CS 120: Intro to Algorithms and their Limitations

Problem Set 1

Due: September 20, 2023 11:59pm **Denny Cao**

---

§1 Asymptotic Notation

Answer 1.a.

f	g	O	o	Ω	ω	Θ
e^{n^2}	e^{2n^2}	T	T	F	F	F
n^3	$n^{3/n}$	F	F	T	T	F
$n^{2+(-1)^n}$	$\binom{n}{2}$	F	F	F	F	F
$(\log n)^{120}\sqrt{n}$	n	T	T	F	F	F
$\log(e^{n^2})$	$\log(e^{2n^2})$	T	F	T	F	T

Answer 1.b.

By definition of Big Θ, we know that $f(n) = \Theta(a^n) \to f(n) = O(a^n) \wedge f(n) = \Omega(a^n)$. By definition of Big O and Big Ω, $f(n) = O(a^n) \to \exists c_1 > 0 \mid f(n) \leq c_1 a^n$ for all n greater than some k and $f(n) = \Omega(a^n) \to \exists c_2 > 0 \mid f(n) \geq c_2 a^n$ for all n greater than some k. We combine into the following inequality:

$$c_1 a^n \leq f(n) \leq c_2 a^n$$

Similarly, $g(n) = \Theta(n^b) \to g(n) = O(n^b) \wedge g(n) = \Omega(n^b)$. Thus, $g(n) = O(n^b) \to \exists c_3 > 0 \mid g(n) \leq c_3 n^b$ for all n greater than some k and $g(n) = \Omega(n^b) \to \exists c_4 > 0 \mid g(n) \geq c_4 n^b$ for all n greater than some k. We combine into the following inequality:

$$c_3 n^b \leq g(n) \leq c_4 n^b$$

We will use these statements to prove the following:

- **$f(g(n)) = \Theta(a^{(n^b)})$ is false.**

  Proof by counterexample. Let $f(n) = 2^n$ and $g(n) = 10n^2$. We will show that $f(n) = \Theta(2^n)$. Let $c_1 = 1, c_2 = 2$. Then, the following statement is true:

  $$2^n \leq f(n) \leq 2(2)^n$$

  Thus, $f(n) = \Theta(2^n)$. We will show that $g(n) = \Theta(x^2)$. Let $c_3 = 1, c_4 = 10$. Then, the following statement is true:

  $$n^2 \leq g(n) \leq 10n^2$$

  $f(g(n)) = 2^{10n^2} = (2^{10})^{n^2}$. Thus, $f(g(n)) \neq \Theta(2^{n^2})$, as there does not exist a k_1 and k_2 such that:

  $$f(g(n)) \leq k_2(2^{n^2})$$

  Thus, $f(g(n)) \neq O(2^{n^2})$, which implies that $f(g(n)) \neq \Theta(2^{n^2})$. □

- $g(f(n)) = \Theta((a^n)^b)$ **is true.**

  Proof. Since for the composite $g(f(n))$, the least value of the input for g is the least value output for f and the greatest value input for g is the greatest value output for f (Both functions are increasing), it is the case that:

  $$c_3(c_1a^n)^b \leq g(f(n)) \leq c_4(c_1a^n)^b$$
  $$c_3c_1^b(a^n)^b \leq g(f(n)) \leq c_4c_1^b(a^n)^b$$

  Let $r_1 = c_3c_1^b$ and $r_2 = c_4c_1^b$. Then:

  $$r_1(a^n)^b \leq g(f(n)) \leq r_2(a^n)^b$$

  As $\exists r_2 > 0 \mid g(f(n)) \leq r_2(a^n)^b$, by definition of Big O, $g(f(n)) = O((a^n)^b)$. As $\exists r_1 > 0 \mid g(f(n)) \geq r_1(a^n)^b$, by definition of Big Ω, $g(f(n)) = \Omega((a^n)^b)$. Thus, as $g(f(n)) = O((a^n)^b) \wedge g(f(n)) = \Omega((a^n)^b)$, by definition of Big Θ, $g(f(n)) = \Theta((a^n)^b)$. □

§2 Understanding Computational Problems and Mathematical Notation

Answer 2.a. If the input is $(11, 10, 4)$, then the output will be $(1, 1, 0, 0)$. `BC`'s output is a valid solution for Π with input $(11, 10, 4)$, as the output satisfies $f(11, 10, 4)$, as $1 + 1(10) + 0(100) + 0(1000) = 11 = n$.

Answer 2.b.

Answer 2.c. When, for an $x = (n, b, k)$, $b < 2, b \in \mathbb{N}$ $(b = 1)$, or when k is less than the amount of digits of n expressed in base b.

Answer 2.d. $|f(x)| = 1$, as there are not multiple representations of a number in base b ; every number has a unique representation in base b.

Answer 2.e. No, not every algorithm A that solves Π also solves Π'.

Proof by counterexample. We will show that there exists an algorithm A that solves Π but does not solve Π. By definition of an algorithm, an algorithm A solves a computational problem Π if $\forall x \in \mathcal{I}(f(x) \neq \emptyset \rightarrow A(x) \in f(x))$. `BC` solves Π. We will show that `BC` does not solve Π'. Let $x = (11, 1, 10)$. In line 2 of `BC`, `if b < 2 then return` $\perp$. As $b = 1$, `BC` will return $\perp$, and thus $f(x) = \emptyset$. However, for the same input, $f'(x) = \emptyset \cup \{(0, 1, \ldots, 9)\} = \{0, 1, \ldots, 9\}$. As `BC` will return $\perp$ and not $\{0, 1, \ldots, 9\}$, there exists an algorithm that solves Π that does not solve Π'. □

§3 Radix Sort

Answer 3.a. (Proving Correctness of Algorithms)

Proof by Induction. We will prove the correctness of Radix Sort by induction on the number of iterations of the inner loop in the algorithm. Let $P(n)$ be the statement that, at every step n, the subintegers of the elements within the input array A become correctly ordered based on their corresponding digits, proceeding from the least significant digit to the most significant digit.

Base Case: $n = 0$. The 0th digit of the elements in A are sorted.

Inductive Hypothesis: Assume that after i iterations of the inner loop (sorting based on the i-th least significant digit), the subintegers of the elements in the array A are correctly ordered based on their first i digits (from right to left). We will show that $P(i) \rightarrow P(i+1)$.

Inductive Step: After the $i + 1$th iteration, the elements will be sorted based on the ith digit from the right. As we use `CountingSort` as a subroutine, and `CountingSort` is stable, the order will be preserved. As $P(i)$ is true, the elements of A are correctly sorted on their first i digits (from right to left). Thus, after the $i + 1$th iteration, if two numbers differ at the ith digit from the right, then they are placed in correct order and if they have the same value, then their position remains unchanged since `CountingSort` preserves order. Thus, the elements of A will become correctly sorted based on their first $i + 1$ digits (from right to left).

Thus, with induction, we have shown that Radix Sort correctly solves the Sorting Problem. □

Answer 3.b. (Analyzing Runtime)

Proof. `CountingSort` takes $O(n + U)$ time when keys are drawn from a universe of size U. In `RadixSort`, `CountingSort` is used as a subroutine on a smaller universe, b. `CountingSort` is ran for each digit. Each key will have $\lceil \log U / \log b \rceil$ digits in base b, as $\lceil \log U / \log b \rceil$ will give the greatest power of b that can be used to represent the largest value in the universe, U. The power will be the greatest amount of digits amongst all keys. As `CountingSort` is ran for each digit, there is a total running time of $(n+b)\lceil \log U / \log b \rceil$.

The value of b changes to minimize the expression depending on the values of n and U. If $b = \min\{n, U\}$, then the expression will be: $O((n + n)(\log U / \log n)) = O(n \log U / \log n)$. □

Answer 3.c. (Implementing Algorithms) In `ps1.py`.

Answer 3.d. (Experimentally Evaluating Algorithms) The shapes of the resulting transition curves fit what asymptotic theory suggests. Around $2^5 = U$, there is a transition from `CountingSort` being more efficient to `MergeSort` being more efficient; asymptotic theory suggests that, with small values of U, `CountingSort` is more efficient and with larger values, `MergeSort` will be more efficient. Around the point where $U = n^{O(1)}$, `RadixSort` is most efficient, as it achieves runtime $O(n)$ at that point.