Data Structure and Algorithm, Spring 2013

Midterm Examination

120 points

Time: 2:20pm-5:20pm (180 minutes), Tuesday, April 16, 2013

Problem 1. In each of the following question, please specify if the statement is **true** or **false**. For questions other than 1-5: If the statement is true, explain why it is true. If it is false, explain what the correct answer is and why. (14 points. 1 point for true/false and 2 points for the explanation for each question)

1. $n^2 = O(n^2)$ (1 point)
2. $n^2 = o(n^2)$ (1 point)
3. $n^2 = \omega(n^2)$ (1 point)
4. $n^2 = \Omega(n^2)$ (1 point)
5. $n^2 = \Theta(n^2)$ (1 point)
6. $f(n) = O(g(n))$ implies $2^{f(n)} = O(2^{g(n)})$.
7. Suppose that we use an array to store the elements in a max heap. a[0] stores the value in the root of the heap, a[1] and a[2] store the values of the child nodes of the root, and so on. With this representation, if the content of the array is sorted in descending order, then it is always a max heap.
8. To evaluate an expression tree, the most straight forward way is to use inorder traversal for the tree.

Problem 2. "Short answer" questions: (34 points)

1. $f(n) = \omega(g(n)) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} =$ _____. (2 points)
2. Explain in what scenario using an algorithm that has an asymptotically lower time complexity might **NOT** be the best choice, even if our only objective is to choose an algorithm that has the shortest running time. (4 points)

3. Please complete Table 1 to show the progress of converting the infix expression "1+(2/3*(4+5)-7)" to its postfix expression using a stack. (6 points)

4. Figure 1 shows two singly linked lists are merged at node x and become one singly linked list afterwards. Explain how you can use two stacks, which only support pop and push operations but not randomly access an element in the stacks, to find the merged node x in $O(M + N)$ time where M and N are the lengths of the two lists including the lengths after the merge node. Note that the elements stored in the list nodes are not distinct. (Hint: the stack should store pointers) (6 points)

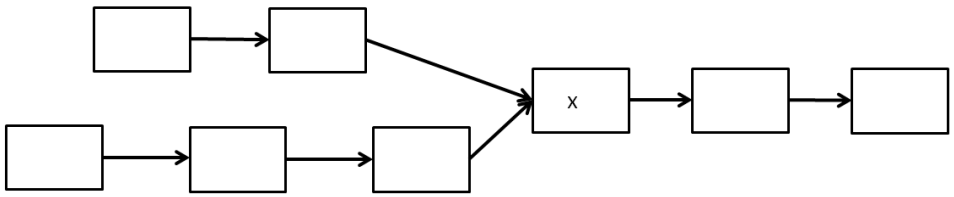

Figure 1: Merged Lists

5. What are the space complexities of the adjacency matrix and the adjacency lists of the graph $G = (V, E)$? Please express it with the Θ-notation in terms of $|V|$ and $|E|$. (4 points)

6. Give the lists of vertices in the order they are discovered when performing (1) DFS and (2) BFS on the graph in Figure 2 starting from vertex A. In the search process, if there are multiple edges to be travelled through, they should be processed in alphabetical order according to the vertex ID of the vertex on the other side of the edge. For example, when at vertex A, edge $< A, B >$ should be processed before edge $< A, D >$. For DFS, your algorithm should check if all vertices are visited also according to the vertex ID in alphabetical order. Note that BFS will only discover all vertices that are connected to the starting vertex while DFS will discover all vertices in the graph. (8 points)

7. Give two advantages of using version control software. (4 points)

Problem 3. Please answer the following questions about trees: (24 points)

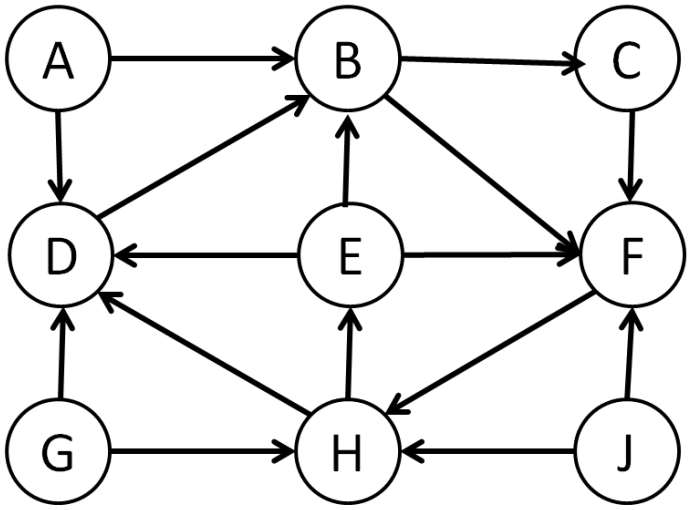

Figure 2: A Graph

1. Please draw how an initially empty binary search tree would look like after the following numbers are inserted in the given order: {15,3,7,4,18,6,5,8,9,17,1}. No need to show it in a step-by-step fashion; you only need to draw the final result. (4 points)

2. If we delete the element with key '7' from the binary search tree that you get from question 1, how would the tree look like? Please draw both possible outcomes. (6 points)

3. We would like to create an inorder threaded binary search tree for the tree from question 1. Please draw this threaded tree. Tree edges should be drawn with solid lines and threads should be drawn with dashed lines. Please also remember to draw the dummy node. (6 points)

4. Implement a C function to find the **preorder** successor of a given node in the threaded tree. You can use the prototype below and assume that each tree node in the threaded tree uses the structure in the following. (8 points)

```
struct threaded_tree_node{
   struct threaded_tree_node *left, *right;
   int left_thread, right_thread;
```

```
      //indicate whether the pointer represents to a thread (=1)
      //or a tree edge (=0)
   int key;
};

struct threaded_tree_node * preorder_successor(struct threaded_tree_node *);
// this is the function to be implemented
```

Problem 4. Please answer the following questions about heap: (35 points)

1. A max heap is a max complete binary tree. Explain what a max tree is and what a complete binary tree is. (6 points)

2. What is the height of the heap with n elements? Express it in terms of n. (Here we define the height of the heap to be the number of levels in this heap) (2 points)

3. Assume that we use the array representation to store the heap as shown in the lecture (explain how you represent a heap in the array if you use a different approach). Please complete the following C function to implement a new operation that can delete an arbitrary element in the heap (not just the root). The index of the element to be deleted in the array is given as the parameter of the function. Make sure your implementation runs in $O(\log n)$. (8 points)

```
int heap_size;
   // this global variable stores the number of elements in the heap
int heap[MAX_NUMBER_ELEMENTS];
   //this global array stores the elements in the heap

void delete_maxheap(int index) {
   // index is the index of the element to be deleted in the array
   int key;

   if (index >= heap_size) {
```

```
      printf("error!");
      return;
   }
   h[index]=h[heap_size-1];
   //moving the last element in the heap to where
   //the element to be deleted is stored
   --heap_size;

   /* insert your code here to adjust the elements in the heap*/

}
```

4. Show that your implementation of delete_maxheap() runs in $O(\log n)$. (4 points)

5. We want to implement a new data structure called the *min-max heap*. This data structure allows the operations of inserting an element, deleting the minimum element, and deleting the maximum element to be completed all in $O(\log n)$ time. The data structure can be implemented with two heaps - a max heap and a min heap. Each element in the array uses the following structure:

```
struct heap_element {
   int key;
   int index_other;
   //this stores the index of the location this element is stored
   //in the other heap
};
```

And we use the following declarations for our min-max heap:

```
struct heap_element minheap[MAX_NUMBER_ELEMENTS];
struct heap_element maxheap[MAX_NUMBER_ELEMENTS];
int heap_size;
```

Add new C code to your original delete_maxheap() function so that when the locations of the elements are adjusted in the max heap, it will also change the index_other value in the same element in the min heap to reflect the change. Mark all the additions to the original functions. (6 points)

6. In addition to delete_maxheap() function that you implemented, assume that we have the following functions implemented. You can assume that in those functions all element location adjustments are reflected in the other heap, similar to your delete_maxheap() function.

```
void delete_minheap(int index);
  //This function deletes an arbitrary element in minheap
int insert_maxheap(int key);
  //This function inserts an element in maxheap
  //return value: the index of the inserted element in the array of maxheap
int insert_minheap(int key);
  //This function inserts an element in minheap
  //return value: the index of the inserted element in the array of minheap
struct heap_element *deletemin_minheap();
  //This function deletes and returns the minimum element in minheap
  //return value: the pointer that points to the deleted node
struct heap_element *deletemax_maxheap();
  //This function deletes and returns the maximum element in maxheap
  //return value: the pointer that points to the deleted node
```

Please implement the following three C functions for the min-max heap.

```
void insert_minmaxheap(int key);
  // insert an element to the min-max heap
int deletemin_minmaxheap();
  //delete and return the minimum element to the min-max heap
int deletemax_minmaxheap();
```

```
//delete and return the maximum element to the min-max heap
```

You are allowed to call the functions listed in the declarations. Make sure your functions run in $O(\log n)$. (9 points)

Problem 5. Derive an algorithm to determine whether a directed graph $G = (V, E)$ contains a universal sink - a vertex with in-degree $|V| - 1$ and out-degree 0, given an adjacency matrix of G. Note that there is no self edge and this is not a multi-graph. Write down the pseudo code or the C code and analyze the running time of your algorithm. (Hint: A[i][j] means there is an edge $< i, j >$. Then A[i][j]=1 means vertex i cannot be a universal sink, and A[i][j]=0 means vertex j cannot be a universal sink.) (8 points if your algorithm runs in $O(|V|)$ and 4 points if your algorithm runs in $O(|V|^2)$)

Problem 6. Please write down two things you like/dislike about this course so far, and how you would change the things you do not like if you were the professor to teach DSA. General comments about the course are also welcome. :) (5 points)

Please write your ID and name here:

Table 1: The progress of converting the infix expression "1+(2/3*(4+5)-7)" to its postfix expression. **(You can take this page off and submit it with the rest of your answer sheets)**

Token	**Stack content after processing the token (right: stack top)**	**Output after processing the token**
1		1
+	+	1
(		
2		
/		
3		
*		
(		
4		
+		
5		
)		
-		
7		
)		
End of String		