Question 1

(a): This is a special instance of the knapsack problem where every item has an equal value of 1. Therefore, given the value we gain out of picking any item in the list is the same, we can focus solely on the sizes. That is, by being greedy and short-sighted and picking items just based on their size, we are not at risk of losing out on the global optimum value since each item contributes equally in value.

(b): Counterexample: eg. $S = 4$ and files $\{1, 2, 4\}$.

Question 2

(a):

```
function CheckSubSet($X$,$Y$)
    $X\_sorted = \text{Mergesort}(X)$
    $Y\_sorted = \text{Mergesort}(Y)$
    $j = 0$
    for $i$ from 0 to $length(X)$ do
        while $j < length(Y)$ and $X\_sorted[i] > Y\_sorted[j]$ do
            $j = j + 1$
        end while
        if $X_{sorted}[i] \neq Y_{sorted}[j]$ or $j \geq length(Y)$ then
            return False
        end if
    end for
    return True
end function
```

(b):

```
function CheckSubSet($X$,$Y$)
    Initialise $Y\_hash$ as a Hash Table.
    for $j$ from 0 to $length(Y) - 1$ do
        $Y\_hash$.insert($Y[j]$)
    end for
    for $i$ from 0 to $length(X) - 1$ do
        if $Y\_hash$.search($X[i]$)== $False$ then
            return False
        end if
    end for
    return True
end function
```

Question 3

```
function SOLUTION($T$)
    if $T == NULL$ then
        return -Inf, 1, 1
    end if

    $left\_global\_max, left\_max, left\_min = SOLUTION(T.left)$
    $right\_global\_max, right\_max, right\_min = SOLUTION(T.right)$

    // path_max: max product from the current node going downwards
    // path_min: min product from the current node going downwards
    $path\_max = max(T.value \times left\_max, T.value \times right\_max,$
        $T.value \times left\_min, T.value \times right\_min, T.value)$
    $path\_min = min(T.value \times left\_max, T.value \times right\_max,$
        $T.value \times left\_min, T.value \times right\_min, T.value)$

    // global_max: max product of ”current node as root” tree
    // path_max: max product from the current node going downwards
    // left_global_max: max product from left sub-tree (this is necessary if T.value=0)
    // right_global_max:max product from right sub-tree (this is necessary if T.value=0)
    $global\_max = max(path\_max,$
        $left\_global\_max,$
        $right\_global\_max,$
        $left\_max \times right\_max \times T.value,$
        $left\_max \times right\_min \times T.value,$
        $left\_min \times right\_max \times T.value,$
        $left\_min \times right\_min \times T.value)$

    return $global\_max, path\_max, path\_min$
end function

function FINDMAXPRODUCT($T$)
    $result, a, b = SOLUTION(T)$
    return $result$
end function
```

Question 4

(a):

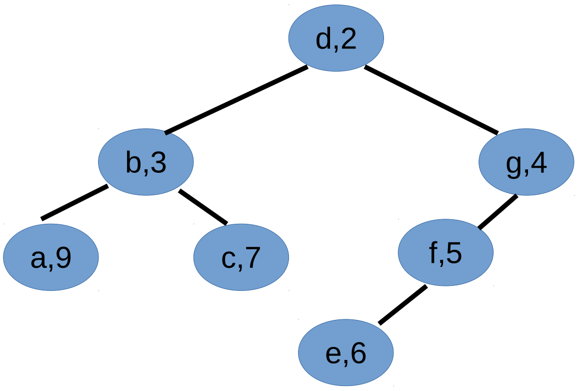

Figure 1: Treap from given set

(b):

```
function BuildTreap($R$)
    if $R$ is empty then
        return $NULL$
    end if

    $r\_minp$ = record in R with minimum priority.
    Initalise empty set of records $R\_left$ and $R\_right$
    Initalise empty tree $T$

    for each $r$ in $R$ do
        if $r.key < r\_minp.key$ then
            Include $r$ in $R\_left$
        else
            Include $r$ in $R\_right$
        end if
    end for

    $T.node = r\_minp$
    $T.left$ = BuildTreap($R\_left$)
    $T.right$ = BuildTreap($R\_right$)

    return $T$
end function
```

Pseudo-code option 2

```
function Rotation($T$,$A$)
    while $T.root! = A$ and $A.parent.priority > A.priority$ do
        if $A.parent.left == A$ then
            // With proper explanations (OR) citation
            rightRotation($T$, $A$)
        else
            // With proper explanations (OR) citation
            leftRotation($T$, $A$)
        end if
    end while
end function

function BSTInsert($T$, $A$)
    $insert = False$
    if $T.root == NULL$ then
        $T.root = A$
        $A.root = NULL$
        $insert = True$
    end if
    $ParentNode = T.root$
    while $insert == False$ do
        if $ParentNode.key > A.key$ then
            if $ParentNode.left == NULL$ then
                $ParentNode.left == A$
                $A.Parent = ParentNode$
                $insert = True$
            else
                $ParentNode = ParentNode.left$
            end if
        else
            if $ParentNode.right == NULL$ then
                $ParentNode.right == A$
                $A.Parent = ParentNode$
                $insert = True$
            else
                $ParentNode = ParentNode.right$
            end if
        end if
    end while
end function
function BuildTreap($R$)
    Initialise empty tree $T$
    for each item $A$ in $R$ do
        BSTInsert($T$,$A$)
        Rotation($T$,$A$)
    end for
end function
```

Pseudo-code option 3

Presort the set of records according to priority and insert them in sorted order.