Algorithm 1: *DPPartition*

Input: *arr, l, r*

Initialisation:

1: **if** ($r - l < c$) **then**
2: *omp task nowait*
3: *qsort* () *or STLSort* ()
4: **end if**
5: $p = l + (r - 1)/2$
6: *MO5*(*arr*, *l*, *r*)
7: *omp task shared*(*new_midL*)
8: *new_midL = LPar* (*arr*, *l*, *p* – 1, *p*) //Partitioning algorithm
9: *omp task shared*(*new_midR*)
10: *new_midR = RPar* (*arr*, *p* + 1, *r*, *p*) //Partitioining algorithm
11: *omp taskwait*
12: *new_midC = MSwap* (*arr*, *new_midL*, *new_midR*, *p*)
13: *omp task*
14: *DPPartition* (arr, *l*, *new_midC* – 1)
15: *omp task*
16: *DPPartition* (*arr*, *new_midC* + 1, *r*)

Algorithm 2: *MO*5

Input: *arr, l, r*

Initialisation:

1: $p = l + (r - l)/2$

2: $q1 = l + (p - l)/2$

3: $q3 = m + (r - p)/2$

4: *SORT* (*arr*[*l*],*arr*[*q*1],*arr*[*p*],*arr*[*q*3],*arr*[*r*])

Algorithm 3: *LPar*

Input: *arr, l, r, p*

Output: *indexl*

Initialisation:

1: *val* = *arr*[*p*]
2: *indexl* = *l*
3: **for** *i* = *l*; *i* <= *r*;*i* = *i* + 1 **do**
4: **if** *arr*[*i*] <= *val* **then**
5: *swap*(*arr*[*i*], *arr*[*indexl*])
6: *indexl* = *indexl* + 1
7: **end if**
8: **end for**
9: **return** *indexl*

Algorithm 4: *RPar*

Input: *arr, l, r, p*

Output: *indexr*

Initialisation:

1: $val = arr[p]$

2: $indexr = r$

3: **for** $j = r; j >= l; j = j - 1$ **do**

4: **if** $arr[j] > val$ **then**

5: $swap(arr[j], arr[indexr])$

6: $indexr = indexr - 1$

7: **end if**

8: **end for**

9: **return** *indexr*

Algorithm 5: *MSwap*

Input: *arr, l, r, p*

Output: *i or j*

Initialization:

1: $i = l$

2: $j = r$

3: **while** $i < j$ *and* $i < p$ *and* $j > p$ **do**

4: *swap*(*arr*[*i*],*arr*[*j*])

5: $i = i + 1$

6: $j = j - 1$

7: **end while**

8: **if** $i > p$ **then**

9: *swap*(*arr*[*j*], *arr*[*p*])

10: **return** *j*

11: **else**

12: *swap*(*arr*[*i*], *arr*[*p*])

13: **return** *i*

14: **end if**