DS8001: Designs of Algorithms and Programming for Massive Data

Lecture 2

A. Miranskyy

September 19, 2016

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

Useful definitions

T The running time of an algorithm

c_i Constant amount of time required to execute i-th line of a pseudocode

Useful definitions

Informal

Definition

- Best-case analysis: running time of the algorithm in case the input is 'optimal'; provides lower bound of T.
- Average-case analysis: running time of the algorithm in case the input is 'average'.
- Worst-case analysis: running time of the algorithm in case the input is 'average'; provides upper bound of T.

It is often difficult to define average- or best- case; thus, worst-case analysis is the most common one.

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

Number of operations
Example 1: simple for-loop

Line	Pseudo code	Cost	Times
1	$k = 0$	c_1	1
2	`for` $i = 1$ `to` n	c_2	$n+1$
3	$\quad k = k + i$	c_3	n

- $T(n) = c_1 + c_2(n+1) + c_3 n = (c_2 + c_3)n + (c_1 + c_2)$.
  - Number of operations grows linearly with n.
- For a given n, best-, average-, and worst-cases are the same.

Question:

Why second line is executed $n + 1$ times?

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

Line	Pseudo code	Cost	Times
1	$k = 0$	c_1	1
2	for $i = 1$ to n	c_2	$n + 1$
3	for $j = 1$ to n	c_3	$n(n + 1)$
4	$k = k + i + j$	c_4	n^2

- $T(n) = c_1 + c_2(n + 1) + c_3 n(n + 1) + c_4 n^2 = (c_3 + c_4)n^2 + (c_2 + c_3)n + (c_1 + c_2)$.
- For a given n, best-, average-, and worst-cases are the same.

Number of operations

Example 3: nested for-loop with two 'control' variables

Line	Pseudo code	Cost	Times
1	$k = 0$	c_1	1
2	`for` $i = 1$ `to` n	c_2	$n+1$
3	`for` $j = 1$ `to` m	c_3	$n(m+1)$
4	$k = k + i + j$	c_4	nm

- $T(n, m) = c_1 + c_2(n+1) + c_3 n(m+1) + c_4 nm = (c_3 + c_4)nm + (c_2 + c_3)n + (c_1 + c_2)$.
- For a given n and m, best-, average-, and worst-cases are the same.

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

- Input: A sequence of numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$.
- Output: A reordering (permutation) $\langle a_1^{'}, a_2^{'}, \ldots, a_n^{'} \rangle$ of the input sequence, such that $a_1^{'} \leq a_2^{'} \leq \ldots \leq a_n^{'}$.
- a_is are also known as keys.

Number of operations

Example 4: Insertion sort

Figure: Graphic example of insertion sort

Detailed coverage of this example is given in [1, Ch. 2.2].

Number of operations

Example 4: Insertion sort

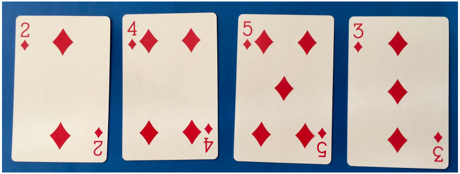

Figure: Step 1

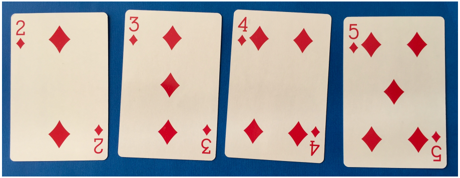

Figure: Step 3

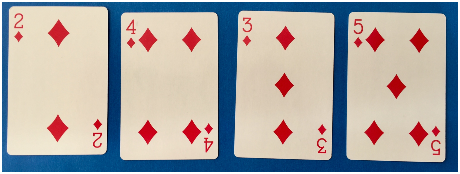

Figure: Step 2

Note that in practice we do not have to physically place new card into the deck until we find its proper spot.

Number of operations

Example 4: Insertion sort

Line	Pseudo code	Cost	Times
1	$\texttt{for}\ j = 2\ \texttt{to}\ n$	c_1	n
2	$\quad key = A[j]$	c_2	$n-1$
3	$\quad$ // Insert $A[j]$ into $A[1, \ldots, j-1]$	0	$n-1$
4	$\quad i = j - 1$	c_4	$n-1$
5	$\quad \texttt{while}\ i > 0\ \texttt{and}\ A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$\qquad A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$\qquad i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$\quad A[i+1] = key$	c_8	$n-1$

- t_j is the number of times a `while` loop on line 5 is executed for a given j.
- $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$.

Number of operations

Example 4: Insertion sort : Best Case

Line	Pseudo code	Cost	Times
1	`for` $j = 2$ `to` n	c_1	n
2	$\quad key = A[j]$	c_2	$n-1$
3	$\quad$ // Insert $A[j]$ into $A[1,\ldots,j-1]$	0	$n-1$
4	$\quad i = j - 1$	c_4	$n-1$
5	$\quad$ `while` $i > 0$ `and` $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$\quad\quad A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$\quad\quad i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$\quad A[i+1] = key$	c_8	$n-1$

- Best-case scenario: the sequence A is already sorted in ascending order. In this case $t_j = 1$. And $T(n)$ simplifies to:
- $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$.
- $T(n)$ is a linear function.

- Worst-case scenario: the sequence A is sorted in descending order. In this case $t_j = j$, since we need to compare $A[j]$ with each element in $A[1, \dots, j-1]$.
- Knowing that $\sum_{j=2}^{n} j = n(n+1)/2 - 1$ and that $\sum_{j=2}^{n}(j-1) = n(n-1)/2$ (remember your first year calculus?), $T(n)$ simplifies to
- $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} j + c_6 \sum_{j=2}^{n}(j-1) + c_7 \sum_{j=2}^{n}(j-1) + c_8(n-1) = (c_5/2 + c_6/2 + c_7/2)n^2 + (c_1 + c_2 + c_4 + c_5/2 - c_6/2 - c_7/2 + c_8)n - (c_2 + c_4 + c_5 + c_8)$.
- $T(n)$ is a quadratic function.

- Average-case scenario: suppose that we draw n numbers from uniform distribution. Then, on average, half the elements in $A[1,\ldots,j-1]$ are less than $A[j]$ and half are greater than $A[j]$. Thus, on average, $t_j = j/2$.
- I will leave it for you to compute. Hint: $\sum_{j=2}^{n} j/2 = n(n+1)/4 - 1$ and $\sum_{j=2}^{n} (j/2-1) = n(n-3)/4 + 1/2$
- $T(n)$ will be messier, but it will still be a quadratic function (similar to the worst case).

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

- Asymptotic notation (Bachmann-Landau notation) was invented by Paul Bachmann [2] (1894) and Edmund Landau [3] (1909).
- Describes limiting behaviour of a function when function's argument(s) tend toward particular value (e.g., infinity).
- We will use it to simplify $T(n)$.

Asymptotic notation

Pronunciation

$\Theta(g(n))$ big-theta of g of n (sometimes theta of g of n)
$O(g(n))$ big-oh of g of n (sometimes oh of g of n)
$o(g(n))$ little-oh of g of n
$\Omega(g(n))$ big-omega of g of n (sometimes omega of g of n)
$\omega(g(n))$ little-omega of g of n

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

Θ-notation

- For a given function $g(n)$ we denote a set of functions $\Theta(g(n))$ such that
  - $\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$.
- $f(n)$ belongs to $\Theta(g(n))$ if it can be "sandwitched" between $c_1 g(n)$ and $c_2 g(n)$ for all $n \geq n_0$.

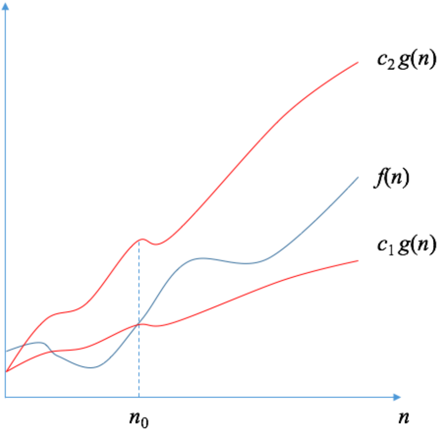

Figure: Graphic example of Θ-notation

Θ-notation

- $f(n)$ belongs to $\Theta(g(n))$ if it can be "sandwitched" between $c_1 g(n)$ and $c_2 g(n)$ for all $n \geq n_0$.
- That is $f(n)$ is equal to $g(n)$ within a constant factor.
- The fancy term for this is that $g(n)$ is an asymptotically tight bound for $f(n)$.

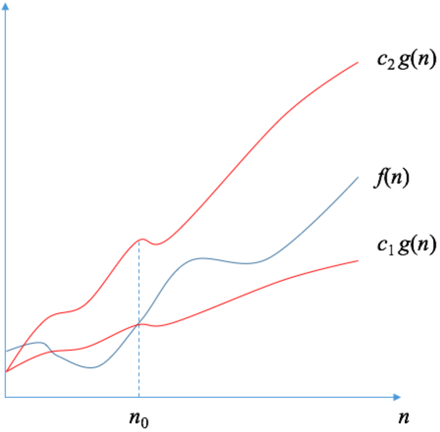

Figure: Graphic example of Θ-notation

Θ-notation

- We can say that $f(n) \in \Theta(g(n))$, to indicate that $f(n)$ is a member of $\Theta(g(n))$. Instead, we usually write $f(n) = \Theta(g(n))$
- Note that the definition of $\Theta(g(n))$ requires that every member of the, i.e. $f(n)$ and $g(n)$ be asymptotically non-negative for sufficiently large n. This assumption holds for other notations discusses below.

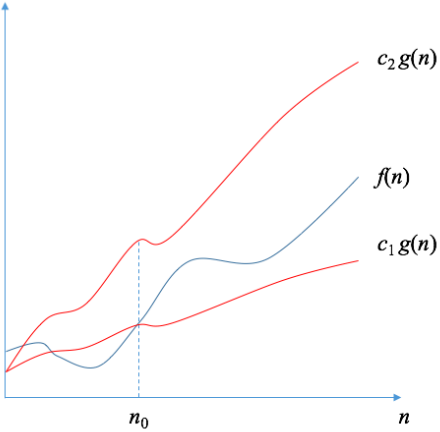

Figure: Graphic example of Θ-notation

- $\Theta(1)$ – constant ($n^0 = 1$)
- $\Theta(\log(n))$ – logarithmic
- $\Theta(n)$ – linear
- $\Theta(n^2)$ – quadratic
- $\Theta(c^n)$ – exponential ($c > 1$)
- $\Theta(n!)$ – factorial

Note that $\cdot$ in $\Theta(\cdot)$ is used in other notations.

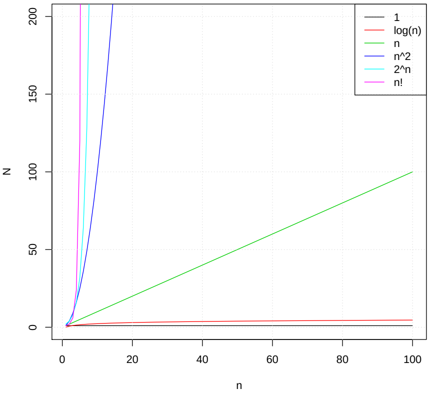

Figure: Number of operations N vs. input size n.

Θ-notation
Example

Show that $n^2/3 - 3n = \Theta(n^2)$. We need to compute positive constants c_1, c_2, n_0 (if they exist), such that

$$c_1 n^2 \leq n^2/3 - 3n \leq c_2 n^2, \forall n \geq n_0.$$

Note that we have undetermined system of equations: three unknowns and two equations – will need to "pinpoint" one of the constants. Dividing by n^2 yields

$$c_1 \leq 1/3 - 3/n \leq c_2.$$

Let's pick $n_0 = 10$, then

$$c_1 \leq 1/30 \leq c_2.$$

E.g., we can set c_1 to $1/31$ and c_2 to $1/29$ for the inequalities to hold. Obviously, other choices of the constants exist.

- Let us look at complexity of a simple for-loop discussed in slide 6:
  - $T(n) = (c_2 + c_3)n + (c_1 + c_2) = \Theta(n)$.
- And for nested loop (discussed in slide 8):
  - $T(n) = (c_3 + c_4)n^2 + (c_2 + c_3)n + (c_1 + c_2) = \Theta(n^2)$.
- If multiple variables are involved (as in the nested for-loop, controlled by m and n, as shown in slide 9), we have to be careful to understand the growth rate of each variable involved in the equation. In the simplest case, as $m, n \to \infty$:
  - $T(n, m) = (c_3 + c_4)nm + (c_2 + c_3)n + (c_1 + c_2) \overset{m,n\to\infty}{=} \Theta(nm)$.

- Finally, for the sorting example given in slides 12-onward:
  - Best case:
    $T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n + (c_2 + c_4 + c_5 + c_8) = \Theta(n)$.
  - Average case:
    $T(n) = \underbrace{\ldots}_{\text{And here goes the formula that you computed on slide 17}} = \Theta(n^2)$.
  - Worst case: $T(n) = (c_5/2 + c_6/2 + c_7/2)n^2 + (c_1 + c_2 + c_4 + c_5/2 - c_6/2 - c_7/2 + c_8)n - (c_2 + c_4 + c_5 + c_8) = \Theta(n^2)$.

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

O-notation

- For a given function $g(n)$ we denote a set of functions $O(g(n))$ such that
  - $O(g(n)) = \{f(n) : \exists c, n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$.
- O-notation gives upper bound of the function, while Θ-notation gives both upper and lower bound.
- O stands for 'Ordnung' in [2, 3] (German for order).

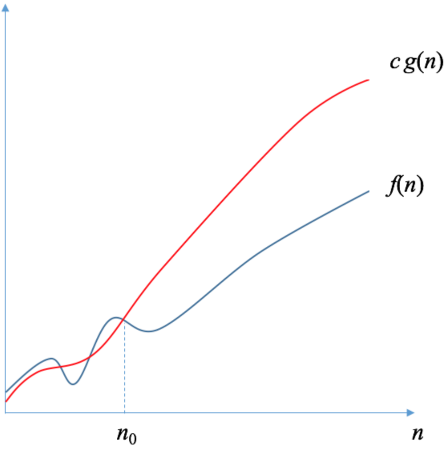

Figure: Graphic example of O-notation

- Note that $f(n) = \Theta(g(n))$ implies that $f(n) = O(g(n))$, i.e., $\Theta(g(n)) \supseteq O(g(n))$.
- Often, O-notation provide an upper bound, without claiming that it it asymptotically tight [1]. For example, we can say that $n = O(n^2)$ (or even that $n = O(n^{100})$) even though $O(n)$ would have been a tighter bound.

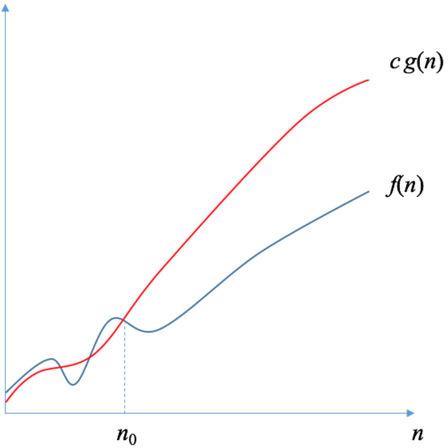

Figure: Graphic example of O-notation

O-notation

- Remember our insertion sort (slides 12-onward), where best-case scenario was $\Theta(n)$ and worst-case was $\Theta(n^2)$?
- Since, $\Theta(g(n)) \supseteq O(g(n))$, $O(n^2)$ gives upper bound on the worst-case insertion sort.
- Colloquially, we say that "running time of insertion sort is $O(n^2)$". We are abusing the terminology here, as run time order will vary with input for a given n. What we typically mean by this statement is that the worst-case run time is $O(n^2)$.

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

- For a given function $g(n)$ we denote a set of functions $\Omega(g(n))$ such that
  - $\Omega(g(n)) = \{f(n) : \exists c, n_0 > 0$ such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$.
- Ω-notation gives asymptotic lower bound of the function.

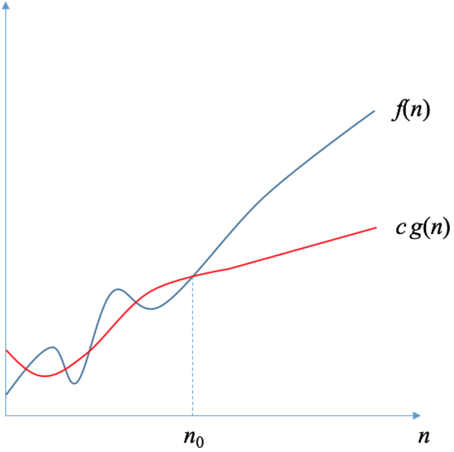

Figure: Graphic example of Ω-notation

Ω-notation

Theorem

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Ω-notation

- We say that running time of an algorithm is $\Omega(g(n))$ meaning that running time is at least $g(n)$ units of time (multiplied by some constant) for sufficiently large n.
- Essentially, we are looking at the best-case scenario. In the case of our insertion sort best case was bounded by $\Theta(n)$, thus lower bound for insertion sort $\Omega(n)$.
- However, we can say that the worst-case running time for the insertion sort is $\Omega(n^2)$, since there does exist an input that will cause the algorithm to use $\Omega(n^2)$ time.

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

o-notation

- For a given function $g(n)$ we denote a set of functions $o(g(n))$ such that
  - $o(g(n)) = \{f(n) :$ for any positive constant $c > 0$, there exists $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$.
- Upper bound provided by O-notation may or may not be asymptotically tight. However, o-notation denotes upper bound that is not asymptotically tight.
- We can say that $2n = O(n)$ and $2n = O(n^2)$.
  - The former is asymptotically tight, the latter is not.
- However, $2n \neq o(n)$ and $2n = o(n^2)$.

Question:

Why $2n \neq o(n)$?

o-notation

- Given that $f(n) < g(n)$ (for $n > n_0$) we can say that

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

- This ratio is often used as a definition of o-notation.

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

ω-notation

- For a given function $g(n)$ we denote a set of functions $o(g(n))$ such that
  - $o(g(n)) = \{f(n)$: for any positive constant $c > 0$, there exists $n_0 > 0$ such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$.
- Lower bound provided by Ω-notation may or may not be asymptotically tight. However, ω-notation denotes lower bound that is not asymptotically tight.
- We can say that $2n^2 = \Omega(n^2)$ and $2n^2 = \Omega(n)$.
  - The former is asymptotically tight, the latter is not.
- However, $2n^2 \neq \omega(n^2)$ and $2n^2 = \omega(n)$.

Question:

Why $2n^2 \neq \omega(n^2)$?

ω-notation

- Given that $f(n) > g(n)$ (for $n > n_0$) we can say that

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

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

Consider example from slide 9

Pseudo code

```
k = 0
for i = 1 to n
   for j = 1 to m
      k = k + i + j
```

- We have scalar variables k, i, j, n, m, each one consuming fixed amount of spaces.
- Thus, memory consumption will look like
- $M(n) = \underbrace{c_1}_{k} + \underbrace{c_2}_{i} + \underbrace{c_3}_{j} + \underbrace{c_4}_{n} + \underbrace{c_5}_{m} = \Theta(1)$

Consider example from slide 12

Pseudo code
for $j = 2$ to n
$key = A[j]$
// Insert $A[j]$ into $A[1, \ldots, j-1]$
$i = j - 1$
while $i > 0$ and $A[i] > key$
$A[i+1] = A[i]$
$i = i - 1$
$A[i+1] = key$

Amount of memory II

Example 4: insertion sort

- We have scalar variables i, j, n, key each one consuming fixed amount of spaces and a data structure storing vector A (can be implemented using array, linked list, etc.).
- Thus, memory consumption will look like
- $M(n) = \underbrace{c_1}_{i} + \underbrace{c_2}_{j} + \underbrace{c_3}_{n} + \underbrace{c_4}_{key} + \underbrace{c_5 n}_{A} = \Theta(n)$

Outline

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

Asymptotic notation and small n

- For small values of n with modern hardware the difference between algorithms is often negligible [4]. E.g., consider linear search, where in the worst-case scenario you have to scan through the whole array to find an element of interest.
- Depending on the implementation details, we can scan 700,000 and 45,000,000 elements (and that's without parallelism) in less than 200ms [4]. That's between 3MB and 172MB of data (assuming we are searching in array of 32-bit integers).
  - 200ms is perceived as instantaneous by human brain [4].

CPU layout

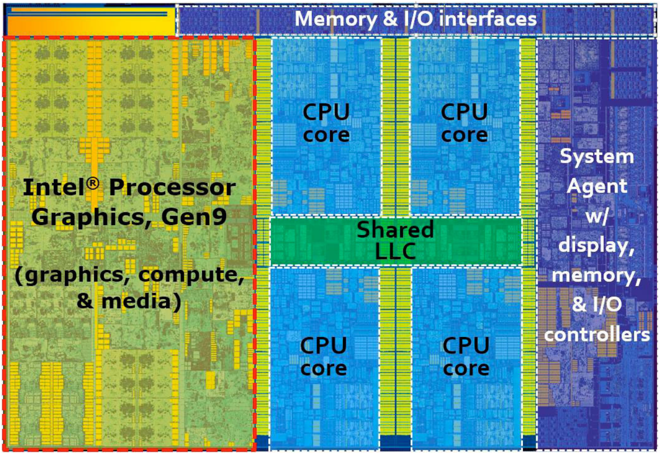

Figure: Intel Skylake die (taken from [5]).

LLC stands for Last Level Cache, (typically L3).

CPU Cache

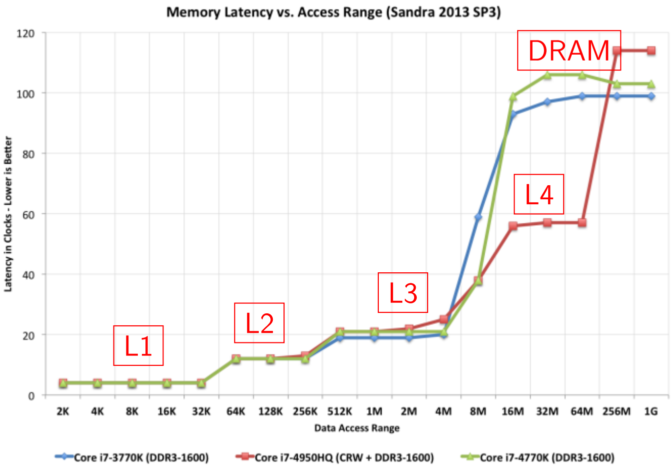

Figure: Memory latency (taken from [6]).

Coefficients matter! I

When it comes to implementation...

- Modern CPUs can do a lot of computatations per second. 3GHz CPU does 3×10^9 cycles per second per core.
  - For simplicity, let's ignore number of instructions per cycle, number of instructions per second, etc.
- CPU manufacturers are working very hard to move the data into cache proactively, so that CPUs do not have to idle [7].

Coefficients matter! II

When it comes to implementation...

- Consider two algorithms: one $O(n)$ another $O(n\log(n))$. For small n $O(n) > O(n\log(n))$ and for large n it's the opposite. Let us quantify the terms 'small' and 'large'.
- This is where the cost of operations c_i becomes important.
- Assume that the $O(n)$ algorithm designed or implemented in such a way that it leads to frequent L3 cache misses leading to waste of ≈ 40 CPU cycles per every operation [7]. Also assume that $O(n\log(n))$ algorithm has no cache misses. Then:
  - $40n > n\log(n) \Rightarrow n < 2^{40} = 1$ trillion.
    - We assume that $\log(n)$ is the binary logarithm $\log_2(n)$.
  - That is $O(n)$ will prevail only when $n > 2^{40}$.
- And what happens if we constantly have to go to DRAM? Then it's 100 to 200 cycles [7], and 2^{100} is a very big number.
  - Far greater than any massive data we are dealing with now...

Coefficients matter! III

When it comes to implementation...

- Of course, if the workload is I/O-bound, then cache misses become the least of your problems, but in many cases we can load significant chunks of data into memory for faster processing.

Summary

1. Analyzing algorithms: time complexity
   - Simple for-loop
   - Simple nested for-loops
   - Simple sorting problem
2. Asymptotic notation
   - Overview
   - Θ-notation
   - O-notation
   - Ω-notation
   - o-notation
   - ω-notation
3. Analyzing algorithms: space complexity
   - Examples
4. Asymptotic notation and practice
   - Examples

References I

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT press, 2009.

[2] P. Bachmann, *Die analytische zahlentheorie*. Teubner, 1894, vol. 2.

[3] E. Landau, "Handbuch der lehre vonder verteilung der primzahlen," *Bd. I, II, Leipzig, Berlin: Teubner*, 1909.

[4] T. A. Limoncelli, "10 optimizations on linear search," *Queue*, vol. 14, no. 4, pp. 10:20–10:33, Aug. 2016. [Online]. Available: http://doi.acm.org/10.1145/2984629.2984631

[5] btarunr, "Techpowerup: Intel "Skylake" Die Layout Detailed," 2015. [Online]. Available: https://www.techpowerup.com/215333/intel-skylake-die-layout-detailed

[6] A. L. Shimpi, "Crystalwell: Addressing the Memory Bandwidth Problem - Intel Iris Pro 5200 Graphics Review: Core i7-4950HQ Tested," 2013. [Online]. Available: http://www.anandtech.com/show/6993/intel-iris-pro-5200-graphics-review-core-i74950hq-tested/3

References II

[7] D. Levinthal, "Performance Analysis Guide for Intel Core i7 Processor and Intel Xeon 5500 processors ," 2009. [Online]. Available: https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf