Data Structure and Algorithm, Spring 2013

Midterm Examination

120 points

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

Problem 1. If the statement is true, explain why it is true. If it is false, explain what the correct answer is and why. (14 points.)

For questions 1-5, 1 point for true/false.

From questions 6-8, 1 point for true/false and 2 points for the explanation. However, -1 point for each negligible error and imprecise explanation.

1. *True.*

2. *False.*

3. *False.*

4. *True.*

5. *True.*

6. *False.*

   Assume that $f(n) = 2n$. From $f(n) = O(n)$, we know $g(n) = n$.

$$\lim_{n\to\infty} \frac{2^{f(n)}}{2^{g(n)}} = \lim_{n\to\infty} \frac{2^{2n}}{2^n} = \lim_{n\to\infty} 2^n = \infty$$
$$2^{f(n)} \notin O(2^{g(n)})$$

   (-1 point if no proof nor counter example.)

7. *True.*

   The definition of max heap:

   - it's complete binary search tree and

- each node is greater than or equal to each of its children.

If you're writing the full proof, You should prove it's complete binary tree first. Let us omit this part.

We already know that the content of the array is storted in descending order. For each two elements indexed i and j, $i < j$, imply that $a[i] \geq a[j]$.

In the array representation, the childs of node x are $2x + 1$ and $2x + 2$. We know $\forall x, x < 2x + 1 < 2x + 2$, then $a[x] \geq a[2x + 1] \geq a[2x + 2]$.

8. *False.*

   Postorder traversal is better and easily to implement.

   (-1 point if no talking about "post-order".)

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

1. (2 points)

   Infinity, by the definition of ω.

2. (4 points)

   An algorithm that have a better asymptotically lower time complexity means that its running time somehow become short when the problem size is 'big enough'. However, in practical use, the problem size may be small.

3. (6 points)

   See Table 1.

4. (6 points)

   If we search from the heads of two linked lists, they may not meet at the same time. On the other hand, we can try searching from their tails. First, put the pointer of each node in two linked lists from head to tail into two stack representatively. Using the last-in-first-out property of stack, we can retrieve these pointer from stack in the reversed order, i.e. from tail to head in the original linked list. Now we can pop

a pointer from each stack and compare them until the first difference appear. Then the previous node is what we want.

5. (6 points)

   The adjacency matrix always records $n \times n$ numbers regardless of the number of edges. Hence its space complexity is $\Theta(|N|^2)$. The adjacency lists have n lists, each for a node. Every item in the list represents an edge, and an edge is represented by one (or two) node(s). Hence total items in the nodes are $|E|$ (or $|2E|$) and the space complexity of adjacency lists is $\Theta(|N| + |E|)$.

6. (8 points)

   DFS: $ABCFHDEGJ$

   BFS: $ABDCFHE$

7. (4 points)

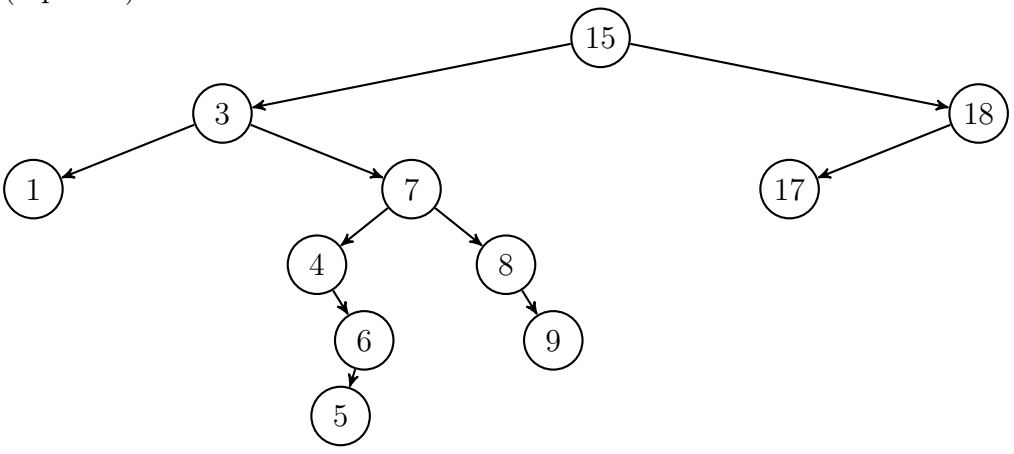
   Each advantage gets 2*pt*

   (a) We can develop project with many people at the same time with version control software.

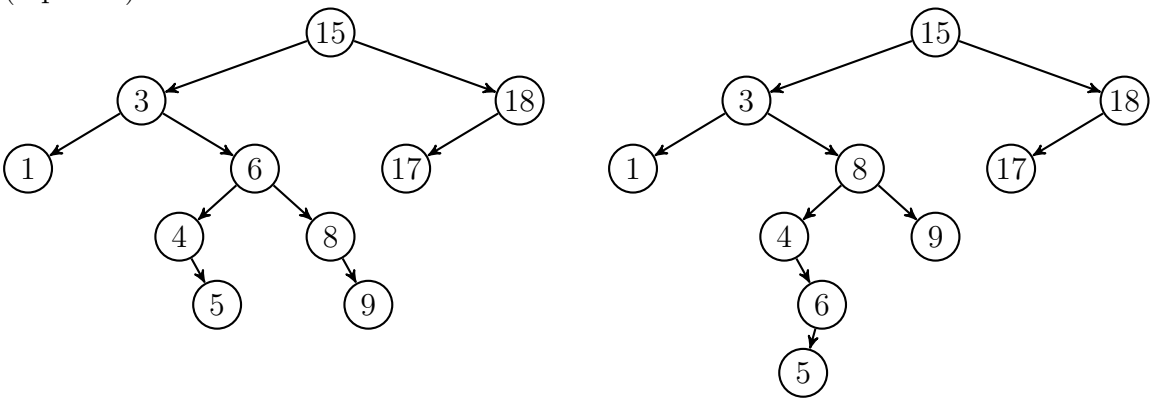
   (b) Version control software allows us to revert a document to a previous revision

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

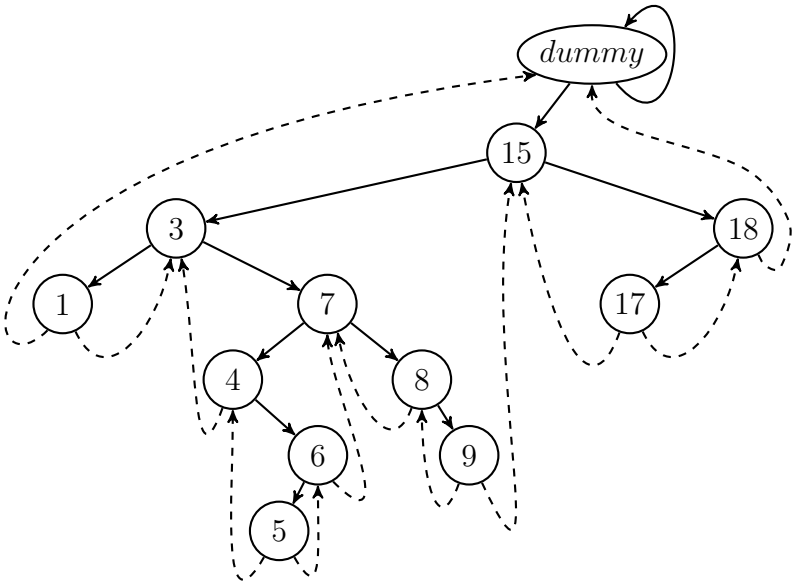
1. (4 points)



2. (6 points)



3. (6 points)



4. (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 *ptr){
    if(!ptr->left_thread) return ptr->left;
    while(ptr->right_thread) ptr=ptr->right;
    return ptr->right;
}
```

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

1. (6 points)

   Max tree(3 points)

   keywords: tree 1*pt, index no smaller* 2*pt*

   A max tree is a tree in which the key value in each node is no smaller than the key values in its children(if any)

   Complete Binary Tree(3 points)

   Lecture Edition keywords: full binary tree 1*pt, correspond* 2*pt*

   A binary tree with n nodes and depth k is complete iff its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k

   Wiki Edition keywords: filled 1*pt, as far left as possible* 2*pt*

   a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

2. (2 points)

   $\lfloor \log_2 n (+1) \rfloor$ or $\lceil \log_2 (n+1) \rceil$

   +1 is depend on whether to define root layer as 0 or 1.

3. (8 points)

   Check if no greater than parent node(3 points)

   keyline: check if root 1*pt, swap* 1*pt, change index* 1*pt*

```
while(index!= 0 && heap[(index-1)/2]<heap[index]){
   key = heap[index];
   heap[index] = heap[(index-1)/2];
   heap[(index-1)/2] = key;
   index = (index-1)/2;
}
```

   Check if no smaller than child nodes(4 points)

   keyline: check if leaves 1*pt, swap* 1*pt, compare* 1*pt,change index* 1*pt, check one child* 1*pt*

```
while(index*2+2 < heap_size && (heap[index] < heap[2*index+1] ||
      heap[index] < heap[2*index+2])){
   key = heap[index];
   if(heap[2*index+1]>heap[2*index]){
      heap[index] = heap[2*index+1];
      heap[2*index+1] = key;
      index = 2*index+1;
   }
   else{
      heap[index] = heap[2*index+2];
  heap[2*index+2] = key;
  index = 2*index+2;
   }
}
if(index *2+1 == heap_size-1 && heap[index] < heap[2*index+1]){
   //only one child
   key = heap[index];
   heap[index] = heap[2*index+1];
   heap[2*index+1] = key;
   index = 2*index+1;
}
```

4. (4 points)

   keywords: first while loop $O(\log n)$ *times* 1*pt, second loop* $O(\log n)$ *times* 1*pt, impossible to enter both* 2*pt*

   If entering the first while loop (i.e. greater than parent node), swap until no greater in height of the tree : $O(\log n)$.

   But the second loop would be skipped since every swapped node is greater than their child.

   If skipping the first loop, the second loop can be done in swap time * height of the tree : $O(\log n)$.

5. (6 points)

Check if the element is deleted and the properties of data structure are kept.(2 points)
Check if the values of index_other in minheap are maintained correctly. (4 points)
Each typo or errors costs 1*pt*

```
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;
    }
    maxheap[index]=maxheap[heap_size-1];
    minheap[ maxheap[index].index_other ].index_other = index;
    --heap_size;
    //moving the last element in the heap to where
    //the element to be deleted is stored

    while(index!= 0 && maxheap[(index-1)/2].key<maxheap[index].key){
        int swap_index = (index-1)/2;
        struct heap_element tmp = maxheap[index];
        maxheap[index] = maxheap[swap_index];
        maxheap[swap_index] = tmp;
        minheap[ maxheap[index].index_other ].index_other = index;
     minheap[ maxheap[swap_index].index_other ].index_other = swap_index;
        index = swap_index;
    }
   while( index*2+1 < heap_size && maxheap[index].key<maxheap[2*index+1].key ||
    index*2+2 < heap_size &&
```

```
(maxheap[index].key < maxheap[2*index+1].key || maxheap[index].key < maxheap[2*index
        int swap_index;
     if(index*2+1==heap_size-1 || maxheap[2*index+1].key>maxheap[2*index].key){
//only one child or the left child is larger
            swap_index = 2*index+1;
        }else{
//the right child is larger
            swap_index = 2*index;
        }
        struct heap_element tmp = maxheap[index];
        maxheap[index] = maxheap[swap_index];
        maxheap[swap_index] = tmp;
        minheap[ maxheap[index].index_other ].index_other = index;
     minheap[ maxheap[swap_index].index_other ].index_other = swap_index;
        index = swap_index;
    }
}
```

6. (9 points)

Function insert_minmaxheap (3 points)

Call two insert functions 2*pt, modify the index_other* 1*pt*

```
void insert_minmaxheap(int key){
   int maxheap_index = insert_maxheap(key);
   int minheap_index = insert_minheap(key);
   max_heap[maxheap_index].index_other = minheap_index;
   max_heap[maxheap_index].index_other = minheap_index;
}
```

Function deletemin_minmaxheap (3 points)

Delete the element in minheap 1*pt, delete the element in maxheap with index_other information* 1*pt, return the key value* 1*pt.*

```
int deletemin_minmaxheap(){
   struct heap_element *ptr;
   ptr = deletemin_minheap();
   delete_maxheap(ptr->index_other);
   return ptr->key;
}
```

Function deletemax_minmaxheap (3 points)

Delete the element in maxheap 1*pt, delete the element in minheap with index_other information* 1*pt, return the key value* 1*pt.*

```
int deletemax_minmaxheap(){
   struct heap_element *ptr;
   ptr = deletemax_maxheap();
   delete_minheap(ptr->index_other);
   return ptr->key;
}
```

Problem 5. $O(|V|)$ code is shown below.

```
// Assume that n = |V|, the number of vertices is from 0 to n-1.
// The function returns the number of universal sink or -1 if non-exist.
int findSink(int A[n][n]){
    int i, now = 0;
    for(i=1;i<n;i++)
        if(A[now][i] == 1)
            now = i;
    for(i=0;i<n;i++)
        if(i != now && (A[i][now] == 0 || A[now][i] == 1))
            now = -1;
    return now;
}
```

Table 1: The progress of converting the infix expression "1+(2/3*(4+5)-7)" to its postfix expression.

Token	**Stack content after processing the token (right: stack top)**	**Output after processing the token**
1		1
+	+	1
(	+(	1
2	+(	12
/	+(/	12
3	+(/	123
*	+(*	123/
(	+(*(	123/
4	+(*(	123/4
+	+(*(+	123/4
5	+(*(+	123/45
)	+(*	123/45+
-	+(-	123/45+*
7	+(-	123/45+*7
)	+	123/45+*7-
End of String		123/45+*7-+

TA List

Problem 1. **余孟桓**

Problem 2. **簡伯宇**

Problem 3. **吳哲仰**

Problem 4. **魏佑霖** (1 ∼ 4)、**張庭維**、**姜俊宇** (5 ∼ 6)

Problem 5. **林均達**