)f_2(n)$ is in $O(g_1(n)g_2(n))$.

Time Complexity Examples (1)

- Example 3.9: **`a = b;`**

This assignment takes constant time, so it is $\Theta(1)$.

- Example 3.10:

```
sum = 0;
for (i=1; i<=n; i++)
   sum += n;
```

Time Complexity Examples (2)

- Example 3.11:

```
sum = 0;
for (j=1; j<=n; j++)
  for (i=1; i<=j; i++)
    sum++;
for (k=0; k<n; k++)
  A[k] = k;
```

Time Complexity Examples (3)

- Example 3.12:

```
sum1 = 0;
for (i=1; i<=n; i++)
  for (j=1; j<=n; j++)
    sum1++;

sum2 = 0;
for (i=1; i<=n; i++)
  for (j=1; j<=i; j++)
    sum2++;
```

Time Complexity Examples (4)

- Example 3.13:

```
sum1 = 0;
for (k=1; k<=n; k*=2)
  for (j=1; j<=n; j++)
    sum1++;

sum2 = 0;
for (k=1; k<=n; k*=2)
  for (j=1; j<=k; j++)
    sum2++;
```

Binary Search

Looking for '45'

Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Key	11	13	21	26	29	36	40	41	45	51	54	56	65	72	77	83

- How many elements are examined in worst case?

Binary Search

```
/** @return The position of an element in
    sorted array A with value k.  If k is not
    in A,return A.length. */
static int binary(int[] A, int k) {
  int l = -1;           // Set l and r
  int r = A.length;   // beyond array bounds
  while (l+1 != r) { // Stop when l, r meet
    int i = (l+r)/2; // Check middle
    if (k < A[i]) r = i;      // In left half
    if (k == A[i]) return i; // Found it
    if (k > A[i]) l = i;      // In right half
  }
  return A.length; // Search value not in A
}
```

Other Control Statements

- **while** loop: analyze like a **for** loop.
- **if** statement: take greater complexity of **then/else** clauses.
- **switch** statement: take complexity of most expensive case.
- Subroutine call: complexity of the subroutine.

Discussion

- Is it always ok to ignore the constants as in the asymptotic analysis?
  - Alg1: $T(n) = 1000n$
  - Alg2: $T(n) = 2n^2$

Problems

- Problem: a task to be performed.
  - Best thought of as inputs and matching outputs.
  - Problem definition should include constraints on the resources that may be consumed by any acceptable solution.

Problems (cont)

- Problems ⇔ mathematical functions
  - A <u>function</u> is a matching between inputs (the <u>domain</u>) and outputs (the <u>range</u>).
  - An <u>input</u> to a function may be single number, or a collection of information.
  - The values making up an input are called the <u>parameters</u> of the function.
  - A particular input must always result in the same output every time the function is computed.
    - Exception: randomized algorithm

Algorithms and Programs

- Algorithm: a method or a process followed to solve a problem.
  - A recipe.

- An algorithm takes the input to a problem (function) and transforms it to the output.
  - A mapping of input to output.

- A problem can have many algorithms.

Analyzing Problems: Example

- May or may not be able to obtain matching upper and lower bounds.

- Example of imperfect knowledge: Sorting

  1. Cost of I/O: $\Omega(n)$.
  2. Bubble or insertion sort: $O(n^2)$.
  3. A better sort (Quicksort, Mergesort, Heapsort, etc.): $O(n \log n)$.
  4. We prove later that sorting is in $\Omega(n \log n)$.

Space/Time Tradeoff Principle

- One can often reduce time if one is willing to sacrifice space, or vice versa.
  - Factorial: how can we make fact(n) super fast?
  - Swapping a and b: how can we do this without additional space?

- Disk-based Space/Time Tradeoff Principle: The smaller you make the disk storage requirements, the faster your program will run.

Multiple Parameters

- Compute the rank ordering for all C pixel values in a picture of P pixels.

```
for (i=0; i<C; i++)  // Initialize count
  count[i] = 0;
for (i=0; i<P; i++)  // Look at all pixels
  count[value(i)]++; // Increment count
sort(count);         // Sort pixel counts
```

- Running time?
  - $\Theta(P + C \log C)$

Space Complexity

- Space complexity can also be analyzed with asymptotic complexity analysis.
  - Amount of memory space to keep while running an algorithm

```
for (i=0; i<C; i++)  // Initialize count
  count[i] = 0;
for (i=0; i<P; i++)  // Look at all pixels
  count[value(i)]++; // Increment count
sort(count);         // Sort pixel counts
```

- Space complexity of the above code?

(input: value[0..P])

What you need to know

- Asymptotic analysis
  - Analyze the time and space complexity of an algorithm
- Time/space tradeoff
  - Understand the main idea
  - Convert an operation to trade time for space (or vice versa)
- How to analyze algorithms with multiple parameters

Questions?