Solutions to Homework 1

1. (Preserving lexicographical order on suffixes.)

We are given string $S[1 : 3n]$ and $\tilde{S}[1 : 2n]$, and name array $N$ constructed according to the algorithm presented in class. Thus $N[i]$ is the lexicographic rank of $S[i : i + 2]$. Let $X$ and $Y$ be the front and back portions of $\tilde{S}$, i.e.,

$$X := (N[i] : i \mod 3 = 1), \quad Y := (N[i] : i \mod 3 = 2).$$

Proposition: $S_i \preceq S_j$ iff $\tilde{S}_I[i] \preceq \tilde{S}_I[j]$.

(The basic idea of the proof is to simulate the lexicographical comparisons we are trying to show, scanning left to right.)

Proof: If $i = j$ the claim is trivial, so assume $i \neq j$. Thus $S_i \neq S_j$ and $\tilde{S}_I[i] \neq \tilde{S}_I[j]$. We consider three cases:

**Case 1.** $i \mod 3 = j \mod 3 = 1$.

In this case, both $I[i]$ and $I[j]$ map to locations in $X$.

$(\Rightarrow)$ Suppose $S_i < S_j$. Then, either (a) there exists a least position $k$ at which $S_i[k] < S_j[k]$, or (b) $S_i$ is a prefix of $S_j$.

In (a), the corresponding $N[k'], N[k'']$ that contain position $k$ in $S_i$ and $S_j$, respectively, must satisfy $N[k'] < N[k'']$ by construction. So $\tilde{S}_I[i] < \tilde{S}_I[j]$ before reaching the boundary between $X$ and $Y$.

In (b), the last letter of $S_i$ corresponds to letter-triplet $(S[3n−1] S[3n] 0)$, whose corresponding $N[k'] < N[k'']$. So $\tilde{S}_I[i] < \tilde{S}_I[j]$, again before reaching the boundary between $X$ and $Y$.

$(\Leftarrow)$ Now suppose $\tilde{S}_I[i] < \tilde{S}_I[j]$. Again the lex. comparison will terminate within $X$ since the last letter of $X$ corresponds to letter-triplet $(S[3n−1] S[3n] 0)$, which cannot equal any other letter in $X$. So for corresponding suffixes of $S$, $S_i < S_j$, by the same reasoning as above.

**Case 2.** $i \mod 3 = j \mod 3 = 2$.

In this case, both $I[i]$ and $I[j]$ map to locations in $Y$. The same reasoning as above applies to this case, except that both suffixes keep to $Y$, and now the last letter of $S_i$ corresponds to letter-triplet $(S[3n] 0 0)$.
Case 3. $i \mod 3 = 1$ and $j \mod 3 = 2$.

Now $I[i]$ maps into $X$ and $I[j]$ maps into $Y$.

($\Rightarrow$) Suppose $S_i \prec S_j$. Again, either (a) there exists some $k$ such that $S_i[k] \prec S_j[k]$, or (b) $S_i$ is a prefix of $S_j$.

If condition (a) holds, the comparison in $\tilde{S}$ of $\tilde{S}_{I[i]}$, $\tilde{S}_{I[j]}$ does not pass the boundary between $X$ and $Y$, since (again) the last letter of $X$ is the special value $0$. So the compared letters in $\tilde{S}$ directly correspond to $S_i$, $S_j$. Hence $\tilde{S}_{I[i]} \prec \tilde{S}_{I[j]}$.

In (b), we see $\tilde{S}_{I[i]} \prec \tilde{S}_{I[j]}$, as the last letter of $S_i$ corresponds to triplet $(S[3n-1] S[3n] 0)$, so the corresponding $N[k'] < N[k'']$.

($\Leftarrow$) Now suppose $\tilde{S}_{I[i]} \prec \tilde{S}_{I[j]}$. The comparison will not pass the $X, Y$ boundary, as the last letter of $X$ does not equal any letter in $Y$. If the comparison does not involve the last letter of $X$ or $Y$, then $S_i \prec S_j$. The same holds if it does involve the last letters of $X$ or $Y$.

Case 4. $i \mod 3 = 2$ and $j \mod 3 = 1$.

Now $I[i]$ maps to $Y$ and $I[j]$ maps to $X$. The same reasoning of the prior case holds.

□

2. (Longest-common-prefix lengths from heights.)

Recall that $\text{lcp}(X, Y)$ is the length of the longest common prefix of strings $X$ and $Y$.

Proposition: For string $S[1 : n]$, let $A[1 : n]$ be its suffix array, and let $H[1 : n]$ be its height array, where $H[i] := \text{lcp}(S_A[i], S_A[i+1])$. Then for any two indices $1 \leq i < j \leq n$,

$$\text{lcp}(S_A[i], S_A[j]) = \min_{1 \leq k < j} \{H[k]\}. \quad (1)$$

Proof: We show that the righthand (RHS) side of (1) is both an upper and lower bound on the lefthand side.

(Upper bound.) We will show that, for all $i \leq k < j$,

$$\text{lcp}(S_A[i], S_A[j]) \leq H[k] = \text{lcp}(S_A[k], S_A[k+1]), \quad (2)$$

which implies the upper bound. We prove this by contradiction. Suppose (2) is false at some $k$ in the interval, and let $\ell$ be the RHS value there.

We define characters $a, b$ like so:

$$a := S_A[i][\ell + 1] = S_A[j][\ell + 1] \neq S_A[k'][\ell + 1] =: b,$$

for at least one $k' \in \{k, k + 1\}$. Since $a < b$ or $b < a$, either $S_A[k'] \prec S_A[i]$ or $S_A[k'] > S_A[j]$, contradicting the fact that $A$ is the suffix array for $S$. So claim (2) must hold.

(Lower bound.) Let $\ell$ be the RHS value of (1).

We have $S_A[k][1 : \ell] = S_A[k+1][1 : \ell]$ for all $i \leq k < j$, so transitively $S_A[i][1 : \ell] = S_A[j][1 : \ell]$. Thus $\text{lcp}(S_A[i], S_A[j]) \geq H[k] = \text{lcp}(S_A[k], S_A[k+1])$.

□
3. (Interval-minimum queries.)

Notation: For integers $i$ and $j$, with $i < j$, we write $[i, j]$ to denote the set $\{i, i+1, \ldots, j\}$. Likewise $[i, i]$ denotes the single-element set $\{i\}$.

Preprocessing. We construct a balanced binary search tree $T$ consisting of all the intervals of $[1, n]$ considered by a binary search that recursively splits the interval $[l, r]$ into $[l, m] \cup [m+1, r]$ where $m := \lfloor \frac{l+r}{2} \rfloor$. Thus the root of $T$ is the interval $[1, n]$, and the leaves of $T$ are the single-element intervals $[1, 1], [2, 2], \ldots, [n, n]$. Tree $T$ has $\Theta(n)$ nodes and height $\Theta(\log n)$.

Each node $v$ of $T$ has the field $v.\text{min} := \min_{l \leq i \leq r} \{A[i]\}$ where $[l, r]$ is the interval for $v$. Note that if $v$ is an internal node with children $L, R$, then $v.\text{min}$ can be computed in constant time by $v.\text{min} = \min\{L.\text{min}, R.\text{min}\}$. Constructing $T$ top-down and filling in its fields bottom-up takes $\Theta(n)$ preprocessing time.

Queries. To answer an interval-minimum query on $A$ for interval $[i, j]$, we find leaf $v$ corresponding to interval $[i, i]$, leaf $w$ corresponding to $[j, j]$, and the nearest common ancestor $u$ of $v, w$ by following the links from child to parent and marking nodes.

Note that $u$ is the deepest node in $T$ such that its interval $[l, r] \supseteq [i, j]$. Interval $[i, j]$ is exactly the union of the intervals for nodes that hang to the right on the path from $u$ to $v$, and hang to the left of the path from $u$ to $w$ (see shaded nodes of Fig. 1). Taking the minimum of the min fields for these nodes answers the query.

Finding $u, v, w$ and the minimum of the in-hanging nodes all take $O(\log n)$ time.

Figure 1: To conduct a query on $[i, j]$, first find the $[i, i]$ and $[j, j]$ leaves $v, w$ and their nearest common ancestor $u$. Then trace along the paths from $u$ to $v$ and $u$ to $w$, accumulating the minimum of $v, w$ and inwards-hanging subtrees (shaded nodes), whose intervals are subsets of $[i, j]$. (The left/right orientations of parent-child links indicate which subtrees are inwards-hanging.)
4. (Single-pair longest common substring.)

**Algorithm.**

(a) Given input strings $X$, $Y$, form string $S := X\#Y$, where $\#$ $\notin \Sigma$. Construct the suffix array $A$ and height array $H$ for $S$.

(b) For each position $A[i]$, mark whether that suffix starts in string $X$ or $Y$.

```
A
suffix origin: Y X X X X Y Y Y X
```

(c) Scan $A$ to find index $k$ such that

i. $A[k]$ and $A[k + 1]$ start in different strings, and

ii. $H[k]$ is maximum.

(d) Output the pair of substrings of $X, Y$ that correspond to positions $A[k], A[k + 1]$ and have length $H[k]$.

For strings $X, Y$ of lengths $m, n$ this takes $\Theta(m + n)$ total time.

**Correctness.** Consider a longest common substring $w$ of $X, Y$ that occurs at positions $i, j$ in $S$.

```
S
X \# Y
```

Suppose the suffixes $S_i, S_j$ occur at indices $\tilde{i}, \tilde{j}$ in $A$, where without loss of generality, $\tilde{i} < \tilde{j}$.

```
A
suffix origin: X \cdots X Y Y \cdots Y
```

Let index $k \in [\tilde{i}, \tilde{j}]$ be such that position $A[k]$ is in $X$ and $A[k + 1]$ is in $Y$ (which must exist). Notice that

$$
\ell := \text{lcp}(S_{A[k]}, S_{A[k+1]}) \\
= H[k] \\
\geq \min_{\tilde{i} \leq k < \tilde{j}} \{H[\tilde{k}]\} \\
= \text{lcp}(S_{A[\tilde{i}]}, S_{A[\tilde{j}]}) \quad \text{by problem 2} \\
= |w|.
$$
Since the substring of length $\ell$ starting at position $A[k], A[k+1]$ is a common substring of $X,Y$ and $\ell \geq |w|$, it is also a longest common substring of $X,Y$. The algorithm finds an optimal $k$ of this form, hence it finds a longest common substring of $X$ and $Y$. 
\[\square\]