DC3 Algorithm Sketch

1. Construct the suffix array of a sample of the suffixes. In the sample, we include the suffixes starting at positions $i \bmod 3 \neq 0$. We recursively find the suffix array of a string of two-thirds length of the original string.
2. Construct the suffix array of the remaining suffixes using the result of the first step.
3. Merge the two suffix arrays into one using comparison-based merging.

DC3 Step 1: Construct the sample.

Given a string $T = yabbadabbado$, we construct suffix array $SA = [12, 1, 6, 4, 9, 3, 8, 2, 7, 5, 10, 11, 0]$.

DC3 Step 1: Construct the sample.

Given a string $T = yabbadabbado$, we construct suffix array $SA = [12, 1, 6, 4, 9, 3, 8, 2, 7, 5, 10, 11, 0]$.

1. For $k = 0, 1, 2$, we can define sets of indices

$$B_k = \{i \in [0, n] | i \bmod 3 = k\}$$

   Which indices do B_0, B_1, and B_2 contain?

DC3 Step 1: Construct the sample.

Given a string $T = yabbadabbado$, we construct suffix array $SA = [12, 1, 6, 4, 9, 3, 8, 2, 7, 5, 10, 11, 0]$.

1. For $k = 0, 1, 2$, we can define sets of indices

$$B_k = \{i \in [0, n] | i \bmod 3 = k\}$$

Which indices do B_0, B_1, and B_2 contain?

- $B_0 = \{0, 3, 6, 9, 12\}, B_1 = \{1, 4, 7, 10\}, B_2 = \{2, 5, 8, 11\}$.

DC3 Step 1: Construct the sample.

Given a string $T = yabbadabbado$, we construct suffix array $SA = [12, 1, 6, 4, 9, 3, 8, 2, 7, 5, 10, 11, 0]$.

1. For $k = 0, 1, 2$, we can define sets of indices

$$B_k = \{i \in [0, n] | i \bmod 3 = k\}$$

Which indices do B_0, B_1, and B_2 contain?

- $B_0 = \{0, 3, 6, 9, 12\}, B_1 = \{1, 4, 7, 10\}, B_2 = \{2, 5, 8, 11\}$.

Let S_i denote a suffix starting at index i in T. Let $C = B_1 \cup B_2$ be the set of sample start indices and S_C is the set of sample suffixes.

$$C = \{1, 4, 7, 10, 2, 5, 8, 11\}$$

$$S_C = \{S_1, S_4, S_7 ... S_8, S_{11}\}.$$

DC3 Step 2: Construct R to sort sample suffixes.

Recall that $T = yabbadabbado$

2. Construct a new string R to sort the sample suffixes.
   Let t_i be the i-th element of T. For $k = 1, 2$, we can construct the strings

$$R_k = [t_k t_{k+1} t_{k+2}][t_{k+3} t_{k+4} t_{k+5}] \dots [t_{\max B_k} t_{\max B_k + 1} t_{\max B_k + 2}].$$

   What do R_1 and R_2 look like?

$$R_1 = [abb][ada][bba][do0] \text{ and } R_2 = [bba][dab][bad][o00].$$

DC3 Step 2: Construct R to sort sample suffixes.

Recall that $T = yabbadabbado$

2. Construct a new string R to sort the sample suffixes.
   Let t_i be the i-th element of T. For $k = 1, 2$, we can construct the strings

$$R_k = [t_k t_{k+1} t_{k+2}][t_{k+3} t_{k+4} t_{k+5}] \ldots [t_{\max B_k} t_{\max B_k + 1} t_{\max B_k + 2}].$$

   What do R_1 and R_2 look like?

$$R_1 = [abb][ada][bba][do0] \text{ and } R_2 = [bba][dab][bad][o00].$$

DC3 Step 2: Construct R to sort sample suffixes.

Recall that $T = yabbadabbado$

2. Construct a new string R to sort the sample suffixes.
   Let t_i be the i-th element of T. For $k = 1, 2$, we can construct the strings

$$R_k = [t_k t_{k+1} t_{k+2}][t_{k+3} t_{k+4} t_{k+5}] \dots [t_{\max B_k} t_{\max B_k+1} t_{\max B_k+2}].$$

   What do R_1 and R_2 look like?

$$R_1 = [abb][ada][bba][do0] \text{ and } R_2 = [bba][dab][bad][o00].$$

DC3 Step 2: Construct R to sort sample suffixes.

Recall that $T = yabbadabbado$

2. Construct a new string R to sort the sample suffixes.
   Let t_i be the i-th element of T. For $k = 1, 2$, we can construct the strings

$$R_k = [t_k t_{k+1} t_{k+2}][t_{k+3} t_{k+4} t_{k+5}] \dots [t_{\max B_k} t_{\max B_k + 1} t_{\max B_k + 2}].$$

What do R_1 and R_2 look like?

$$R_1 = [abb][ada][bba][do0] \text{ and } R_2 = [bba][dab][bad][o00].$$

We can concatenate R_1 and R_2 into a string R.

$$R = [abb][ada][bba][do0][bba][dab][bad][o00]$$

DC3 Step 2: Construct R to sort sample suffixes.

Recall that $T = yabbadabbado$

2. Construct a new string R to sort the sample suffixes.
   Let t_i be the i-th element of T. For $k = 1, 2$, we can construct the strings

$$R_k = [t_k t_{k+1} t_{k+2}][t_{k+3} t_{k+4} t_{k+5}] \dots [t_{\max B_k} t_{\max B_k+1} t_{\max B_k+2}].$$

   What do R_1 and R_2 look like?

$$R_1 = [abb][ada][bba][do0] \text{ and } R_2 = [bba][dab][bad][o00].$$

   We can concatenate R_1 and R_2 into a string R.

$$R = [abb][ada][bba][do0][bba][dab][bad][o00]$$

   The nonempty suffixes of R correspond to S_C of sample suffixes. By sorting the suffixes of R, we get the order of the sample suffixes S_C.

DC3 Step 3: Sort the characters of R

Recall that $T = yabbadabbado$.

3. Sort the suffixes of R. First, radix sort the *characters* of R (the triples $[t_i t_{i+1} t_{i+2}]$) and rename them with their ranks to obtain a new string R'.

$$R = [abb][ada][bba][do0][bba][dab][bad][o00]$$

Rank	Index in R	Character
1	0	abb
2	1	ada
3	6	bad
4	2	bba
4	4	bba
5	5	dab
6	3	do0
7	7	o00

Index in R	R′ (Rank)
0	1
1	2
2	4
3	6
4	4
5	5
6	3
7	7

DC3 Step 4: Sort the suffixes of R' (if needed).

Recall that $T = yabbadabbado$, and $R' = [1, 2, 4, 6, 4, 5, 3, 7]$.

4. If any of the characters of R are the same, recursively sort the suffixes of R'.

Rank	Index in R′	Suffix
1	8	$
2	0	12464537$
3	1	2464537$
4	6	37$
5	4	4537$
6	2	464537$
7	5	537$
8	3	64537$
9	7	7$

DC3 Step 4: Sort the suffixes of R' (if needed).

Recall that $T = yabbadabbado$, and $R' = [1, 2, 4, 6, 4, 5, 3, 7]$.

4. If any of the characters of R are the same, recursively sort the suffixes of R'.

Rank	Index in R′	Suffix
1	8	$
2	0	12464537$
3	1	2464537$
4	6	37$
5	4	4537$
6	2	464537$
7	5	537$
8	3	64537$
9	7	7$

But how does this relate to the suffixes of the original string T?

DC3 Step 4: Sort the suffixes of R' (if needed).

We can write the correspondence between start indices of the suffixes R' to the start indices of T.

```
T = y a b b a d a b b a d o

R =  [a b b][a d a][b b a][d o 0]…
```

Start Index of Suffix in R′	Start Index of Suffix in T
0	1
1	4
2	7
3	10
4	2
5	5
6	8
7	11

DC3 Step 5: Use sorted order of R' to sort the sample suffixes of T

5. Combining the results in the last two tables, we see that we can assign a rank to each suffix in S_C.

Suffix Start Index in T	Rank of Suffix
0	x
1	1
2	4
3	x
4	2
5	6
6	x
7	5
8	3
9	x
10	7
11	8
12	x
13	0
14	0

DC3 Step 6: Sort non-sample suffixes

6. The non-sample suffixes are the suffixes with start indices in B_0. We represent each of these suffixes S_i by a tuple, $(t_i, \text{rank}(S_{i+1}))$.

Start Index of Suffix in T	Tuple Representation
0	(y, 1)
3	(b, 2)
6	(a, 5)
9	(a, 7)
12	(0, 0)

We can compare these suffixes as follows

$$S_i \leq S_j \iff (t_i, \text{rank}(S_{i+1})) \leq (t_j, \text{rank}(S_{j+1})).$$

Radix-sorting the tuples gives us an ordering of the non-sample suffixes.
What is the sorted order of these suffixes?

DC3 Step 6: Sort non-sample suffixes

6. The non-sample suffixes are the suffixes with start indices in B_0. We represent each of these suffixes S_i by a tuple, $(t_i, \text{rank}(S_{i+1}))$.

Start Index of Suffix in T	Tuple Representation
0	(y, 1)
3	(b, 2)
6	(a, 5)
9	(a, 7)
12	(0, 0)

We can compare these suffixes as follows

$$S_i \leq S_j \iff (t_i, \text{rank}(S_{i+1})) \leq (t_j, \text{rank}(S_{j+1})).$$

Radix-sorting the tuples gives us an ordering of the non-sample suffixes. What is the sorted order of these suffixes? $S_{12}, S_6, S_9, S_3, S_0$.

DC3 Step 7: (Almost done!) Merge

7. We can merge the two sorted sets of suffixes using standard comparison-based merging.
   To compare a suffix $S_i \in B_1$ with $S_j \in B_0$,

$$S_i \leq S_j \iff (t_i, \mathrm{rank}(S_{i+1})) \leq (t_j, \mathrm{rank}(S_{j+1})).$$

   To compare a suffix $S_i \in B_2$ with $S_j \in B_0$,

$$S_i \leq S_j \iff (t_i, t_{i+1}, \mathrm{rank}(S_{i+2})) \leq (t_j, t_{j+1}, \mathrm{rank}(S_{j+2})).$$

DC3 Step 7: (Almost done!) Merge

7. We can merge the two sorted sets of suffixes using standard comparison-based merging.
   To compare a suffix $S_i \in B_1$ with $S_j \in B_0$,
   $$S_i \leq S_j \iff (t_i, \text{rank}(S_{i+1})) \leq (t_j, \text{rank}(S_{j+1})).$$
   To compare a suffix $S_i \in B_2$ with $S_j \in B_0$,
   $$S_i \leq S_j \iff (t_i, t_{i+1}, \text{rank}(S_{i+2})) \leq (t_j, t_{j+1}, \text{rank}(S_{j+2})).$$

   Final Suffix Array: [12, 1, 6, 4, 9, 3, 8, 2, 7, 5, 10, 11, 0].

DC3 Step 7: (Almost done!) Merge

7. We can merge the two sorted sets of suffixes using standard comparison-based merging.
   To compare a suffix $S_i \in B_1$ with $S_j \in B_0$,

$$S_i \le S_j \iff (t_i, \text{rank}(S_{i+1})) \le (t_j, \text{rank}(S_{j+1})).$$

   To compare a suffix $S_i \in B_2$ with $S_j \in B_0$,

$$S_i \le S_j \iff (t_i, t_{i+1}, \text{rank}(S_{i+2})) \le (t_j, t_{j+1}, \text{rank}(S_{j+2})).$$

   Final Suffix Array: [12, 1, 6, 4, 9, 3, 8, 2, 7, 5, 10, 11, 0].
   What's the recurrence for this algorithm?

DC3 Step 7: (Almost done!) Merge

7. We can merge the two sorted sets of suffixes using standard comparison-based merging.
   To compare a suffix $S_i \in B_1$ with $S_j \in B_0$,

   $$S_i \leq S_j \iff (t_i, \text{rank}(S_{i+1})) \leq (t_j, \text{rank}(S_{j+1})).$$

   To compare a suffix $S_i \in B_2$ with $S_j \in B_0$,

   $$S_i \leq S_j \iff (t_i, t_{i+1}, \text{rank}(S_{i+2})) \leq (t_j, t_{j+1}, \text{rank}(S_{j+2})).$$

   Final Suffix Array: [12, 1, 6, 4, 9, 3, 8, 2, 7, 5, 10, 11, 0].
   What's the recurrence for this algorithm?

   $$T(n) = T\left(\frac{2n}{3}\right) + O(n).$$