```
function 3DBPP($items$, $W$, $H$, $D$)
    $bins \leftarrow \emptyset$;
    $items \leftarrow items \cup$ GENERATESUPERITEMS($items$);
    repeat
        $packedItems \leftarrow$ PACKBIN($items, W, H, D$) ;
        $bin \leftarrow$ COMPLETEBIN($packedItems, W, H, D$);
        $bins \leftarrow bins \cup bin$;
        $items \leftarrow items \setminus packedItems$;
    until $items = \emptyset$;
    return $bins$;
end function
```

```
function PACKBIN($items$, $W$, $H$, $D$)
    $packed \leftarrow \emptyset$;
    $currentPacking \leftarrow \emptyset$;
    $toPack \leftarrow items$;
    $planes \leftarrow \{(0, \emptyset, \emptyset)\}$;      ▷ PriorityQueue, (z, supportItems, upperItems)
    repeat
        $p \leftarrow$ DEQUEUE($planes$);              ▷ Polls the lowest plane from the set
        $currentPacking \leftarrow$ PACKPLANE($p, toPack, W, H, D$);
        $packed \leftarrow packed \cup currentPacking$;
        $toPack \leftarrow toPack \setminus packed$;
        UPDATEPLANES($planes, currentPacking, 5$);
    until $planes = \emptyset \vee toPack = \emptyset \vee currentPacking = \emptyset$;
    return $packed$;
end function
```

```
procedure UPDATEPLANES($planes$, $packed$, $tolerance$)
    for $item$ in $packed$ do
        if $\nexists p \in planes : |p.z - item.z| \leq tolerance$ then
            $planes \leftarrow planes \cup (item.z, \emptyset, \emptyset)$;
        end if
        for $p \in planes : 0 \leq p.z - (item.z + item.w) \leq tolerance$ do
            $p.supportItems \leftarrow p.supportItems \cup item$;
        end for
        for $p \in planes : p.z < (item.z + item.w)$ do
            $p.upperItems \leftarrow p.upperItems \cup item$;
        end for
    end for
end procedure
```

```
function packPlane($plane$, $toPack$, $W$, $H$, $D$)
    $packed \leftarrow \emptyset$;
    repeat
        $bestScore \leftarrow 0$;
        $bestPacking \leftarrow null$;
        for $item \in toPack : plane.z + item.z \leq H$ do
            $packing \leftarrow$ 2DBPPwithObstacles($item, packed, plane, W, H, D$);
            removeInfeasibilities($packing$);
            $score \leftarrow$ scorePacking($packed \cup packing, W, H, D$);
            if $score > bestScore$ then
                $bestScore \leftarrow score$;
                $bestPacking \leftarrow packing$;
            end if
        end for
        $packed \leftarrow packed \cup bestPacking$;
        $toPack \leftarrow toPack \setminus bestPacking$;
    until $toPack = \emptyset \vee bestPacking = null$;
    return $packed$;
end function
```

```
function scorePacking($packed$, $W$, $H$, $D$)
    $A \leftarrow$ sumArea($packed$);
    $V \leftarrow$ sumVolume($packed$);
    return $A + V$;
end function
```