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 ← ∅;
    repeat
        bestScore ← 0;
        bestPacking ← null;
        for item ∈ toPack : plane.z + item.z ≤ H do
            packing ← 2DBPPwithObstacles(item, packed, plane, W, H, D);
            score ← scorePacking(packed ∪ packing, W, H, D);
            if score > bestScore then
                bestScore ← score;
                bestPacking ← packing;
            end if
        end for
        packed ← packed ∪ bestPacking;
        toPack ← toPack \ bestPacking;
    until toPack = ∅ ∨ bestPacking = null;
    return packed;
end function


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