The University of Newcastle
Discipline of Computer Science and Software Engineering

COMP2230/6230 Algorithms

Tutorial Week 4

9^{th}-13^{th} August 2021

Tutorial

1. Write a function *invert(s)* that inverts the contents of a stack. For example, if stack *s* originally contained 1, 2 and 3, where 3 is the most recently added item, after *invert(s) s* contains 3, 2 and 1, where 1 is the most recently added item. Use the abstract data type stack (do not refer to *data* or *t*). You may use additional stacks.

2. Suppose that *start* is a reference to the first node in the linked list or, if the linked list is empty, *start* = null. Write an algorithm that is passed *start* and a value *val*. The algorithm adds a node at the beginning of the linked list whose data is *val* and returns *start*. What is the worst case time of your algorithm?

3. A priority queue is implemented using an array. An item is inserted by putting it at the end of the array. Show that, in the worst case, performing *n* insertions and *n* deletions in an initially empty priority queue takes time $\Theta(n^2)$.

4. Write a linear time algorithm that swaps the left and right children of each node in a binary tree.

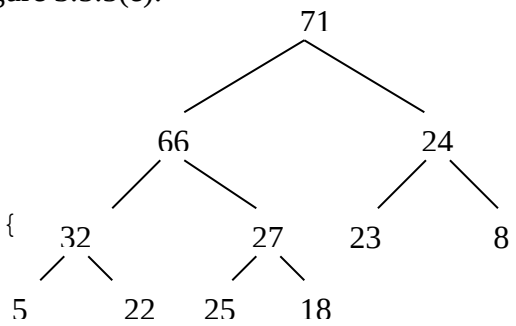
5. Write a non-recursive version of Algorithm 3.4.14 (inserting in a binary search tree) from the textbook.

```
BSTinsert(root,val){
     //set up node to be added to tree
     temp=new node
     temp.data=val
     temp.left=temp.right=null
     if(root==null) return temp
     BSTinsert_recurs(root,temp)
     Return root
}

BSTinsert_recurs(root,temp){
     If(temp.data ≤ root.data)
          If(root.left==null)
               Root.left=temp
          Else
               BSTinsert_recurs(root.left,temp)
     Else
          If(root.right==null)
```

```
                    Root.right=temp
               Else

          BSTinsert_recurs(root.right,temp)
     }
```

6. Trace insert when 61 is inserted in Figure 3.5.5(e).

```
Heap_insert(val,v,n){
     i=n=n+1
//i is the child i/2 parent
//if i >1, i is not the root
     while(i>1 && val >v[i/2]){
          v[i]=v[i/2]
          i=i/2
     }
     v[i]=val
}
```



7. Assuming the data in the heap are distinct, what are possible locations for:
   - The smallest element?
   - The second smallest element?
   - The second largest element?
8. Show that inserting n elements into an initially empty heap takes time $\Theta(n \log n)$ in the worst case.
9. Given the *key* array:

18	24	71	5	22	23	66	32	8	25	27

   Show the *into* and *outof* arrays when the heap in figure 3.5.5(e) of the text is implemented as an indirect heap

Homework

10. Write a version of *enqueue* that checks for full array. If the array is full, the functions simply returns false. If the array is not full, the behaviour is the same as original *enqueue*, except that the function also returns true.

11. Write an algorithm to reverse a linked list. You may modify only the *next* fields of the nodes. What is the worst case time of your algorithm?

12. Show how to implement a queue using priority queue.

13. Prove that postorder runs in time $\Theta(n)$ when the input is an n-node binary tree.

14. Write an algorithm that returns the number of terminal nodes (leaves) in a binary tree.

15. Write a linear time algorithm that determines whether a binary tree is a binary search tree.

16. Write an algorithm to search for a particular item in a binary search tree. The worst case time must be $\Theta(h)$ where h is the height of the tree. If the item is found, return a reference to the location. If it is not found, return null.

17. Write a constant time algorithm for returning the largest value in an indirect heap. Show that the algorithm runs in constant time.

Extra Questions

18. Write an algorithm to merge two linked lists. Assume that the data in each list are in nonincreasing order. The result is one linked list, containing all of the data, in nonincreasing order. You may modify only the next fields of the nodes. What is the worst case time of your algorithm?

19. A deque (pronounced “deck”) is like a queue, except that items may be added and deleted at the rear or the front. Implement a deque using an array, and implement it using a linked list. Do not incorporate error checking into your functions.

20. Show how to implement a stack using priority queue.

21. A *binary expression tree* is a binary tree in which the internal nodes represent binary operators and the terminal nodes represent values. An operator operates on its left and right subtrees. Write a postorder algorithm that prints the data in a binary expression. (i.e. The algorithm produces an arithmetic expression in *postfix* notation.)

22. If T is a binary tree where at each node v the datum in v's left child is less than or equal to the datum in v, and the datum in v's right child is greater than or equal to the datum at v, then T is a binary search tree. Is this statement true or false? Explain.

23. Write an $O(\log n)$ *siftup* algorithm for a heap. The input to *siftup* is an index i

and a heap structure in which the value of each node is greater than or equal to the values of its children (if any), except for the node at index *i* which may have a value greater than its parent. *siftup* is to restore the heap property. Prove the running time.

24. Write an algorithm that inserts a value into an indirect heap. This algorithm should run in logarithmic time. Show that it does.

25. Suppose that in heapsort (p. 145 in the text), the call to heapify (Algorithm 3.5.12) is replaced with repeated calls to insert (Algorithm 3.5.10) to create the initial heap. What is the worst case running time of this version of heapsort?

```
Heapsort(v,n){
      //make v into a heap
      heapify(v,n)
      for i=n downto 2{
            //v[1] is the largest among v[1],…,v[i]
            //put it in the correct cell
            swap(v[1],v[i]
            //heap is now at indexes 1 through i-1
            //restore heap
            siftdown(v,1,i-1)
      }}

      heapify(v,n){
            //n/2 is the index of the parent of the last node
            for i=n/2 downto 1
                  siftdown(v,i,n)
      }
```