COMS E6998: Algorithms for Massive Data (Fall'23) — Oct 30, 2023

# Lecture 16: Monotonicity Testing, LIS, LCS

Instructor: *Alex Andoni* — Scribes: *Andrei Coman*

## 1 Monotonicity Testing

**Algorithm 1** Monotonicity Testing (Under Distinct Elements Assumption)

**for** iter $= 1, ..., T = O(\frac{1}{\epsilon})$ **do**
  Let $i \in_r [n]$
  Binary search for $y \triangleq x_i$ in $x[1, ..., n]$
  Reject if the binary search did not return $i$
  **return** Accept

**Algorithm 2** Binary Search

**Input:** Interval $[s, t]$
**if** $s = t$ **then**
  **return** s
$m \leftarrow \lfloor \frac{s+t}{2} \rfloor$
**if** $x_m < x_s$ or $x_m > x_t$ **then**
  **return** Reject
**if** $y \leq x_m$ **then**
  Recurse on $[s, m]$
**else**
  Recurse on $[m+1, t]$

**Claim 1.** *If $x$ is $\varepsilon$-far from increasing, then*

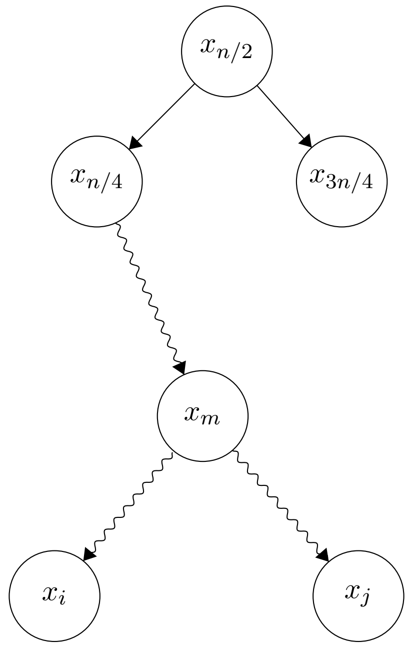
$$\mathbb{P}r_{i \in_r [n]}\left[\textit{binary search fails}\right] \geq \varepsilon$$

*Proof.* We call an index $i$ "good" if binary search on $x_i$ succeeds.
Now, consider two indices good indices $i, j \in [n], i < j$ and let $x_m = \text{LCA}(x_i, x_j)$. We have

$$x_m = \text{LCA}(x_i, x_j) \Rightarrow x_i < x_m < x_j \Rightarrow x_i < x_j$$

Therefore, the values at good indices are sorted in increasing order. So, all good indices form an increasing subsequence of $x$. However, since $x$ is $\varepsilon$-far from increasing, its longest increasing subsequence has length at most $(1-\varepsilon)n$. Therefore, there can be at most $(1-\varepsilon)n$ good indices and

$$\mathbb{P}\text{r}_{i \in_r [n]}\left[\text{binary search fails}\right] \geq 1 - \frac{(1-\varepsilon)n}{n} = \varepsilon$$




□

**Fact 2.** *Algorithm 1 is adaptive. The following modification makes it non-adaptive.*

---

**Algorithm 3** Non-adaptive Monotonicity Testing

---

Pick $i \in_r [n]$
Generate all the Binary Search queries assuming $x$ is sorted
Perform the original algorithm
**if** at any moment, algorithm reads an unexpected location **then**
  **return** Reject
**return** "Accept"

---

**Fact 3.** *Algorithm 1 assumes all values $(x_i)_{i=1}^n$ are distinct. This assumption can be relaxed as follows:*

---

**Algorithm 4** Monotonicity Testing

---

Modify $x$ into $x'$ over a different alphabet such that
  $x'_i = (x_i, i)$
Perform the original algorithm

---

**Remark 4.** *The $O\left(\frac{\log n}{\varepsilon}\right)$ sample-complexity of the algorithm above is tight.*

## 2 Longest Increasing Subsequence

The original problem (monotonicity testing) is equivalent to distinguishing between

$$\begin{gathered}\text{LIS}(x) = n \\ \text{and} \\ \text{LIS})(x) < (1-\varepsilon)n\end{gathered}$$

Can we estimate LIS($x$)?

**Fact 5.** *Suppose $x$ is a random permutation. Then*

$$\mathbb{E}[LIS(x)] = 2\sqrt{n} - c_1 n^{1/3} \pm O(n^{1/6})$$

*where $c_1 \approx 1.758$. Variance is of the order of $n^{1/3}$.*

1. (Saks-Seshadhri '17) We can distinguish between

$$\begin{gathered}\text{LIS}(x) > \lambda n \\ \text{and} \\ \text{LIS})(x) < \lambda n - \varepsilon n\end{gathered}$$

   within runtime $O\left((1/\varepsilon)^{O(1/\varepsilon)} \cdot (\log n)^{O(1)}\right)$ (notice the exponential dependence on $1/\varepsilon$).

   This is equivalent to estimating LIS($x$) up to an additive $\pm\varepsilon n$ term. The result is meaningful when LIS($x$) $\geq \frac{n}{\log n}$, i.e. $\varepsilon > \frac{1}{\log n}$, for, otherwise, 0 is a $\frac{1}{\log n}$-approximation.
2. (Andoni-Shekel Nosatzki-Sinha-Stein '22) Given that LIS($x$) $\geq n/k$, we can estimate LIS($x$) up to a factor of $O(n^{o(1)})$ within time $O(kn^{o(1)})$.

   **Note 6.** *If LIS($x$) $\approx n/k$, then $\Omega(k)$ time is necessary. Intuitively, this is because, in a string $x$ which is decreasing, with the exception of a random contiguous $n/k$-long increasing section, we need $\Omega(k)$ queries to hit the increasing area.*

## 3 Longest Common Subsequence (Length)

For $x, y \in \Sigma^n$, we define LCS($x, y$) to be the length of the longest non-contiguous common subsequence. This problem can be solved in $O(n^2)$ time via dynamic programming and, under the Strong Exponential Time Hypothesis (SETH), it requires $O(n^{2-o(1)})$ time.

**Theorem 7.** *(Shekel Nosatzki) If the Longest Increasing Subsequence can be computed within $O(k \cdot T)$ time within an $\alpha$-approximation, the Longest Common Subsequence length can be computed in $O(n \cdot T)$ time within an $\alpha$-approximation (or, also, in $O(n)$ time within an $\alpha T^2$-approximation).*

**Corollary 8.** *We can compute an $n^{o(1)}$-approximation of the LCS length within $O(n)$ time.*

# 4 Sublinear Algorithms for Graphs

Given a graph $G = (V, E), |V| = n, |E| = m$, two representations of $G$ are common:

1. Adjacency matrix: used for dense ($m \approx n^2$) graphs. If $m \ll n^2$, even finding an edge in the matrix takes more than $O(1)$-time.
2. Adjacency list: for every node $i \in V = [n]$, store a list of neighbours $A_i$. This representation admits several possible access queries:

   (a) $j^{\text{th}}$-neighbour of $i$ query $(i, j)$;

   (b) edge existence query (is $(i, j)$ an edge?).

Problems for graphs admitting sublinear algorithms are typically of the following types:

1. Testing: given some property $P$ (e.g. bipartiteness, (dis)connectedness), does $G$ have property $P$? Generally, we want to distinguish between $G$ having property $P$ and $G$ being $\varepsilon$-far from $P$, for some suitable notion of farness, most commonly:

   **Definition 9.** *$G$ is $\varepsilon$-far from $P$ if we need to delete at least $\varepsilon m$ edges from $G$ to satisfy $P$.*

   However, this is not a well-motivated definition and, therefore, we will not study problems of this type.
2. Estimate some function $f(G)$ (e.g. the value of $G$'s MST or the number of connected components)
3. Solve a problem on $G$ which admits multiple answers (e.g. find a coloring of $G$)

# 5 Problem: Estimating the MST Cost

**Assumption 10.** *all edges in $E$ have cost in $\{1, 2, ..., M\}$ for some constant $M$, and $G$ is connected. Access to $G$ is through an adjacency list with "$j^{th}$ neighbour" query (returns the edge and its weight).*

**Theorem 11.** $\forall \varepsilon > 0$*, we can estimate the MST cost up to a $(1 \pm \varepsilon)$-factor in time*

$$O\left(\frac{M^4 d}{\varepsilon^3}\right) \leq O\left(\frac{nM^4}{\varepsilon^3}\right)$$

*where $d$ is the maximum degree in $G$.*

**Note 12.** *The best known upper bound is $O\left(\frac{m}{n} M \left(\frac{1}{\varepsilon}\right)^{O(1)}\right)$.*