Qual Exam (Fall 2014) Algorithms
Answer all questions. Closed book exam.

Question 1 (Divide-and-Conquer Algorithm) (25%)

You are given two sorted lists of size m and n, where $n \leq m$. We want to design an $O(\log m + \log n)$-time algorithm for computing the kth smallest element in the union of the two lists. You are asked to complete the codes given in the following template:

```
// Let the two sorted lists be A[1..n] and B[1..m].
// Without loss of generality, we assume that n <= m.
// Also, we assume 1 <= k <= n+m.

int kth_smallest( int[] A, int[] B, int k ) {
    if (k <= n && A[k] <= B[1]) return A[k]; // special case
    if (k <= m && B[k] <= A[1]) return B[k]; // special case
    i = 1;
    j = min(k-1,n);    // j is the minimum of k-1 and n

    // Loop invariant:
    // The k smallest data are located in A[1..p] and B[1..k-p],
    //                              for some p where i <= p <= j
    while (i <= j) {
       mid = (i+j)/2;
       // to be completed









     }
}
```

Justify that the running time is $O(\log m + \log n)$.

Answer:

The missing codes are:

```
if      (A[mid] > B[k-mid+1]) j = mid-1; // A[mid] is not among the k smallest data
else if (B[k-mid] > A[mid+1]) i = mid+1; // B[k-mid] is not among the k smallest data
else return max(A[mid],B[k-mid]);        // the k smallest data are in A[1..mid], B[1..k-mid]
```

The algorithm runs in time $O(\lg \min(k-1,n)) = O(\lg \min(k,n)) = O(\lg n)$. Since it is assumed that $n \leq m$, the algorithm runs in time $O(\min(\lg n, \lg m)) = O(\lg n + \lg m)$.

Question 2 (Minimum Spanning Tree and Dijkstra's Algorithm) (25%)

The following statements may or may not be correct. In each case, either prove it (if it is correct) or give a counterexample (if it isn't correct). Always assume that the graph $G = (V, E)$ is undirected and connected. Do not assume that edge weights are distinct.

(a) Let C be a cycle in a graph G. Suppose the cycle C has a unique lightest edge e. Then e must be part of every minimum tree spanning the graph G.

Answer: Counterexample: $\{(v_1, v_2, 1), (v_1, v_3, 1), (v_2, v_3, 2), (v_2, v_4, 3), (v_3, v_4, 3)\}$. Edge (v_2, v_3) is the lightest edge in the cycle $(v_2, v_3), (v_2, v_4), (v_3, v_4)$, but MST consisting $(v_1, v_2), (v_1, v_3), (v_2, v_4)$ does not contain (v_2, v_3).

(b) The shortest-path tree computed by Dijkstra's algorithm is necessarily a minimum spanning tree.

Answer: Counterexample: $\{(s, v_1, 2), (s, v_2, 2), (v_1, v_2, 1)\}$. The tree computed by Dijkstra from s consisting of edges (s, v_1) and (s, v_2) is not a MST as any MST must include (v_1, v_2).

(c) The shortest path between two nodes is necessarily part of some minimum spanning tree.

Answer: Counterexample: $\{(v_1, v_2, 1), (v_2, v_3, 1), (v_3, v_4, 1), (v_1, v_4, 2)\}$. The shortest path from v_1 to v_4 is the single edge (v_1, v_4). But this edge (v_1, v_4) is not part of the unique MST $(v_1, v_2), (v_2, v_3), (v_3, v_4)$.

Question 3 (Greedy Algorithm) (25%)

Give a linear-time algorithm that takes as input a tree and determines whether it has a perfect matching: a set of edges that touches each node exactly once. You can assume that the tree is represented hierachically with r at the root. To access the children of a node n, you can write:

`for each child` x `of` n

Hint: Design a subprogram `matching(`n`)` that returns 1 if there is a perfect matching for the subtree rooted at node n, returns 0 if every subtree rooted at some child of n has a perfect matching while n is unmatched, and returns -1 otherwise.

Answer:

```
// determine if the subtree rooted at n has a perfect matching
// return  1  if there is a perfect matching
// return  0  each child subtree of n has a perfect matching; n is unmatched.
// return -1  otherwise
int matching(n)
   if n is leaf then return 0
   n_is_matched = false
   for each child x of n:
       case matching(x):
        -1: return -1
         0: if n_is_matched then return -1
            else n_is_matched = true
         1: // do nothing
   if n_is_matched then return 1
   else                 return 0
```

There is a perfect matching for the given tree if `matching(`r`)` returns 1.

Question 4 (Dynamic Programming) (25%)

Give an $O(nt)$-time algorithm for the following task.

> Input: A set of n distinct positive integers $\{a_1, a_2, \dots, a_n\}$,
> a positive integer t.
>
> (The set is in fact presented as a list $[a_1, a_2, \dots a_n]$.)
>
> Question: Does some subset of the a_i's add up to t?
>
> Important Note: the subset which sum is t is allowed to be a multiset in which the same a_i can appear more than once with no limit to how many times that the same number can be selected in the multiset.

Argue that your algorithm runs in $O(nt)$.

Answer:

Let $Q(i, x)$ be true if there is a subset of $\{a_1, a_2, \dots, a_i\}$ which sum is x, and false otherwise.

Base cases: $Q(0, 0) = \text{true}$; $Q(0, x) = \text{false}$ if x is not 0

Recursive formula:
$Q(i, x) = Q(i - 1, x) \vee Q(i, x - a_i)$ if $x \geq a_i$
$Q(i, x) = Q(i - 1, x)$ if $x < a_i$

The final answer is given in $Q(n, t)$.

Each subproblem $Q(i, x)$ is computed in $O(1)$ time. We need to compute $Q(i, x)$ for $0 \leq i \leq n$ and $0 \leq x \leq t$ row-wise in order of increasing i. Total time is $O(nt)$ as there are $O(nt)$ subproblems.