COMS W3261, Section 1, Spring 2024
Streaming Algorithms

Instructor : Josh Alman
Notes by : William Pires

1 Big-O Notation

When measuring the running some resource usage (such as space or time) of an algorithm, the real answer is often a complicated function. It could be that your algorithm runs in time

$$5n^3 + 7n^2 - 4n + 2\log_{10}(n) - 3\log_2(\log_2(n)) + 100$$

on inputs of length n. We don't care so much about this, indeed as $n \to \infty$ only the leading term ($5n^3$) matters. Also, we don't care so much about constants, they often depend on the model of computation you consider. In this case we *ignore lower order terms, and constants* and say the algorithm runs in time $O(n^3)$.

Definition 1. Given two functions $f, g : \mathbb{N} \to \mathbb{R}^+$, we say that $f(n) = O(g(n))$ if there exists constants $c > 0$, n_0 such that :

$$\forall n \geq n_0 \ : \ f(n) < c * g(n)$$

This means that once n is large enough, $f(n)$ is less than $g(n)$, up to constant factors. Here are some examples :

1. $n = O(n^2)$.
2. $3n^2 + \log(n) = O(n^2)$.
3. For any constants $a, b \in \mathbb{N}$, we have $\log_a(n) = O(\log_b(n))$. *So we will often ignore the base of the logarithm, since it's the same up to constants.*
4. $n^3 = 2^{O(\log(n))}$. Why ? $n^3 = 2^{3\log(n)}$ and $3\log(n) = O(\log(n))$.

You should know that when you have a polynomial $f := a_k n^k + a_{k-1} n^{k-1} + \ldots a_1 x + a_0$, we have $f(n) = O(n^k)$. This also means $an^k = O(n^k)$.

While O notation, means "$f \leq g$ asymptotically", we can define o which says "$f < g$ asymptotically".

Definition 2. Given two functions f, $g : \mathbb{N} \rightarrow \mathbb{R}^+$, we say that $f(n) = o(g(n))$ if **for all** constants $c > 0$, there exists n_0 such that :

$$\forall n \geq n_0 \ : \ f(n) < c * g(n)$$

Notice that the definition is slightly different. Here are some examples :

1. $\sqrt{n} = o(n)$. Why ?
   Given c, we set $n_0 = 1/c^2$. Then for $n > n_0$, we have $\sqrt{n} > 1/c$, so $\sqrt{n}c > 1$. So, $c \cdot n = \sqrt{n} \cdot c\sqrt{n} > \sqrt{n}$.
2. $n = o(n \log(n))$. Why ?
   Set $n_0 = 2^{1/c}$. Then for $n > n_0$, we have $c \cdot n \log(n) > n \cdot c \log(2^{1/c}) = n$.
3. $0.001n \neq o(n)$.

We will also need some extra notation, to mention $g \geq f$ or $f > g$ asymptotically :

Definition 3. Given two functions f, $g : \mathbb{N} \rightarrow \mathbb{R}^+$.

- We say that $g(n) = \Omega(f(n))$ if and only if $f(n) = O(g(n))$.
- We say that $g(n) = \omega(f(n))$ if and only if $f(n) = o(g(n))$.

Finally when f and g are the "same up to constants" we use Θ.

Definition 4. Given two functions f, $g : \mathbb{N} \rightarrow \mathbb{R}^+$, we say $f = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $g(n) = O(f(n))$.

Again are some examples :

1. $0.1n^3 = \Omega(n^3)$.
2. $n + \log_2(n) = \Omega(n)$.
3. $n^2 \log(n) = \omega(n^2 + \log(n))$.
4. $100n^2 + 3n - \sqrt{n} = \Theta(n^2)$.

2 Streaming algorithms lower bounds

Remember the following languages from the previous lecture.

$$L_1 := \{w \in \{0,1\}^* \mid \text{w has same \# 0s and 1s}\}$$

We gave a $2\lceil \log_2(n+1) \rceil = O(\log(n))$ space algorithm for L_1.

$$L_2 := \{w' \mid w' = ww \text{ and } w \in \{0,1\}^*\}$$

We gave a $O(n)$ space algorithm for L_2.

$$L_3 := \{w \in \{0,1\}^* \mid \text{\# of 1 in } w \text{ is divisible by } 8\}$$

We have a $O(1)$ (3 bits) space algorithm for L_3. Note that we can't hope to do better than constant space, so this optimal (up to constant).

We will now show how to give lower bounds for streaming algorithms.

Definition 5 (Length-n distinguishable strings)**.** Fix a language L over $\Sigma = \{0,1\}$. Two strings $x, y \in \Sigma^*$ are length-n distinguishable if there exists another string z such that :

- $|xz| \leq n$.
- $|yz| \leq n$.
- Exactly one of xz and yz is in L.

The idea is that an algorithm $\mathcal{A}$ must put x and y into different memory configurations after reading them. Why ? Since after reading x or y, $\mathcal{A}$ ends in the same memory configurations, it means it also ends in the same memory configurations on inputs xz and yz. But in one case you must accept and in the other you must reject. So $\mathcal{A}$ would have to make a mistake.

Definition 6 (Length-n distinguishing set)**.** A length-n distinguishing set for L is a set $S_n \subseteq \{0,1\}^*$ such that any two distinct x, y in S_n are length-n distinguishable.

Using length-n distinguishing sets is the main way we will be showing lower bounds. In particular, we have the following :

Theorem 1. If L has a length-n distinguishing set S_n, then any streaming algorithm for L must use $\Omega(\log(|S_n|))$ space for inputs of length $\leq n$.

Proof. The main idea is that the algorithm must put all the string x in S_n into different memory states after reading them. Let's assume to the contrary that L has a streaming algorithm $\mathcal{A}$ that decides it, and on inputs of length $\leq n$ the algorithm uses $p = \lfloor \log(|S_n|) \rfloor - 1$ space. There are

$$\sum_{i=0}^{p} 2^i = 2^{p+1} - 1 = 2^{\lfloor \log(|S_n|) \rfloor} - 1 < S_n$$

possible memory configurations for $\mathcal{A}$ on inputs of length $\leq n$.

By the pigeonhole principle [1], there must be two distinct strings $x, y \in S_n$, such that they are mapped to the same memory configuration M after running $\mathcal{A}$ on them.

If from M the algorithm reads in z, it reaches a new memory configuration M'. Thus on input xz and yz, $\mathcal{A}$ ends in memory configuration M'. But if the input was xz, the algorithm must reject, and if it was yz it must accept (or vice versa). But the stop rule of $\mathcal{A}$ can only map M' to one of Accept or Reject. So it must make a mistake on one of xy or yz.

Note that by the definition of a length-n distinguishing set, xz and yz both have length $\leq n$. So this implies that if $S(n) \leq \lfloor \log(|S_n|) \rfloor - 1$, $\mathcal{A}$ must make a mistake on some input of length $\leq n$. This contradicts that this is an algorithm that decides L. □

3 Lower bound examples

We will now show space lower bounds for L_1 and L_2, in particular we will show we can't do better than the algorithms we gave last lecture (up to constant factors).

Theorem 2. Any streaming algorithm for $L_1 := \{w \in \{0,1\}^* \mid whassame\#0sand1s\}$ must use $\Omega(\log_2(n))$ space on inputs of length $\leq n$.

Proof. We want to give a length-n distinguishing set for L. It suffices to give a set of size $\Omega(n)$. [2]

In particular, we will give a set of size $\lfloor n/2 \rfloor$, consider

$$S_n := \{1^a \mid 0 \leq \lfloor n/2 \rfloor\}.$$

Note that the algorithm from the previous lecture we gave for L_1 mapped all the strings in S_n to different memories. Indeed, on input 1^a, the algorithm would store that the number of 1s was a, and the number of 0s was 0.

[1] Here the pigeons are the strings in S_n, the holes are the memory configurations. There's $|S_n|$ strings, and $< |S_n|$ configurations.

[2] Even a set of size $n^{0.001}/100$ would be enough since $\log_2(n^{0.001}/100) = 0.001 \log(n) - \log(100) = O(\log(n))$.

We claim the above is a length n distinguishing set. Given $x = 1^a, y = 1^b \in S_n$ $(a \neq b)$, we set $z = 0^a$. We then have that $xz = 1^a0^a$ must be accepted while $yx = 1^b0^a$ must be rejected. Also since $a, b \leq n/2$ by the definition of S_n, we have that $|x|, |y|, |z| \leq n/2$ so $|xz|, |yz| \leq n$.

So since this is a size $\lfloor n/2 \rfloor$ length-n distinguishing set for L_1, we have by Theorem 1 that $S(n) = \Omega(\log(n))$. □

In particular, from our upper bound of $O(\log_2(n))$ on the space needed by an algorithm for L_1, we can conclude the space needed for a streaming algorithm that decides this language is $\Theta(\log(n))$.

Theorem 3. Any streaming algorithm for $L_2 := \{w' \mid w' = ww \text{ and } w \in \{0,1\}^*\}$ must use $\Omega(n)$ space on inputs of length $\leq n$.

Proof. To prove this, we will construct a length-n distinguishing set of size $2^{\lfloor n/2 \rfloor}$, so let $k = \lfloor n/2 \rfloor$ and pick

$$S_n = \{x \in \{0,1\}^k \mid |x| \leq k\}.$$

This set has size 2^k. So assuming this is a length-n distinguishing set, by Theorem 1 we get our $\Omega(\lfloor n/2 \rfloor) = \Omega(n)$ space lower bound for L_2. So we now show this is indeed a length-n distinguishing set.

Given $x, y \in S_n$ $(x \neq y)$, we set $z = x$. We then have that $xz = xx$ must be accepted, while $yz = yx$ must be rejected (since $x \neq y$ and $|x| = |y|$). Also note that size $|x|, |y| = \lfloor n/2 \rfloor$ by the definition of S_n, so we have $|xz| = |yz| \leq n$. Hence, this is indeed a length-n distinguishing set. □