APPENDIX A
OVERVIEW ON THE INDUCTION PHASE

Notice that $\mathsf{suf}(i) < \mathsf{suf}(j)$ if (1) $x[i] < x[j]$ or (2) $x[i] = x[j]$ and $\mathsf{suf}(i+1) < \mathsf{suf}(j+1)$; otherwise, $\mathsf{suf}(i) > \mathsf{suf}(j)$. This observation is utilized by the IS algorithms to sort suffixes as follows:

S1 Clear S-type sub-buckets in sa. Scan sa^* leftward and insert each element into the current rightmost empty position in the corresponding S-type sub-bucket.

S2 Clear L-type sub-buckets in sa and insert $n-1$ into the leftmost position in $\mathsf{sa\_bkt_L}(x[n-1])$. Scan sa rightward with i increasing from 0 to $n-1$. For each scanned non-empty $sa[i]$ with $t[sa[i]-1] = 0$, insert $sa[i]-1$ into the current leftmost empty position in $\mathsf{sa\_bkt_L}(x[sa[i]-1])$.

S3 Clear S-type sub-buckets in sa. Scan sa leftward with i decreasing from $n-1$ to 0. For each scanned non-empty $sa[i]$ with $t[sa[i]-1] = 1$, insert $sa[i]-1$ into the current rightmost empty position in $\mathsf{sa\_bkt_S}(x[sa[i]-1])$.

In brief, given sa^*, S1 inserts all the S*-type suffixes into sa in their lexical order. Then, S2-S3 induce the order of L- and S-type suffixes from those already sorted in sa, respectively, where the relative order of two suffixes induced into the same sub-bucket matches their insertion order according to the rule stated above. To be more specific, we show in Fig. 1 a running example of the induction phase.

As depicted, the input string x contains 6 S*-type suffixes sorted in line 3. When finished inserting the S*-type suffixes in lines 5-6, we first find the head of each L-type sub-bucket (marked by the symbol $\wedge$) and insert $\mathsf{suf}(13)$ into sa. Notice that $\mathsf{suf}(13)$ consists of only one character, it must be the smallest L-type suffixes starting with 1. Thus, we put $\mathsf{suf}(13)$ into the leftmost position in $\mathsf{sa\_bkt_L}(1)$ in line 8. Then, we scan sa from left to right for inducing the order of all the L-type suffixes. In lines 10-11, when visiting $sa[0] = 13$ (marked by the symbol @), we check the type array t to find $x[12] = 2$ is L-type and hence insert $\mathsf{suf}(12)$ into the current leftmost empty position in $\mathsf{sa\_bkt_L}(2)$. Similarly, in lines 12-13, we visit the next scanned item $sa[1] = 11$ and see that $t[10] = 0$, thus we place $\mathsf{suf}(10)$ into the current head of $\mathsf{sa\_bkt_L}(3)$. Following this way, we get all the L-type suffixes sorted in sa. After that, we first find the end of each S-type sub-bucket in lines 25-26 and scan sa leftward for inducing the order of all the S-type suffixes in lines 27-40. When visiting $sa[13] = 2$, we see $x[1]$ is S-type and thus put $\mathsf{suf}(1)$ into the current rightmost empty position in $\mathsf{sa\_bkt_S}(1)$. Then, at $sa[12] = 8$, we see $x[7] = 1$ is S-type and thus put $\mathsf{suf}(7)$ into the current rightmost empty position in $\mathsf{sa\_bkt_S}(1)$. To repeat scanning sa in this way, we get all the S-type suffixes sorted in sa.

The work in **[?]** describes how to compute the LCP array during the execution of S2-S3. Given two suffixes placed at the neighboring positions in sa, their LCP-value can be computed according to one of the following two cases in respect to whether or not they are inserted into the same sub-bucket: if yes, then their LCP-value is one greater than that of the two suffixes from which inducing them; otherwise, their LCP-value equals to zero. In this way, we can determine $lcp[i]$ immediately after the computation of $sa[i]$. The problem here is how to obtain the LCP-values of these inducing suffixes starting at the next positions in x,

00	p:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
01	$x[p]$:	2	1	3	1	3	1	2	1	3	1	3	1	2	1
02	$t[p]$:	L	S*	L	S*	L	S*	L	S*	L	S*	L	S*	L	L
03	$sa^*[p]$:	11	5	9	3	7	1								
04	Insert the sorted S*-type suffixes into sa^*:														
05	bucket:				1					2			3		
06	$sa^*[p]$:	{-1	11	5	9	3	7	1}	{-1	-1	-1}	{-1	-1	-1	-1}
07	Sort L-type suffixes:														
08	$sa^*[p]$:	{13	11	5	9	3	7	1}	{-1	-1	-1}	{-1	-1	-1	-1}
09		^							^			^			
10		{13	11	5	9	3	7	1}	{12	-1	-1}	{-1	-1	-1	-1}
11		@^								^		^			
12		{13	11	5	9	3	7	1}	{12	-1	-1}	{10	-1	-1	-1}
13		^	@							^			^		
14		{13	11	5	9	3	7	1}	{12	-1	-1}	{10	4	-1	-1}
15		^		@						^				^	
16		{13	11	5	9	3	7	1}	{12	-1	-1}	{10	4	8	-1}
17		^			@					^					^
18		{13	11	5	9	3	7	1}	{12	-1	-1}	{10	4	8	2}
19		^				@				^					^
20		{13	11	5	9	3	7	1}	{12	6	-1}	{10	4	8	2}
21		^					@				^				^
22		{13	11	5	9	3	7	1}	{12	6	0}	{10	4	8	2}
23		^						@			^				^
24	Sort S-type Suffixes:														
25		{13	-1	-1	-1	-1	-1	-1}	{12	6	0}	{10	4	8	2}
26								^			^				^
27		{13	-1	-1	-1	-1	-1	1}	{12	6	0}	{10	4	8	2}
28							^				^				@^
29		{13	-1	-1	-1	-1	7	1}	{12	6	0}	{10	4	8	2}
30						^					^			@	^
31		{13	-1	-1	-1	3	7	1}	{12	6	0}	{10	4	8	2}
32					^						^		@		^
33		{13	-1	-1	9	3	7	1}	{12	6	0}	{10	4	8	2}
34				^							^	@			^
35		{13	-1	-1	9	3	7	1}	{12	6	0}	{10	4	8	2}
36				^							@^				^
37		{13	-1	5	9	3	7	1}	{12	6	0}	{10	4	8	2}
38			^							@	^				^
39		{13	11	5	9	3	7	1}	{12	6	0}	{10	4	8	2}
40		^							@		^				^

Fig. 1. An Example for inducing the suffix and LCP arrays.

which is modeled as a range minimum query in **[?]** and can be answered within amortized $\mathcal{O}(1)$ time. For example, when scanning $sa[0]$ and $sa[5]$ in lines 10-11 and 20-21 of Fig. 1, $\mathsf{suf}(12)$ and $\mathsf{suf}(6)$ are sequentially induced into the neighboring positions in $\mathsf{sa\_bkt_L}(2)$. In the meantime, if we keep recording the minimum over $lcp(0,5]$, then we can obtain the LCP-value of the inducing suffixes $\mathsf{suf}(13)$ and $\mathsf{suf}(7)$ when putting the induced suffix $\mathsf{suf}(6)$ into sa.