Overlapping intervals search

```r
# Define a function to find overlap in a series of time intervals
find_overlapping_intervals <- function(intervals) {

  # Create an events list
  events <- list()
  # For each interval, add a start event and an end event
  # event is a vector containing a point in time, an event type (start or end), and an int
  for (i in 1:length(intervals)) {
    events <- c(events, list(c(intervals[[i]][1], TRUE, i)), list(c(intervals[[i]][2], FAL
  }

  # Sort events by their coordinates, and then by their type (start before end)
  sorted_events <- events[order(sapply(events, '[', 1), -sapply(events, '[', 2))]


  # Initializes a vector to track the current activity interval
  active_intervals <- integer(0)

  # Initializes a list of the same length as the interval list to store overlapping interv
  overlap <- vector("list", length(intervals))

  # use for to iterate over sorted events
  for (event in sorted_events) {
    idx <- event[3]

    if (event[2]) { # If it's a start event
      for (active_idx in active_intervals) {
# Adds each other's indexes to the overlapping list of currently active intervals and curr
        overlap[[idx]] <- c(overlap[[idx]], active_idx)
        overlap[[active_idx]] <- c(overlap[[active_idx]], idx)
      }
# Adds the current interval to the active interval list
      active_intervals <- c(active_intervals, idx)
    } else { # If it's an end event
      active_intervals <- setdiff(active_intervals, idx)# Removes the current interval fro
    }
  }
# Returns a list of overlapping intervals
  return(overlap)
}

# Define a list of intervals to test
intervals <- list(c(1, 5), c(2, 6), c(1, 8), c(7, 9), c(10, 15))

# Using the function and print the result
overlaps <- find_overlapping_intervals(intervals)
print(overlaps)
```

```
[[1]]
[1] 3 2

[[2]]
[1] 1 3

[[3]]
[1] 1 2 4

[[4]]
[1] 3

[[5]]
NULL
```

1. Function definition:

The find_overlapping_intervals function receives a list of intervals, where each element is a two-digit operator that represents the start and end point of an interval.

2. Create event list:

The function first creates an empty list, events. For each interval in the interval, the function adds two events to events: A "start" event (marked TRUE) that contains the start time of the interval and the index of the interval. An “end” event (marked FALSE) that contains the end time of the interval and the index of the interval. This way, each interval is represented by two events in Events.

3. Sort events:

Use the order function to sort the events, first by point in time (time coordinates) and then by event type (start event before end event). The sorting step is important to check for overlap later.

4. Detection overlap:

Initialize active_intervals (an empty integer initialization) to track the time interval for the current activity (that is, it has started but not yet ended). Initializes overlap (a list) to store indexes of other intervals that each interval overlaps.

5. Handle each event:

Iterate through the sorted list of events: If a start event is encountered, its index is added to the mid-active_interval and the overlap is updated to reflect the overlap between the current active interval and the newly started interval. If it is an end event, the active_interval removes the corresponding interval index because the interval has ended.

6. Return result:

The function finally overlaps a list where each element is an integer operator, containing the indexes of other intervals that overlap the corresponding interval returns.

7. Test function:

In the last part of the code, I use list(c(1, 5), c(2, 6), c(1, 8), c(7, 9), c(10, 15)), call the function to test, and then print the result for overlaps.