Classify the type of each character

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	T	A	T	A	A	T	A	A	T	A	T	A	A	T	A	$
t	L	S	L	S	S	L	S	S	L	S	L	S	S	L	L	S

S = 1, L = 0 ② 0 1 ①

0 1 0 1 1 0 1 1

1-byte (8 bits) representation

```
void SA_IS(unsigned char *s, int *SA, int n, int K, int cs) {
   int i, j;
   unsigned char *t=(unsigned char *)malloc(n/8+1); // LS-type array in bits
   // Classify the type of each character
①  tset(n-2, 0); tset(n-1, 1); // the sentinel must be in s1, important!!!
②  for(i=n-3; i>=0; i--)
     tset(i, (chr(i)<chr(i+1) || (chr(i)==chr(i+1) && tget(i+1)==1))?1:0);
```

$S[i]$ is S-type if $S[i] = S[i+1]$, or $S[i]=S[i+1]$ and $S[i+1, n-1] = S[i+2, n-1]$

```
unsigned char mask[]={0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01};
#define tget(i) ( (t[(i)/8]&mask[(i)%8]) ? 1 : 0 )
#define tset(i, b) t[(i)/8]=(b)?(mask[(i)%8]|t[(i)/8]):((~mask[(i)%8])&t[(i)/8])
#define chr(i) (cs==sizeof(int)?((int*)s)[i]:((unsigned char *)s)[i])
```

In the initial step, s s is an array of 1-byte characters, but in subsequent steps, s can be an array of integers.

Sort all the S-substrings

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S	T	A	T	A	A	T	A	A	T	A	T	A	A	T	A	$
t	L	S	L	S	S	L	S	S	L	S	L	S	S	L	L	S
		*		*			*			*		*				*

* = LMS

② Pack LMS from the ends of each bucket.

```
        $        A                                          T
③ SA: {15} {-1 -1 -1 -1 11=09=06=03=01} {-1 -1 -1 -1 -1 -1}
        @^     ^                            ^
       {15} {14 -1 -1 -1 11=09=06=03=01} {-1 -1 -1 -1 -1 -1}
         ^   @    ^                         ^
       {15} {14 -1 -1 -1 11=09=06=03=01} {13 -1 -1 -1 -1 -1}
         ^        ^        @                   ^
       {15} {14 -1 -1 -1 11=09=06=03=01} {13<10=08=05=02=00}
         ^                           @                      ^
```

Some steps are omitted.

①

bkt (size)				
$	A	C	G	T
1	9	0	0	6

②

bkt (end position)				
$	A	C	G	T
1	10	10	10	16

③

bkt (start positions)				
$	A	C	G	T
0	1	10	10	10

This table is used for moving ^ forward.

```
   int *bkt = (int *)malloc(sizeof(int)*(K+1)); // bucket array
①  getBuckets(s, bkt, n, K, cs, true); // find ends of buckets
   for(i=0; i<n; i++) SA[i]=-1;
②  for(i=1; i<n; i++)  if(isLMS(i)) SA[--bkt[chr(i)]]=i;

③  induceSAl(t, SA, s, bkt, n, K, cs, false); // process from the beginning
④  induceSAs(t, SA, s, bkt, n, K, cs, true);  // process from the end
```

```
①  // find the start or end of each bucket
   void getBuckets(unsigned char *s, int *bkt, int n, int K, int cs, bool end) {
      int i, sum=0;
      for(i=0; i<=K; i++) bkt[i]=0; // clear all buckets
      for(i=0; i<n; i++) bkt[chr(i)]++; // compute the size of each bucket
      for(i=0; i<=K; i++) { sum+=bkt[i]; bkt[i]=end ? sum : sum-bkt[i]; }
   }
```

end start

```
②  #define isLMS(i) (i>0 && tget(i) && !tget(i-1))
```

i-1	i
L	S
0	1

```
③  void induceSAl(unsigned char *t, int *SA, unsigned char *s, int *bkt,
                  int n, int K, int cs, bool end) {
      int i, j;
      getBuckets(s, bkt, n, K, cs, end); // find starts of buckets
      for(i=0; i<n; i++) {
         j=SA[i]-1;
         if(j>=0 && !tget(j)) SA[bkt[chr(j)]++]=j; }
   }
```

Sort all the S-substrings

```
     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
S    T  A  T  A  A  T  A  A  T  A  T  A  A  T  A  $
t    L  S  L  S  S  L  S  S  L  S  L  S  S  L  L  S
        *     *        *        *     *           *
```

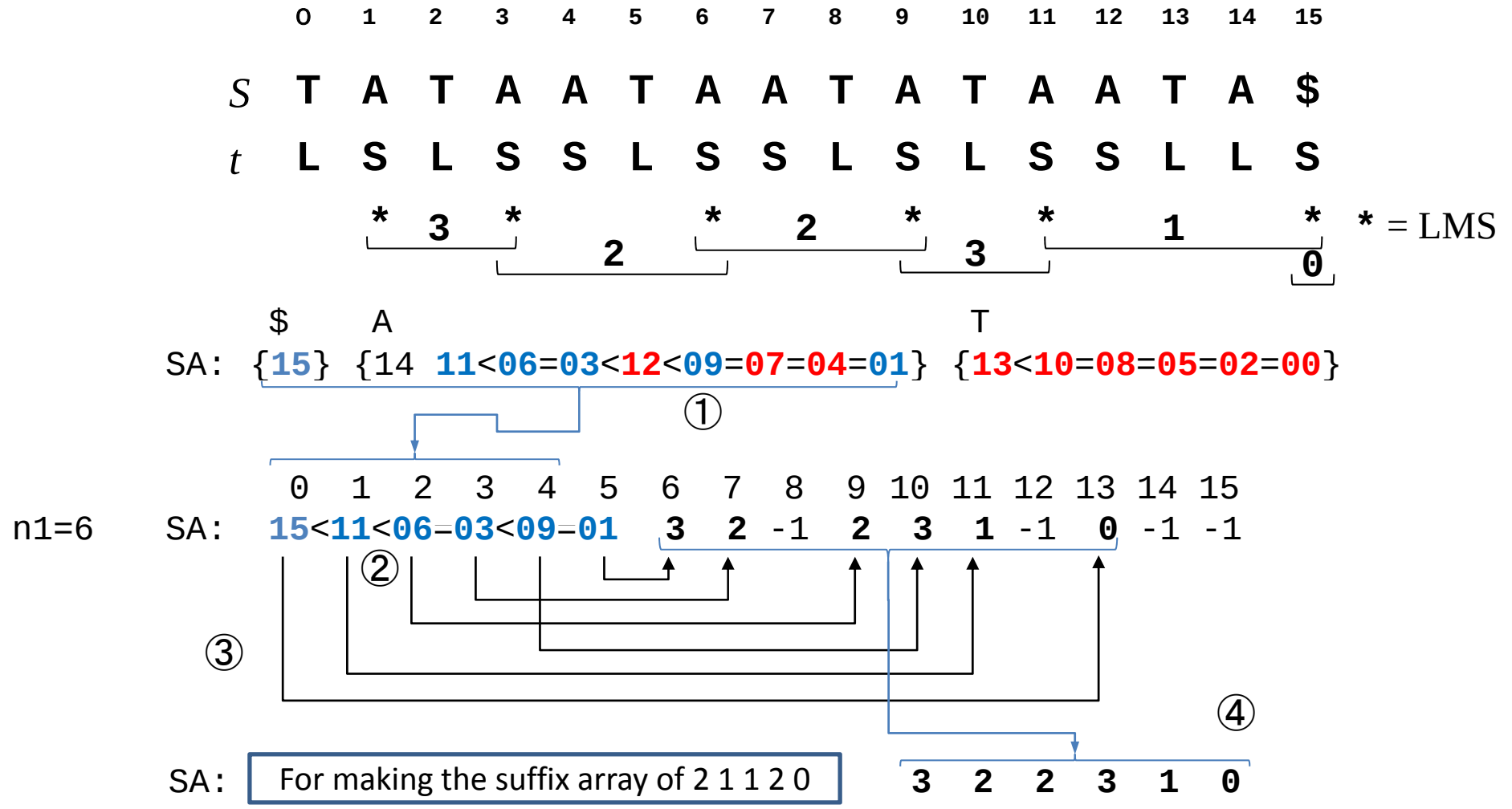
* = LMS

```
       $     A                                T
SA: {15} {14 -1 -1 -1 11=09=06=03=01} {13<10=08=05=02=00}
      ^        Some steps are omitted.      ^          @    ^
    {15} {14 -1 -1 -1 12<09=07=04=01} {13<10=08=05=02=00}
      ^              ^          @                           ^
    {15} {14 -1 -1 03<12<09=07=04=01} {13<10=08=05=02=00}
      ^           ^          @                              ^
    {15} {14 11<06=03<12<09=07=04=01} {13<10=08=05=02=00}
      ^     ^          @                                    ^
```

Bkt (ending position)				
$	A	C	G	T
1	10	10	10	16

This table is used for moving ^ backward.

④
```
void induceSAs(unsigned char *t, int *SA, unsigned char *s, int *bkt,
        int n, int K, int cs, bool end) {
   int i, j;
   getBuckets(s, bkt, n, K, cs, end); // find ends of buckets
   for(i=n-1; i>=0; i--) {
      j=SA[i]-1;
      if(j>=0 && tget(j)) SA[--bkt[chr(j)]]=j; }
}
```



① Pack LMS prefixed into SA according to the temporary ordering, and fill the remaining cells with -1.

② Check if neighboring LMS prefixes are equal or not.

③ Encode LMS prefixes into non-negative integers according to the ordering.

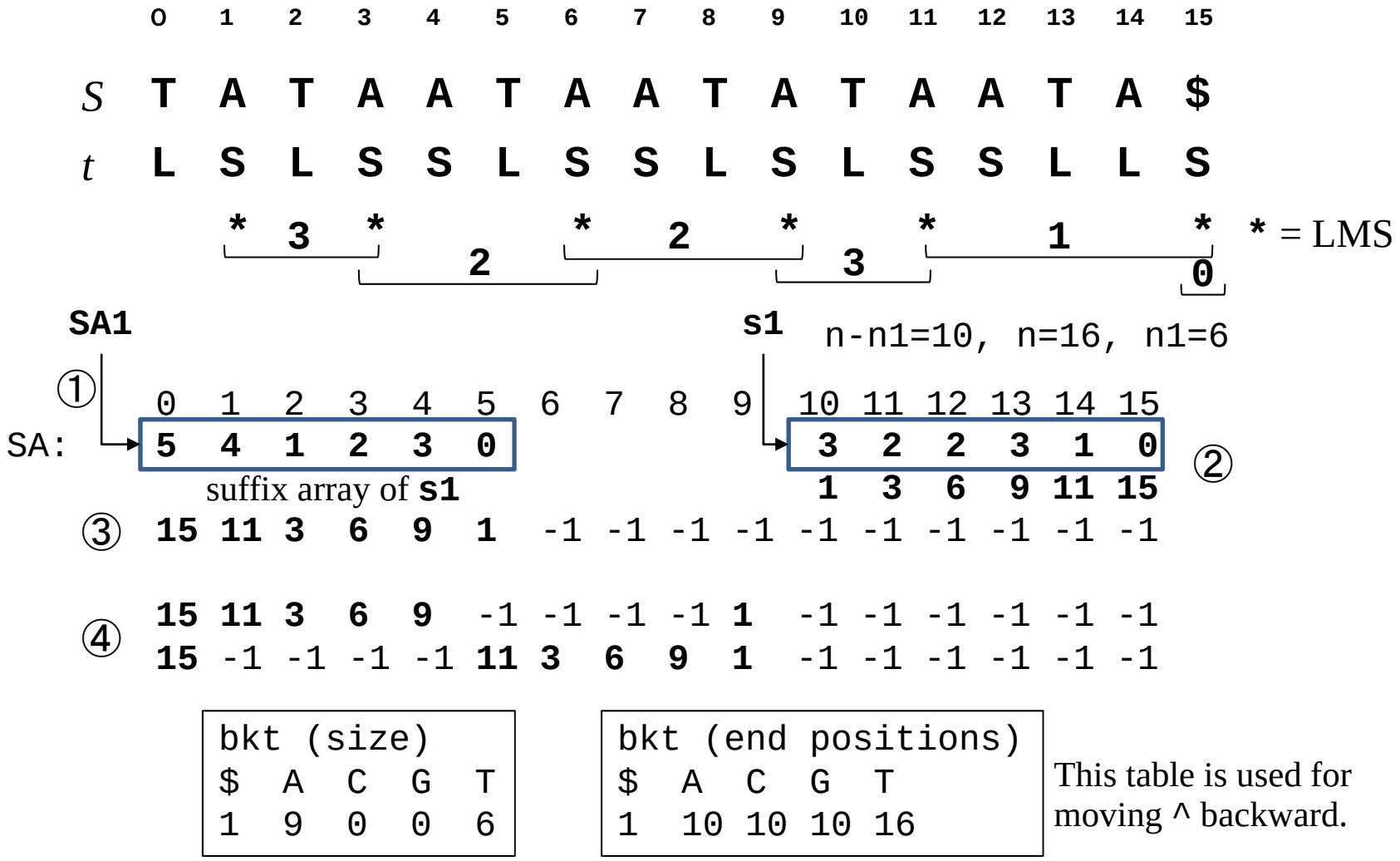
④ Move the encoded non-negative integers to the tail.

```
    // compact all the sorted substrings into the first n1 items of SA
    // 2*n1 must be not larger than n (proveable)
①   int n1=0;
    for(i=0; i<n; i++)
       if(isLMS(SA[i])) SA[n1++]=SA[i];

    // find the lexicographic names of all substrings
②   for(i=n1; i<n; i++) SA[i]=-1; // init the name array buffer
    int name=0, prev=-1;
    for(i=0; i<n1; i++) {
       int pos=SA[i]; bool diff=false;
       for(int d=0; d<n; d++)
          if(prev==-1 || chr(pos+d)!=chr(prev+d) || tget(pos+d)!=tget(prev+d))
             { diff=true; break; }
          else if(d>0 && (isLMS(pos+d) || isLMS(prev+d))) break;

③      if(diff) { name++; prev=pos; }
       pos=(pos%2==0)?pos/2:(pos-1)/2;
       SA[n1+pos]=name-1;
    }

④   for(i=n-1, j=n-1; i>=n1; i--)
       if(SA[i]>=0) SA[j--]=SA[i];
```



① Compute the suffix array of **s1**

② Decode the nonnegative integers to the original positions of LMS prefixes.

③ Translate the suffix array of **s1** into the ordered list of original positions, and fill the remaining cells with -1.

④ Move the original positions into proper positions.

```
   // stage 2: solve the reduced problem
   // recurse if names are not yet unique
   int *SA1=SA, *s1=SA+n-n1;
①  if(name<n1)
      SA_IS((unsigned char*)s1, SA1, n1, name-1, sizeof(int));
   else // generate the suffix array of s1 directly
      for(i=0; i<n1; i++) SA1[s1[i]] = i;

   // stage 3: induce the result for the original problem
   bkt = (int *)malloc(sizeof(int)*(K+1)); // bucket array

   // put all left-most S characters into their buckets
   getBuckets(s, bkt, n, K, cs, true); // find ends of buckets
②  for(i=1, j=0; i<n; i++)
      if(isLMS(i)) s1[j++]=i; // get p1
③  for(i=0; i<n1; i++) SA1[i]=s1[SA1[i]]; // get index in s
   for(i=n1; i<n; i++) SA[i]=-1; // init SA[n1..n-1]
④  for(i=n1-1; i>=0; i--) {
      j=SA[i]; SA[i]=-1;
      SA[--bkt[chr(j)]]=j;
   }
```

```
     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
S    T  A  T  A  A  T  A  A  T  A  T  A  A  T  A  $
t    L  S  L  S  S  L  S  S  L  S  L  S  S  L  L  S
        *     *        *        *     *           *     * = LMS

      $     A                             T
SA: {15} {-1 -1 -1 -1<11<03<06<09<01} {-1 -1 -1 -1 -1 -1}
      @^    ^                           ^
    {15} {14 -1 -1 -1<11<03<06<09<01} {-1 -1 -1 -1 -1 -1}
      ^    @   ^                       ^
    {15} {14 -1 -1 -1<11<03<06<09<01} {13 -1 -1 -1 -1 -1}
      ^       ^        @    途中経過省略      ^
    {15} {14 -1 -1 -1<11<03<06<09<01} {13<10<02<05<08<00}
      ^                          @                    ^
```

bkt_size				
$	A	C	G	T
1	9	0	0	6

bkt_start				
$	A	C	G	T
0	1	10	10	10

This table is used for moving ^ forward.

induceSAl(t, SA, s, bkt, n, K, cs, false);

- Scan SA from the left by moving "@" on position *i* such that *S*[*i*-1] is L-type, and put position *i* -1 into the open slot labeled with "^" in the bucket of *S*[*i*-1].
- **Proposition:** In each bucket, L-type suffixes come before S-type ones.
- **Proposition:** Suffixes newly added are sorted among each bucket.

```
      0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
S     T  A  T  A  A  T  A  A  T  A  T  A  A  T  A  $
t     L  S  L  S  S  L  S  S  L  S  L  S  S  L  L  S
         *     *        *        *     *           *      * = LMS

      $    A                                   T
SA:  {15} {14 -1 -1 -1 11<03<06<09<01} {13<10<02<05<08<00}
       ^                              ^              @     ^
     Some steps are omitted.
     {15} {14 -1 -1 -1 12<09<01<04<07} {13<10<02<05<08<00}
       ^              ^           @                        ^
     {15} {14 -1 -1 06<12<09<01<04<07} {13<10<02<05<08<00}
       ^           ^           @                           ^
     {15} {14 -1 03<06<12<09<01<04<07} {13<10<02<05<08<00}
       ^        ^        @                                 ^
     {15} {14 11<03<06<12<09<01<04<07} {13<10<02<05<08<00}
```

bkt_size				
$	A	C	G	T
1	9	0	0	6

bkt_end				
$	A	C	G	T
1	10	10	10	16

This table is used for moving ^ backward.

```
inducеSAs(t, SA, s, bkt, n, K, cs, true);
```

- Scan SA from the <u>right</u> by moving "@" on position i such that $S[i\text{-}1]$ is <u>S-type</u>, and put position i -1 into the open slot labeled with "^" in the bucket of $S[i\text{ -}1]$.
- **Proposition:** Suffixes newly added are sorted among each bucket.