CSE 373 – Data Structures and Algorithms
Autumn 2003.
Final Exam. 12/17/2003

Student name	Student number

Question	
1	/15
2	/16
3	/22
4	/ 9
5	/20
6	/18
Total	/100

Question 1 (15 points)

1.a Only one of the following three arrays represents a minimum binary-heap. Circle its number.

[✓ **1.**]

-∞	3	5	18	20	7	19	40	24	32

2.

-∞	3	5	18	20	19	7	40	24	32

3.

-∞	3	5	7	40	18	19	20	24	32

1.b Consider the (only) minimum binary-heap from above. Show the content of the array after delete_min is performed.

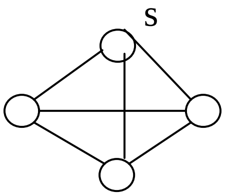
-∞	5	7	18	20	32	19	40	24

1.c Show the content of the array after insert(1) is performed on the resulting heap (after delete_min)

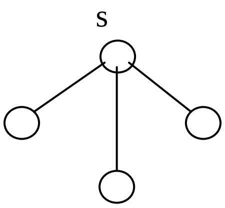
-∞	1	5	18	7	32	19	40	24	20

Question 2 (16 points)

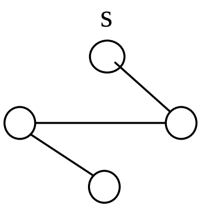
Assume that we perform DFS(s) on the following graph, in order to find a spanning tree. That is, a depth first search is performed starting from the vertex s and any edge that is the first to reach a non-marked vertex is added to the tree.



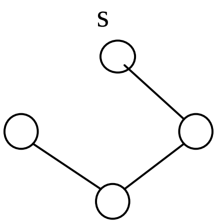
2.1. For each of the following graphs, circle *yes* if this is a possible output, or circle *no* if this is not a possible output of the DFS(s) execution.



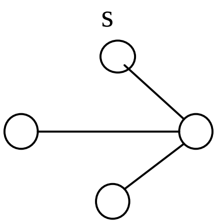
No



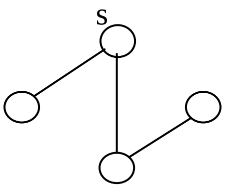
Yes



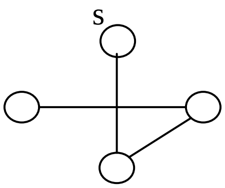
Yes



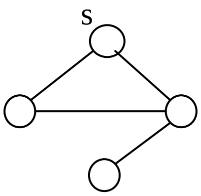
No



No

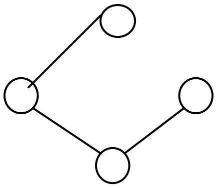


Yes



No

2.2. Draw one additional spanning tree that can be the output of DFS(s) and does not appear above.



Question 3 (22 points)

Given an array of *n* integers, and an additional integer *s*, your goal is to determine if there is a pair of two elements in the array, *a[i]* and *a[j]*, such that $i \neq j$ and $a[i]+a[j]=s$.

For example, if *A= {4, 8, -3, 2, 15, 7}* and *s=9* then the output should be *'yes'* (since 2+7=9). If *A= {4, 8, -3, 2, 15, 7}* and *s=8* then the output should be *'no'* (since there is no pair of two different elements in the array whose sum is *8*).

Rules: 1. You are not allowed to change the array in order to answer.
2. If the answer is *'yes'* then there is no need to report *a[i]* and *a[j]* nor the indices *i* and *j*, only to determine the positive answer.

For each of the following complexity constraints, describe your algorithm (in words or pseudo-code, whatever you prefer), and justify its time and space complexity. You can assume that the elements are indexed *1* to *n*, i.e., *a[1]* is the leftmost element and *a[n]* is the rightmost element.

a. Give an algorithm that uses *O(1)* additional space and finds an answer in time $O(n^2)$.

Check all pairs:

```
for i=1 to n-1
    for j = i+1 to n
        If (a[i]+a[j] =s) return 'yes'
Return 'no'
```

Every pair is checked, the number of iterations is (n-1)+(n-2)+...+1 = $O(n^2)$. Each iteration is O(1). The space comp. is O(1) since only two variables (i,j) are needed.

b. Give an algorithm, based on *hashing* that uses *O(n)* additional space and whose average time complexity is *O(n)* [assuming that each insert/find hashing operation takes on average *O(1)* time].

```
Allocate a hash-table of size m where m=O(n).
Select a hash-function and a collision-handle strategy (any can do).
for i = 1 to n do
    search for (s-a[i]) in the hash-table
    if find then return 'yes'
            else insrt a[i] to the hash table.
return 'no'
```

Time complexity: Each element is inserted once, and causes one 'seach' operation. So the total number of hash-operations is at most 2n, each takes on average O(1), which gives atotal of O(n).
Space complexity: O(n) – a reasonable size for the hash-table is about 2n.

Note: If you first insert all the elements into a hash-table, and then go through the array and search for (s-a[i]) (for each i), you have to pay attention not to say ‘yes’ if s=2*a[i]. In this case you have to search for two elements with value a[i].

<u>Question 4 (9 points)</u>

Circle the correct answer. No need to justify your choice (3 points each).

1. After an insert operation to an AVL tree

   a. There is at least one rotation
   [✓ b.] It might be that no rotation is needed
   c. There are d rotations, where d is the depth of the new element.

2. A binomial queue consisting of 17 elements is **merged** with a binomial queue with 15 elements.

   [✓ a.] In the resulting binomial queue there is one tree of size 32.
   b. In the resulting binomial queue there are two trees each of size 16.
   c. (a) or (b) depending on the values of the keys in the binomial queues.

3. 20 elements need to be stored in a hash-table, using open-addressing and linear probing. Among the possibilities below, the most suitable table size is

   a. 20
   [✓ b.] 37 - must be a prime!
   c. 40

Question 5 (20 points)

Consider a tree in which each node has at most three children. Each node has the following structure:

```
struct node {
          label       int;
          left        struct node pointer;
          middle    struct node pointer;
          right       struct node pointer;
};
```

For nodes with less than three children the appropriate fields are NULL. The initial values of the `label’ fields are unknown.

Complete the pseudocode of the **recursive** function *fill_label(k, root),* that gets as input an integer *k* and a pointer to a tree root, and fills the `label’ fields of all the tree nodes, such that for each node, *v*, the value of *v.label* fulfills:

$$v.label = \begin{cases} k & \text{if v is the tree root} \\ 1+ u.label & \text{if v is a left child of u} \\ 2+ u.label & \text{if v is a middle child of u} \\ 3+ u.label & \text{if v is a right child of u} \end{cases}$$

```
fill_label( k integer, v struct node pointer):void{

          if (v=null) return;
          v.label=k;
          fill_label(k+1, v.left);
          fill_label(k+2, v.middle);
          fill_label(k+3, v.right);

}
```

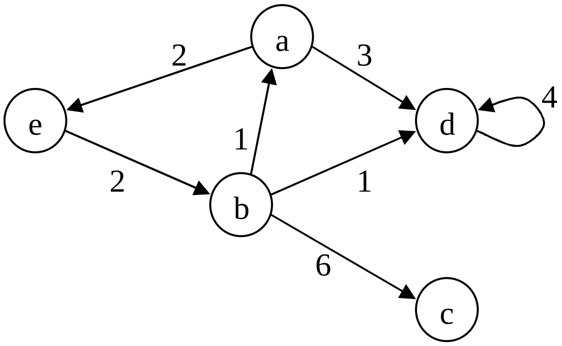
Question 6 (18 points)

Consider a weighted directed graph $G=(V,E)$ in which each edge has an integral weight between 1 and 6. The graph is given as a list of triplets of the form (v,u,w). The triplet (v,u,w) corresponds to a single edge from the vertex v to the vertex u, whose weight is w.

The goal is to sort the vertices according to the total sum of outgoing-edges weight. In other words, for each vertex v, let *out_weight(v)* be $\Sigma_{(v,u)\in E}\ w(v,u)$, then the goal is to print the vertices in the order $v_1,v_2,\ldots v_n$ such that

$$out\_weight(v_1) \leq out\_weight(v_2) \leq \ldots \leq out\_weight(v_n).$$

For example, for the following graph the required output is *c,e,d,a,b*



Suggest an algorithm for this sorting problem. The time complexity of your algorithm should be $O(n+m)$, where n is the number of vertices, and m is the number of edges in the graph. Draw a figure and describe in words the data-structures you are using. Describe the steps of your algorithm, and justify its time complexity. No need to write pseudo-code.

Solution:

1. Allocate an array to store the out_weight values, initialize it to 0.
2. Read the input and sum into out_weight[v] the weights of the outgoing edges of v. (each edge (v,u,w,) in the input implies out_weight[v]=out_weight[v]+w.)
3. Allocate an array 'sort' of size 6m, initialize it to 0.
4. Go through the out_weight array, and use bucket-sort into the array 'sort' to sort the vertices according to their out_weight values.

Time complexity: Init the out_weight array: O(n). Reading the input and updating the out_weight array: O(m). Bucket sort: O(n+m). (No vertex has out_weight larger than 6m, so it is OK that the length of the array 'sort' is 6m. This bound is 6n if no parallel edges are allowed).
All together: O(n+m)

Space complexity:
O(n) for the out_weight array and O(m) for the 'sort' array.

Remark: It is possible to use adjacency lists but there is no need to do it