Algorithm 1 EmergeSort with Distinct-Window Constraint

```
Require: $|items| \geq 2(w+1)+1$
procedure EmergeSort($items$, $cmp$, $w$)                          ▷ $w$ = window size
    /* — access bookkeeping — */
    $lastSeen[1 \dots n] \leftarrow 0$;    $tick \leftarrow 1$
    function MarkAccess($i, j$)
        $lastSeen[i]$, $lastSeen[j] \leftarrow tick$;    $tick \leftarrow tick + 1$
    end function
    function IsRecent($i$)
        return $(0 < tick - lastSeen[i] \leq w)$
    end function
    function Compare($i, j$)
        MarkAccess($i, j$)
        return $cmp(items[i], items[j])$
    end function

    /* — initial singleton runs — */
    $mergeQ \leftarrow$ deque of $(\{2i\}, \{2i{+}1\}, \varnothing)$ for $i = 0 \dots \lfloor n/2 \rfloor - 1$
    $danglingRun \leftarrow \{\ n-1\ \}$ if $n$ odd else $\varnothing$

    while $mergeQ \neq \varnothing$ do
        $(L, R, M) \leftarrow$ front($mergeQ$)
        if IsRecent($L$.front) $\vee$ IsRecent($R$.front) then          ▷ enforce distinct-window
            $pool \leftarrow \{k \mid \neg$IsRecent$(k)\}$
            choose $i, j$ randomly from $pool$
            Compare($i, j$)                                             ▷ placeholder comparison
            continue
        end if
        pop_front($mergeQ$)
        if Compare($L$.front, $R$.front) $\leq 0$ then
            push_back($M$, $L$.pop_front())
        else
            push_back($M$, $R$.pop_front())
        end if
        if $L \neq \varnothing$ and $R \neq \varnothing$ then
            push_back($mergeQ$, $(L, R, M)$)
            continue
        end if
        $M \leftarrow M \cup L \cup R$
        if $danglingRun = \varnothing$ then
            $danglingRun \leftarrow M$
        else
            push_back($mergeQ$, $(M, danglingRun, \varnothing)$)
            $danglingRun \leftarrow \varnothing$
        end if
    end while
    return $items$[indices in $danglingRun$]
end procedure
```