1. **Suffix array construction:** consider the string $S = cabcccaccccb$. Let S_i be the suffix starting at position i, indexed from 0. We want to construct the suffix array for S in linear time.

   For reference we can write out S and the corresponding indices:

index i	0	1	2	3	4	5	6	7	8	9	10	11
$S[i]$	c	a	b	c	c	c	a	c	c	c	c	b

Step 1: What are R_1 and R_2 for S? How many special characters do we need at the end of each string? Sort the suffixes of the concatenation of R_1 and R_2. What are the roles of radix sort and recursion in this step?

Solution: R_1 and R_2 are string formed from making each triplet of letters into a single character, beginning at positions 1 and 2, respectively. Since $|S| \mod 3 = 0$, we need to add one special character to R_1 and two special characters to R_2.

$$R_1 = [abc][cca][ccc][cb\$]$$
$$R_2 = [bcc][cac][ccc][b\$\$]$$

The concatenation of R_1 and R_2 is R:

$$R = [abc][cca][ccc][cb\$][bcc][cac][ccc][b\$\$]$$

To sort the suffixes of R, we first use radix sort to sort the characters in R, each of which have length 3. Then we can lexicographically rename these characters to get the string we will recurse on: R'.

sorted triples		new label
$[abc]$	$\rightarrow$	A
$[b\$\$]$	$\rightarrow$	B
$[bcc]$	$\rightarrow$	C
$[cac]$	$\rightarrow$	D
$[cb\$]$	$\rightarrow$	E
$[cca]$	$\rightarrow$	F
$[ccc]$	$\rightarrow$	G

$$\Rightarrow \quad R' = AFGECDGB$$

If all the characters in R' had been unique, then we would be done sorting the suffixes of R'. Because we have repeated characters, we need to use recursion by calling DC3 on R'. Translating back into the original string indices, this gives us the suffix ordering:

$$\boxed{S_1 < S_{11} < S_2 < S_5 < S_{10} < S_4 < S_8 < S_7}$$

Step 2: Sort the suffixes beginning at the indices i where $i \mod 3 = 0$. How can we use what we've already done in Step 1?

Solution: Now we want to sort the suffixes S_0, S_3, S_6, S_9. We can instead sort the tuples $(S[i], \text{rank}(S_{i+1}))$ since we know all the ranks for the suffixes S_{i+1} since $(i+1) \mod 3 = 1$ in this case.

$$S_0 \to (c, 1)$$

$$S_3 \to (c, 6)$$

$$S_6 \to (a, 8)$$

$$S_9 \to (c, 5)$$

The key observation here is that we can now sort these tuples using radix sort. Now we have the suffix ordering:

$$\boxed{S_6 < S_0 < S_9 < S_3}$$

Step 3: Merge the two lists of sorted suffixes. If we are comparing S_i and S_j, where $i \mod 3 = 0$, how do the cases $j \mod 3 = 1$ and $j \mod 3 = 2$ differ?

Solution: Now we have two lists of sorted suffixes, and we want to merge them. Let $B_i = \{j \mid j \mod 3 = i\}$, for $i = 0, 1, 2$. We have two cases.

Case 1 $i \in B_0$ and $j \in B_1$. Then we want to compare:

$$S_i \to (S[i], \text{rank}(S_{i+1}))$$

$$S_j \to (S[j], \text{rank}(S_{j+1}))$$

since $i + 1 \in B_1$ and $j + 1 \in B_2$, which index suffixes we have already sorted.

Case 2 $i \in B_0$ and $j \in B_2$. Then we need to go one character further:

$$S_i \to (S[i], S[i+1], \text{rank}(S_{i+2}))$$

$$S_j \to (S[j], S[j+1], \text{rank}(S_{j+1}))$$

since $i + 2 \in B_2$ and $j + 2 \in B_1$ in this case.

Performing this merge step yields the final result:

$$\boxed{S_1 < S_6 < S_{11} < S_2 < S_0 < S_5 < S_{10} < S_4 < S_9 < S_3 < S_8 < S_7}$$

As a suffix array this would look like (because of the \$):

$$[12, 1, 6, 11, 2, 0, 5, 10, 4, 9, 3, 8, 7]$$

2. Let $B = nkknak\$reia$ be the Borrows-Wheeler transform of a string S. Find the original string S.

   Solution: B represents the last characters of the sorted rotations of S. First we find the first characters of the sorted rotations:

row	F $\cdots$	$\cdots$ L
1	\$	n
2	a	k
3	a	k
4	e	n
5	i	a
6	k	k
7	k	\$
8	k	r
9	n	e
10	n	i
11	r	a

   Now beginning with \$ in the L column, we see that the first character of our string must be k (row 7). This is the second k in the F column, so we now need to find the second k in the L column, which occurs in row 3. Looking at the character in the F column of row 3, we see the next character is an a. Continuing in this way, we get:

$$S = karkkainen$$

   Kärkkäinen and Sanders (2003) wrote the DC3 algorithm for suffix array construction presented in class.

3. Say we wanted to create the DC6 algorithm. Which indices should we sort in Step 1 (i.e. which numbers mod 6), and which should we sort in Step 2? What general property are we looking for when dividing up the indices?

   Solution: (sketch) During the merge step, we used the fact that the set $\{1, 2\}$ is a *difference cover* for 3, which means that all possible values of $i \mod 3$ can be constructed by taking differences within the set (mod 3). For example:

$$(1 - 1) \mod 3 = 0$$
$$(2 - 1) \mod 3 = 1$$
$$(1 - 2) \mod 3 = 2$$

   Then when merging, we can continue doing character comparisons until we hit suffixes which we have already sorted (i.e. whose start indices mod 3 are in the difference

cover). If we haven't sorted a proper difference cover, there will be some comparisons we cannot perform in this way. So for a DC6 algorithm, we need to find a difference cover of 6. One is $\{1, 2, 4\}$:

$$\begin{aligned}
(1-1) &\mod 6 = 0 \\
(2-1) &\mod 6 = 1 \\
(4-2) &\mod 6 = 2 \\
(4-1) &\mod 6 = 3 \\
(2-4) &\mod 6 = 4 \\
(1-2) &\mod 6 = 5
\end{aligned}$$