APPENDIX A. DISTANCE COVARIANCE WITHOUT CHARACTERISTIC FUNCTIONS

The population version of distance covariance and distance correlation can be defined without characteristic functions, see Lyons (2013). This definition is as follows. Let $X \in \mathbb{R}^p$ and $Y \in \mathbb{R}^q$ be random variables with finite expectations. The random distance functions are $a(X, X') := |X - X'|_p$ and $b(Y, Y') = |Y - Y'|_q$. Here the primed random variable X' denotes an independent and identically distributed (i.i.d.) copy of the variable X, and similarly Y, Y' are i.i.d.

Introduce the real-valued function

$$m(x, F_X) = E[a(x, X)] = E|x - X| = \int |x - x'| dF_X(x'),$$

where F_X is the cumulative distribution function (cdf) of X, and

$$m(X, F_X) = \int |X - x'| dF_X(x'),$$

which is a real-valued random variable. For simplicity we write $m(x) := m(x, F_X)$ and $m(X) := m(X, F_X)$.

Next we define the counterpart of centered distance matrices. The centered distance function is

$$A(x, x') := a(x, x') - m(x) - m(x') + E[m(X')].$$

For random variables we have

$$A(X, X') = a(X, X') - m(X) - m(X') + E[m(X')],$$

where

$$E[m(X')] = E[m(X)] = E[m(X, F_X)] = \int \int |x - x'| \, dF_X(x') \, dF_X(x).$$

Similarly define the centered distance function $b(y, y')$ and the random variable $B(Y, Y')$. Now for X, X' i.i.d., and Y, Y' i.i.d., such that X and Y have finite expectations, the population distance covariance $\mathcal{V}(X, Y)$ is defined by

$$\mathcal{V}^2(X, Y) := E[A(X, X') \, B(Y, Y')]. \tag{A.1}$$

We have that $\mathcal{V}^2(X, Y)$ is always nonnegative, and equates zero if and only if X and Y are independent.

APPENDIX B. ALGORITHMS

Algorithm 1 is the algorithm that can compute for the distance covariance at $O(n \log n)$. Algorithm 2 realizes the idea that is described in the proof of Lemma 4.5. Algorithm 3 is a subroutine that will be called in Algorithm 2.

APPENDIX C. PROOFS

Proof of Lemma 3.3. Given the definitions of a_{ij} and b_{ij}, one can verify the following properties:

$$a_{ij} = a_{ji}, b_{ij} = b_{ji}, \quad \text{(symmetricity)}$$
$$a_{ii} = 0, b_{ii} = 0, \quad \text{(zero diagonal)}.$$

One can also verify the following equalities:

$$\sum_{i \neq j} a_{ij} b_{i\cdot} = \sum_{i=1}^{n} a_{i\cdot} b_{i\cdot}, \sum_{i \neq j} a_{ij} b_{\cdot j} = \sum_{j=1}^{n} a_{\cdot j} b_{\cdot j}, \sum_{i \neq j} b_{ij} a_{i\cdot} = \sum_{i=1}^{n} a_{i\cdot} b_{i\cdot}, \sum_{i \neq j} b_{ij} a_{\cdot j} = \sum_{j=1}^{n} a_{\cdot j} b_{\cdot j}; \tag{A.1}$$

$$\sum_{i \neq j} a_{i\cdot} = (n-1) a_{\cdot\cdot}, \quad \sum_{i \neq j} b_{i\cdot} = (n-1) b_{\cdot\cdot}; \tag{A.2}$$

$$a_{i\cdot} = a_{\cdot i}, \text{ and } b_{i\cdot} = b_{\cdot i}. \tag{A.3}$$

The following will be used in our simplification too. We have

$$\begin{aligned} \sum_{i \neq j} a_{i\cdot} b_{\cdot j} &= \sum_{i=1}^{n} a_{i\cdot} \sum_{j: j \neq i} b_{\cdot j} \\ &= \sum_{i=1}^{n} a_{i\cdot} (b_{\cdot\cdot} - b_{\cdot i}) \\ &= a_{\cdot\cdot} b_{\cdot\cdot} - \sum_{i=1}^{n} a_{i\cdot} b_{\cdot i} \\ &\overset{\text{(A.3)}}{=} a_{\cdot\cdot} b_{\cdot\cdot} - \sum_{i=1}^{n} a_{i\cdot} b_{i\cdot}; \end{aligned} \tag{A.4}$$

Similarly, we have

$$\sum_{i \neq j} b_{i\cdot} a_{\cdot j} = a_{\cdot\cdot} b_{\cdot\cdot} - \left(\sum_{i=1}^{n} a_{i\cdot} b_{i\cdot} \right). \tag{A.5}$$

Algorithm: Fast Computing for Distance Covariance (FaDCor)

Inputs: Observations $x_1, \ldots, x_n$, and $y_1, \ldots, y_n$.

Outputs: The distance covariance that was defined in (3.3).

1. Sort $x_1, \ldots, x_n$, and $y_1, \ldots, y_n$. Let I^x and I^y denote the order indices; i.e., if for $i, 1 \le i \le n$, $I^x(i) = k$, then x_i is the kth smallest observations among $x_1, \ldots, x_n$. Similarly if for $i, 1 \le i \le n$, $I^y(i) = k$, then y_i is the kth smallest observations among $y_1, \ldots, y_n$.

2. Let $x_{(1)} < \cdots < x_{(n)}$, and $y_{(1)} < \cdots < y_{(n)}$ denote the order statistics.
   Denote the partial sums:
   $$s^x(i) = \sum_{j=1}^{i} x_{(j)}, \quad s^y(i) = \sum_{j=1}^{i} y_{(j)}, \quad i = 1, \ldots, n.$$
   They can be computed using the following recursive relation: $s^x(1) = x_{(1)}, s^y(1) = y_{(1)}$,
   $$s^x(i+1) = s^x(i) + x_{(i+1)}, \quad s^y(i+1) = s^y(i) + y_{(i+1)}, \text{ for } i = 1, \ldots, n-1.$$

3. Compute α_i^x, α_i^y, β_i^x, and β_i^y that are defined in Lemma 4.1 and 4.2, using the following formula: for $i = 1, \ldots, n$, we have
   $$\alpha_i^x = I^x(i) - 1, \qquad \alpha_i^y = I^y(i) - 1,$$
   $$\beta_i^x = s^x(I^x(i) - 1), \qquad \beta_i^y = s^y(I^y(i) - 1).$$

4. Compute $x_{\cdot}$ and $y_{\cdot}$ per their definitions in Lemma 4.1 and 4.2.

5. Using (4.1) and (4.2), compute $\sum_{i-1}^{n} a_{i\cdot} b_{i\cdot}$.

6. Using (4.3) and (4.4), compute $a_{\cdot\cdot}$ and $b_{\cdot\cdot}$.

7. Use Algorithm *PartialSum2D* to compute for $\gamma_i(\{1\})$, $\gamma_i(\{x_j y_j\})$, $\gamma_i(\{y_j\})$, and $\gamma_i(\{x_j\})$.

8. Using (4.5) to compute $\sum_{i \ne j} a_{ij} b_{ij}$.

9. Finally, apply the results of steps 5., 6., and 8. to (3.3).

Algorithm 1: The $O(n \log n)$ algorithm to compute for the distance covariances.

Algorithm: Fast Algorithm for a 2-D Partial Sum Sequence (PartialSum2D)

Inputs: Observations $x_1, \ldots, x_n$, $y_1, \ldots, y_n$, and $c_1, \ldots, c_n$.

Outputs: Quantity $\gamma_i(\{c_j\}) = \sum_{j:j\neq i} c_j S_{ij}$ that is defined in Lemma 4.4.

1. Compute for the order statistics $x_{(1)} < \cdots < x_{(n)}$ for $x_1, \ldots, x_n$. Then rearrange triplets (x_i, y_i, c_j)'s such that we have $x_1 < \cdots < x_n$. Each triplet (x_i, y_i, c_j) $(1 \leq i \leq n)$ stay unchanged.

2. Let $y_{(1)} < \cdots < y_{(n)}$ denote the order statistics for $y_1, \ldots, y_n$, and assume that $I^y(i), i = 1, 2, \ldots, n$, are the order indices; i.e., if $I^y(i) = k$, then y_i is the k-th smallest among $y_1, \ldots, y_n$. Without loss of generality, we may assume that $y_i = I^y(i)$.

3. Evidently aforementioned function $I^y(i)$ is invertible. Let $(I^y)^{-1}(j)$ denote its inverse. Define the partial sum sequence: for $1 \leq i \leq n$,

$$s^y(i) = \sum_{j=1}^{i} c_{(I^y)^{-1}(j)}.$$

   The following recursive relation enables an $\mathrm{O}(n)$ algorithm to compute for all $s^y(i)$'s,

$$s^y(1) = c_{(I^y)^{-1}(1)}, \quad s^y(i+1) = s^y(i) + c_{(I^y)^{-1}(i+1)}, \text{ for } i \geq 1.$$

4. For $1 \leq i \leq n$, define

$$s^x(i) = \sum_{j=1}^{i} c_j.$$

   Again the above partial sums can be computed in $\mathrm{O}(n)$ steps.

5. Compute $c_\cdot = \sum_{j=1}^{n} c_j$.

6. Call Subroutine *DyadUpdate* to compute for $\sum_{j:j<i, y_j<y_i} c_j$ for all $i, 1 \leq i \leq n$.

7. By (A.6), we have that

$$\gamma_i(\{c_j\}) = c_\cdot - c_i - 2s^y(i) - 2s^x(i) + 4 \sum_{j:j<i, y_j<y_i} c_j.$$

Algorithm 2: A subroutine that will be needed in the fast algorithm for the distance covariance. This algorithm realizes the ideas in the proof of Lemma 4.5.

Subroutine: A Dyadic Updating Scheme (DyadUpdate)

Inputs: Sequence $y_1, \ldots, y_n$ and $c_1, \ldots, c_n$, where $y_1, \ldots, y_n$ is a permutation of $\{1, \ldots, n\}$.

Outputs: Quantities $\gamma_i := \sum_{j:j<i, y_j<y_i} c_j$, $i = 1, 2, \ldots, n$.

1. Recall that we have assumed $n = 2^L$. If n is not dyadic, we simply choose the smallest L such that $n < 2^L$. Recall that for $\ell = 0, 1, \ldots, L-1$, $k = 1, 2, \ldots, 2^{L-\ell}$, we define a close interval

$$I(\ell, k) := [(k-1) \cdot 2^\ell + 1, \ldots, k \cdot 2^\ell].$$

2. Assign $s(\ell, k) = 0, \forall \ell, k$, and $\gamma_1 = 0$.

3. For $i = 2, \ldots, n$, we do the following.

   (a) Fall all (ℓ, k)'s, such that $y_{i-1} \in I(\ell, k)$. Then for these (ℓ, k)'s, do update

$$s(\ell, k) \leftarrow s(\ell, k) + c_{i-1}.$$

   (b) Find nonnegative integers $\ell_1 > \cdots > \ell_\tau \geq 0$ such that

$$y_i - 1 = 2^{\ell_1} + \cdots + 2^{\ell_\tau}.$$

   Let $k_1 = 1$. For $j = 2, \ldots, \tau$, compute

$$k_j = (2^{\ell_1} + \cdots + 2^{\ell_{j-1}}) \cdot 2^{-\ell_j} + 1.$$

   (c) Compute $\gamma_i = \sum_{j=1}^{\tau} s(\ell_j, k_j)$.

Algorithm 3: A subroutine that will be called in Algorithm 2.

In the following, we simplify the statistic in (3.2). We have

$$\begin{aligned}
\Omega_n &\overset{(3.2)}{=} \frac{1}{n(n-3)}\sum_{i\neq j}\widetilde{A}_{i,j}\widetilde{B}_{i,j} \\
&\overset{(3.1)}{=} \frac{1}{n(n-3)}\sum_{i\neq j}\left(a_{ij}-\frac{a_{i\cdot}}{n-2}-\frac{a_{\cdot j}}{n-2}+\frac{a_{\cdot\cdot}}{(n-1)(n-2)}\right) \\
&\qquad\qquad \left(b_{ij}-\frac{b_{i\cdot}}{n-2}-\frac{b_{\cdot j}}{n-2}+\frac{b_{\cdot\cdot}}{(n-1)(n-2)}\right) \\
&= \frac{1}{n(n-3)}\sum_{i\neq j}\left[a_{ij}b_{ij}-\frac{a_{ij}(b_{i\cdot}+b_{\cdot j})}{n-2}-\frac{b_{ij}(a_{i\cdot}+a_{\cdot j})}{n-2}+\frac{(a_{i\cdot}+a_{\cdot j})(b_{i\cdot}+b_{\cdot j})}{(n-2)^2}\right. \\
&\qquad\left.+\frac{a_{ij}b_{\cdot\cdot}+b_{ij}a_{\cdot\cdot}}{(n-1)(n-2)}-\frac{(a_{i\cdot}+a_{\cdot j})b_{\cdot\cdot}+(b_{i\cdot}+b_{\cdot j})a_{\cdot\cdot}}{(n-1)(n-2)^2}+\frac{a_{\cdot\cdot}b_{\cdot\cdot}}{(n-1)^2(n-2)^2}\right].
\end{aligned}$$

Furthermore, we have

$$\begin{aligned}
\Omega_n &\overset{(A.2)}{=} \frac{1}{n(n-3)}\sum_{i\neq j}a_{ij}b_{ij} \\
&\qquad -\frac{1}{n(n-2)(n-3)}\sum_{i\neq j}[a_{ij}(b_{i\cdot}+b_{\cdot j})+b_{ij}(a_{i\cdot}+a_{\cdot j})] \\
&\qquad +\frac{1}{n(n-2)^2(n-3)}\sum_{i\neq j}(a_{i\cdot}+a_{\cdot j})(b_{i\cdot}+b_{\cdot j}) \\
&\qquad -\frac{a_{\cdot\cdot}b_{\cdot\cdot}}{(n-1)(n-2)^2(n-3)} \\
&\overset{(A.1),(A.3)}{=} \frac{1}{n(n-3)}\sum_{i\neq j}a_{ij}b_{ij}-\frac{4}{n(n-2)(n-3)}\sum_{i=1}^{n}a_{i\cdot}b_{i\cdot} \\
&\qquad +\frac{1}{n(n-2)^2(n-3)}\sum_{i\neq j}(a_{i\cdot}+a_{\cdot j})(b_{i\cdot}+b_{\cdot j})-\frac{a_{\cdot\cdot}b_{\cdot\cdot}}{(n-1)(n-2)^2(n-3)}.
\end{aligned}$$

Now bringing in (A.4) and (A.5), we have

$$\begin{aligned}
\Omega_n &= \frac{1}{n(n-3)}\sum_{i\neq j}a_{ij}b_{ij}-\frac{4}{n(n-2)(n-3)}\sum_{i=1}^{n}a_{i\cdot}b_{i\cdot}-\frac{a_{\cdot\cdot}b_{\cdot\cdot}}{(n-1)(n-2)^2(n-3)} \\
&\qquad +\frac{1}{n(n-2)^2(n-3)}\left[2(n-1)\sum_{i=1}^{n}a_{i\cdot}b_{i\cdot}+2\left(a_{\cdot\cdot}b_{\cdot\cdot}-\sum_{i=1}^{n}a_{i\cdot}b_{i\cdot}\right)\right] \\
&= \frac{1}{n(n-3)}\sum_{i\neq j}a_{ij}b_{ij}-\frac{2}{n(n-2)(n-3)}\sum_{i=1}^{n}a_{i\cdot}b_{i\cdot}+\frac{a_{\cdot\cdot}b_{\cdot\cdot}}{n(n-1)(n-2)(n-3)},
\end{aligned}$$

which is (3.3). □

Proof of Lemma 3.5. We use arithmetic induction. Suppose $n = r+1$, (3.6) becomes

$$(r+1)U_{r+1,r}(x_1,\ldots,x_{r+1}) = \sum_{i=1}^{r+1}U^{-i}_{r+1,r}(x_1,\ldots,x_{r+1}).$$

By defining $h(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{r+1}) = U^{-i}_{r+1,r}(x_1, \ldots, x_{r+1})$, we can verify that $h(\cdot)$ is a kernel function with r variables. Consequently, $U_{r+1,r}(x_1, \ldots, x_{r+1})$ can be written as (3.4).

Now suppose for any $n \geq n'$, $U_{nr}(x_1, \ldots, x_n)$ has the form as in (3.4), with the function $h(\cdot)$ that was defined above. Applying (3.6) with $n = n' + 1$, we can show that $U_{n'+1,r}(x_1, \ldots, x_{n'+1})$ still has the form as in (3.4), with the same function $h(\cdot)$ that was defined above. We omit further details. □

Proof of Theorem 3.8. It is evident to verify that the followings are true: for $i \neq k$,

$$\begin{aligned} a^{-k}_{i\cdot} &= a_{i\cdot} - a_{ik}, \\ b^{-k}_{i\cdot} &= b_{i\cdot} - b_{ik}, \\ a^{-k}_{\cdot\cdot} &= a_{\cdot\cdot} - a_{\cdot k} - a_{k\cdot} = a_{\cdot\cdot} - 2a_{\cdot k}, \\ b^{-k}_{\cdot\cdot} &= b_{\cdot\cdot} - 2b_{\cdot k}. \end{aligned}$$

We then have

$$\begin{aligned} \Omega^{-k}_{n-1} &= \frac{\sum_{i\neq j, i\neq k, j\neq k} a_{ij}b_{ij}}{(n-1)(n-4)} - \frac{2\sum_{i=1, i\neq k}^{n}(a_{i\cdot} - a_{ik})(b_{i\cdot} - b_{ik})}{(n-1)(n-3)(n-4)} \\ &\quad + \frac{(a_{\cdot\cdot} - 2a_{\cdot k})(b_{\cdot\cdot} - 2b_{\cdot k})}{(n-1)(n-2)(n-3)(n-4)}. \end{aligned}$$

For the right hand side of (3.7), we have the following:

$$\begin{aligned} \sum_{k=1}^{n}\Omega^{-k}_{n-1} &= \sum_{k=1}^{n}\frac{\sum_{i\neq j, i\neq k, j\neq k} a_{ij}b_{ij}}{(n-1)(n-4)} - \sum_{k=1}^{n}\frac{2\sum_{i=1, i\neq k}^{n}(a_{i\cdot} - a_{ik})(b_{i\cdot} - b_{ik})}{(n-1)(n-3)(n-4)} \\ &\quad + \sum_{k=1}^{n}\frac{(a_{\cdot\cdot} - 2a_{\cdot k})(b_{\cdot\cdot} - 2b_{\cdot k})}{(n-1)(n-2)(n-3)(n-4)} \\ &= \frac{(n-2)\sum_{i\neq j} a_{ij}b_{ij}}{(n-1)(n-4)} - \frac{2\left[(n-3)\sum_{i=1}^{n} a_{i\cdot}b_{i\cdot} + \sum_{i\neq k} a_{ik}b_{ik}\right]}{(n-1)(n-3)(n-4)} \\ &\quad + \frac{(n-4)a_{\cdot\cdot}b_{\cdot\cdot} + 4\sum_{k=1}^{n} a_{k\cdot}b_{k\cdot}}{(n-1)(n-2)(n-3)(n-4)} \\ &= \frac{\sum_{i\neq j} a_{ij}b_{ij}}{n-3} - \frac{2}{(n-2)(n-3)}\sum_{i=1}^{n} a_{i\cdot}b_{i\cdot} + \frac{a_{\cdot\cdot}b_{\cdot\cdot}}{(n-1)(n-2)(n-3)}. \end{aligned}$$

Compare with (3.3), we can verify that the above equates to $n \cdot \Omega_n$, which (per Theorem 3.6) indicates that Ω_n is a U-statistic. The kernel function of the corresponding U-statistic is the inner product that was defined in (3.2) with $n = 4$. □

Proof of Lemma 4.1. We have

$$\begin{aligned} a_{i\cdot} &= \sum_{\ell=1}^{n} a_{i,\ell} = \sum_{\ell=1}^{n} |x_i - x_\ell| \\ &= \sum_{x_\ell<x_i} (x_i - x_\ell) + \sum_{x_\ell>x_i} (x_\ell - x_i) \\ &= x_i \left(\sum_{x_\ell<x_i} 1 - \sum_{x_\ell>x_i} 1 \right) - \sum_{x_\ell<x_i} x_\ell + \sum_{x_\ell>x_i} x_\ell. \end{aligned}$$

It is easy to verify that

$$\sum_{x_\ell>x_i} 1 = n - 1 - \alpha_i^x,$$

and

$$\sum_{x_\ell>x_i} x_\ell = x_\cdot - x_i - \beta_i^x.$$

Taking into account the above two equations, we have

$$\begin{aligned} a_{i\cdot} &= (2\alpha_i^x - n + 1)x_i - \beta_i^x + x_\cdot - x_i - \beta_i^x \\ &= x_\cdot + (2\alpha_i^x - n)x_i - 2\beta_i^x, \end{aligned}$$

which is (4.1). □

Proof of Lemma 4.5. Without loss of generality (WLOG), we assume that $x_1 < x_2 < \cdots < x_n$. We have

$$\begin{aligned} \gamma_i(\{c_j\}) &= \sum_{j:j\neq i} c_j S_{ij} \\ &= \sum_{j:j>i,y_j>y_i} c_j + \sum_{j:j<i,y_j<y_i} c_j - \sum_{j:j>i,y_j<y_i} c_j - \sum_{j:j<i,y_j>y_i} c_j. \end{aligned}$$

Note that we can verify the following equations:

$$\begin{aligned} \sum_{j:j<i,y_j<y_i} c_j + \sum_{j:j>i,y_j<y_i} c_j &= \sum_{j:y_j<y_i} c_j, \\ \sum_{j:j<i,y_j<y_i} c_j + \sum_{j:j<i,y_j>y_i} c_j &= \sum_{j:j<i} c_j, \\ \sum_{j:j>i,y_j>y_i} c_j + \sum_{j:j<i,y_j<y_i} c_j + \sum_{j:j>i,y_j<y_i} c_j + \sum_{j:j<i,y_j>y_i} c_j &= \sum_{j:j\neq i} c_j = c_\cdot - c_i, \end{aligned}$$

where $c_\cdot = \sum_{j=1}^{n} c_j$. We can rewrite $\gamma_i(\{c_j\})$ as follows:

$$\gamma_i(\{c_j\}) = c_\cdot - c_i - 2 \sum_{j:y_j<y_i} c_j - 2 \sum_{j:j<i} c_j + 4 \sum_{j:j<i,y_j<y_i} c_j. \tag{A.6}$$

We will argue that the three summations on the right hand side can be implemented by $\mathrm{O}(n \log n)$ algorithms. First, term $\sum_{j:j<i} c_j$ is a formula for partial sums. It is known that an $\mathrm{O}(n)$ algorithm exists, by utilizing the relation:

$$\sum_{j:j<i+1} c_j = c_i + \sum_{j:j<i} c_j.$$

Second, after sorting y_j's at an increasing order, sums $\sum_{j:y_j<y_i} c_j$ is transferred into a partial sums sequence. Hence it can be implemented via an $\mathrm{O}(n)$ algorithm. If QuickSort (Hoare 1961) (Knuth 1997, Section 5.2.2: Sorting by Exchanging (pages 113-122)) is adopted, the sorting of y_j's can be done via an $\mathrm{O}(n \log n)$ algorithm.

We will argue that sums $\sum_{j:j<i,y_j<y_i} c_j, i = 1, \ldots, n$, can be computed in an $\mathrm{O}(n \log n)$ algorithm. WLOG, we assume that $y_i, i = 1, 2, \ldots, n$, is a permutation of the set $\{1, 2, \ldots, n\}$. WLOG, we assume that n is dyadic; i.e., $n = 2^L$, where $L \in \mathbb{N}$ or L is a nonnegative integer. For $\ell = 0, 1, \ldots, L-1$, $k = 1, 2, \ldots, 2^{L-\ell}$, we define an close interval

$$I(\ell, k) := [(k-1) \cdot 2^\ell + 1, \ldots, k \cdot 2^\ell].$$

We then define the following function

$$s(i, \ell, k) := \sum_{j:j<i, y_j \in I(\ell,k)} c_j,$$

where $i = 1, \ldots, n$, $\ell = 0, 1, \ldots, L-1$, and $k = 1, 2, \ldots, 2^{L-\ell}$.

We argue that computing the values of $s(i, \ell, k)$ for all i, ℓ, k, can be done in $\mathrm{O}(n \log n)$. First of all, it is evident that for all ℓ, k, we have

$$s(1, \ell, k) \equiv 0.$$

Suppose for all $i' \le i$, $s(i', \ell, k)$'s have been computed for all ℓ and k. For each $0 \le \ell \le L-1 < \log_2 n$, there is only one k^*, such that $y_i \in I[\ell, k^*]$. By the definition of $s(\cdot, \cdot, \cdot)$, we have

$$s(i+1, \ell, k) = \begin{cases} s(i, \ell, k) + c_i, & \text{if } k = k^*, \\ s(i, \ell, k), & \text{otherwise.} \end{cases}$$

The above dynamic programming style updating scheme needs to be run for n times (i.e., for all $1 \le i \le n$), however each stage requires no more than $\log_2 n$ updates. Overall, the computing for all $s(i, \ell, k)$ takes no more than $\mathrm{O}(n \log n)$.

For a fixed i, $1 \le i \le n$, we now consider how to compute for $\sum_{j:j<i,y_j<y_i} c_j$. If $y_i = 1$, obviously we have $\sum_{j:j<i,y_j<y_i} c_j = 0$. For $y_i > 1$, there must be a unique sequence of positive integers $\ell_1 > \ell_2 > \cdots \ell_\tau > 0$, such that

$$y_i - 1 = 2^{\ell_1} + 2^{\ell_2} + \cdots + 2^{\ell_\tau}.$$

Since $y_i \le n$, we must have $\tau \le \log_2 n$. We then define $k_\alpha, \alpha = 1, \ldots, \tau$ as follows

$$\begin{aligned}
k_1 &= 1, \\
k_2 &= 2^{\ell_1 - \ell_2} + 1, \\
&\vdots \\
k_\alpha &= (2^{\ell_1} + \cdots + 2^{\ell_{\alpha-1}})/2^{\ell_\alpha} + 1, \\
&\vdots \\
k_\tau &= (2^{\ell_1} + \cdots + 2^{\ell_{\tau-1}})/2^{\ell_\tau} + 1.
\end{aligned}$$

One can then verify the following: for $2 \le i \le n$,

$$\sum_{j:j<i,y_j<y_i} c_j = \sum_{\alpha=1}^{\tau} s(i, \ell_\alpha, k_\alpha).$$

Since $\tau \le \log_2 n$, the above takes no more than $\mathrm{O}(\log n)$ numerical operations. Consequently, computing $\sum_{j:j<i,y_j<y_i} c_j$ for all $i, 1 \le i \le n$, can be done in $\mathrm{O}(n \log n)$. (We realized that the above approach utilized the AVL tree structure (Adelson-Velskii and Landis 1962).) From all the above, we established the result. □