May 2018. Algorithms. Qualifying exam Solutions.

Closed books, closed notes.

1. (30 points) You are given two arrays of size n each. Elements of the arrays are numbers. Each array may contain duplicate numbers in it. Design an algorithm that returns true if the arrays are disjoint, i.e. have no elements in common. Your algorithm should take O(n log n) time. Write down your algorithm as pseudocode and argue that its running time is indeed O(n log n). You do not need to write Java code, but be precise – a competent programmer should be able to take your description and easily implement it. You may freely use standard data structures and algorithms from the Algorithms course in your solution, without explaining how they are implemented.

Solution 1:

Suppose the two arrays are called array1 and array2. Indices of elements in the arrays range from 0 to n-1. The idea is to insert all elements of array1 into an (e.g.) red-black tree and then check if any element of array2 is in the tree.

Pseudocode:

```
S = new red-black tree
for i=0 to n-1 do
    S.insert(array1[i])
for i=0 to n-1 do
    if (S.search(array2[i])returns true) then return false
return true
```

Running time: Operations insert and search have O(log n) running time for a red-black tree with n nodes.

Solution 2:

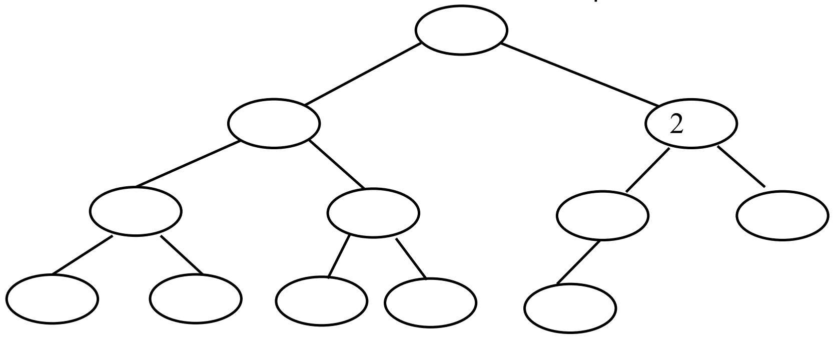
Suppose the two arrays are called array1 and array2. Indices of elements in the arrays range from 0 to n-1. The idea is to sort both arrays and then compare elements of the arrays in increasing order.

Pseudocode:

```
Mergesort(array1)
Mergesort(array2)
i=0 // index in array1
j=0 // index in array2
while (i<n && j<n)
    if (array1[i]==array2[j])
        return false
    else
        if (array1[i]<array2[j])
            i=i+1
        else
            j=j+1
return true
```

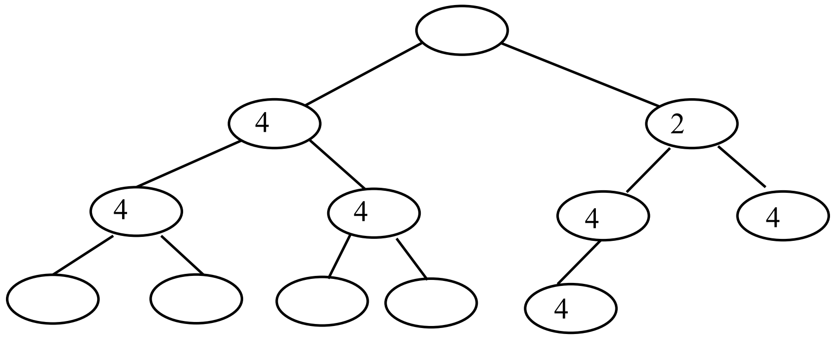
Running time: Mergesort takes O(n log n) time. While loop takes O(n) time. Total is O(n log n).

2. (10 points) You are given the structure of a heap (min-heap) as sketched below, where the second-smallest value in the set is marked with a 2. Mark a 4 for each node that can possibly contain the fourth-smallest value in the set. Assume that there are no duplicate node values.



Solution:

Nodes marked with 4 are the answer.



Any of the nodes shown below the 2 could be the 4th smallest element. If the left child of the root is the 3rd smallest element, then either of its children, but not its grandchildren, could be the 4th smallest. If the 3rd smallest is under the 2, then the left child of the root could also be the 4th smallest.

3. (15 points) Explain, in words, how to use Kruskal's algorithm to compute the number of connected components in an undirected graph.

Solution:

In a graph with c connected components, the spanning trees for the components have a total of n − c edges, where n is the total number of vertices in the graph. Therefore, if we execute Kruskal's algorithm we will always end up with exactly k = n − c edges, so c = n − k.

4. Given a list of n integers, $v_1, \dots, v_n$, the product-sum is the **largest** sum that can be formed by multiplying some of the adjacent elements in the list and adding the products together. Each element can be multiplied with at most one of its neighbors. For example, given the list 1, 2, 3, 1 the product sum is 8 = 1 + (2 × 3) + 1, and given the list 2, 2, 1, 3, 2, 1, 2, 2, 1, 2 the product sum is 19 = (2 × 2) + 1 + (3 × 2) + 1 + (2 × 2) + 1 + 2.

a) (5 points) Compute the product-sum of 1, 4, 3, 2, 3, 4, 2.

Solution: 29 = 1 + (4 × 3) + 2 + (3 × 4) + 2.

b) (25 points) Give an optimal dynamic programming algorithm for computing the product-sum of a list of n integers, $v_1, \dots, v_n$, and analyze its running time.

Solution:

Subproblems: $PS(i)$ is the product-sum of the first i integers on the list, 0≤i≤n.

Recursive solution:

$$PS(i) = \begin{cases} \max\{PS(i-1) + v_i,\ PS(i-2) + v_{i-1} \cdot v_i\} & if\ i \geq 2 \\ v_1 & if\ i = 1 \\ 0 & if\ i = 0 \end{cases}$$

Dynamic programming algorithm (pseudocode):

```
Prod-Sum(int[ ]v, n)
if (n == 0)
    return 0;
int [ ] PS = new int [n + 1];
PS[0] = 0;
PS[1] = v[1];
for int j = 2 to n
    PS[j] = max(PS[j − 1] + v[j], PS[j − 2] + v[j-1] ∗ v[j]);
return PS[n];
```

Running time:

There are O(n) iterations of the for loop. Each iteration takes constant time. Therefore, the running time of the algorithm is O(n).

c) (15 points) What do you need to add to your algorithm to determine not just the product-sum, but also which pairs are multiplied together.

Solution:

At each iteration of the loop we need to keep track how the max value was obtained (by adding v[j] to PS[j − 1] or by adding v[j-1] ∗ v[j] to PS[j − 2]. We could use array op of characters for that. In the for loop we can set op[j]='+' if PS[j − 1] + v[j] is the max value and op[j]='*' if PS[j − 1] + v[j-1] ∗ v[j] is the max value.

After the loop we can print an optimal product-sum expression using the following recursive procedure:

```
Print_Product_Sum (int[]v, int[] op, i)
if (i==0)
        return;
if (i==1)
        print v[1];
else  // if  i >=2
        if (op[i]==’+’)
                Print_Product_Sum(v, op, i-1);
                print “+” + v[i];
        else
                if (i==2)
                        print  v[i-1] + “*” + v[i]];
                else
                        Print_Product_Sum(v, op, i-2);
                        print “+” + v[i-1] + “*” + v[i]];
```