— Sample —

CS 308 Data Structures

Fall 2000 - Dr. George Bebis

Final Exam

Duration: 12:00 - 2:00 pm

Name:

1. **True/False** (2 pts each) **To get credit, you must (very briefly) for your answers !!**

(1.1) **T** (**F**) The maximum number of nodes in a tree that has ~~N~~ L levels is 2^{L}

it is $2^L - 1$

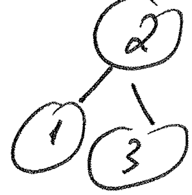
(1.2) (**T**) **F** A queue should be used when implementing Breadth First Search (BFS).

Using a queue, allows us to "remember" the most immediate neighbors

(1.3) **T** (**F**) The largest value of a binary search tree is always stored at the root of the tree.

this is true for heaps only!

counter example:



(1.4) **T** (**F**) A complete tree is also a full tree.

the leaf nodes of a complete tree are not at the same level

(1.5) **T** **F** In a binary tree, every node has exactly two children.

up to two ... it can have 0, 1, or 2 children

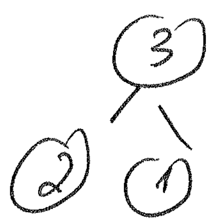
(1.6) **T** **F** Tree operations typically run in $O(d)$ time where d is the number of nodes in the tree.

d is the height of the tree, that is, $d = \lg N$ where N is the # of nodes

(1.7) **T** **F** To delete a dynamically allocated tree, the best traversal method is *postorder*

first, we delete the children, then the parent.

(1.8) **T** **F** In a heap, the left child of a node is always less than the right child of a node.



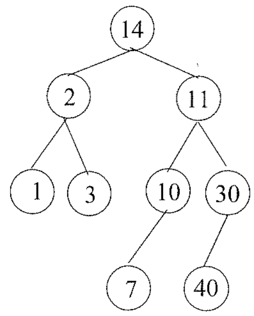
counter-example

(1.9) **T** **F** Implementing a priority queue using heaps is more efficient than using linked lists.

Enqueue / Dequeue take $O(\lg N)$ in the case of heaps

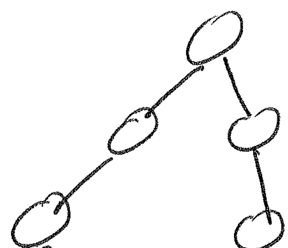
Enqueue → $O(N)$
Dequeue → $O(1)$
} in the case of lists ...

(1.10) **T** **F** The ancestors of node 10 are nodes 11, 2, and 14.



only 11 and 14

(1.11) **T** **F** Every binary tree is either complete or full.



neither full or complete

(1.12) **T** **F** Heaps are useful for searching binary trees efficiently.

no, we do not know which subtree to search ....

(1.13) **T** **F** A complete directed graph with 8 vertices has 64 edges.

8x7 = 56 edges

(1.14) **T** **F** The linked-list implementation of a graph is more efficient in finding whether two vertices are directly connected or not.

the array based implementation is more useful...

(1.15) **T** **F** The order in which elements are inserted in a binary search tree in unimportant.

It is very important: if the elements are inserted in sorted order, then the height of the tree will become $O(N)$!

2. **Short answers** (3 pts each)

(2.1) What is the number of nodes in a full tree with L levels ? Prove it (show all the steps carefully).

0, 1, …, L−1 } L levels

$$N = 2^0 + 2^1 + \dots + 2^{L-1} = \frac{2^L - 1}{2 - 1} = 2^L - 1$$

(2.2) What is the maximum number of levels (height) of a tree with *N* nodes ? What is the minimum number of levels (height) of a tree with *N* nodes ? Justify your answers.

N levels max

max height = N

min height = $\lg(N+1)$

tree is as compressed as possible... (full tree)

(2.3) Assume A is an array-based tree with 70 nodes. $N = 70$

What is the index of the first leaf node ? leaves : $N/2$ to $N-1$

$N/2 = 70/2 = \underline{35}$

Who is the parent of A[50] ?

$$A\left[\frac{50-1}{2}\right] = A[49/2] = A[24]$$

↑ integer division.

Who are the children of A[10] ? A[2*10+1] = A[21]
A[2*10+2] = A[22]

How many leaf nodes does the tree have ?

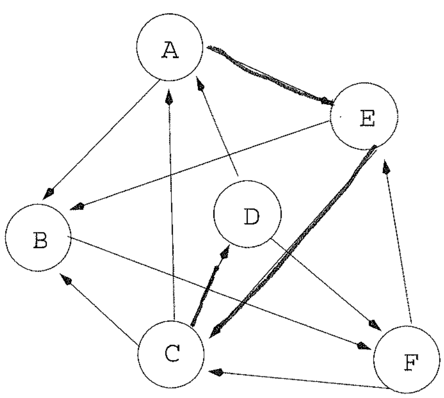
from N/2 to N-1 : 69-35+1 = 35

(2.4) We have discussed two different approaches to implement a priority queue. Which are these two approaches ? How do they compare in terms of efficiency (time wise) ? Justify your answer.

1st approach using heaps
Enqueue O(lgN)
Dequeue O(lgN) } O(lgN) on the average

2nd approach using linked lists
Enqueue : O(N)
Dequeue : O(1) } O(N) on the average

(2.5) Given the graph below, draw its adjacency matrix representation (store the vertices in alphabetical order)



0	A
1	B
2	C
3	D
4	E
5	F

	0	1	2	3	4	5
0		1			1	
1						1
2	1	1		1		
3	1					1
4		1	1			
5			1		1	

(2.6) Using the graph in (2.5), is there a path from A to D ? Demonstrate how Depth First Search (DFS) solves this problem (to get credit, you need to show all the steps clearly).

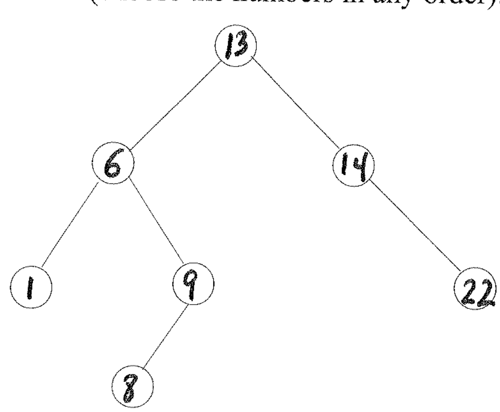
A E C D

Show the stack in each iteration

(2.7) A graph can be represented using either an adjacency list or an adjacency matrix representation. Compare the two approaches (list their advantages/disadvantages)

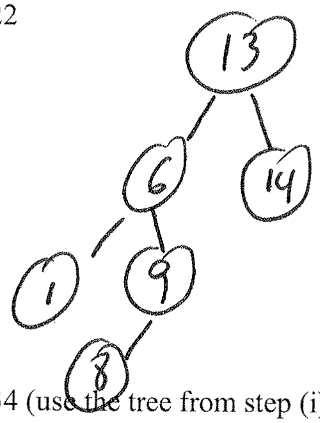
Matrix	List
– memory: $O(V+E^2) = O(V^2)$	– $O(V+E)$
– better for dense graphs	– better for sparse graphs
– easier to check if two vertices are connected	– easier to find the vertices adjacent from another vertex.

(2.8) Label the following binary tree with numbers from the set {6,22,9,14,13,1,8} so that it is a legal binary search tree (choose the numbers in any order).

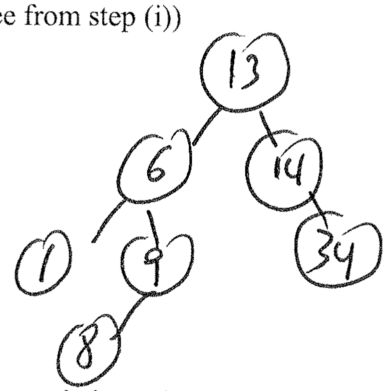


(2.9) Show how the tree in (2.8) would look like after each of the following operations:

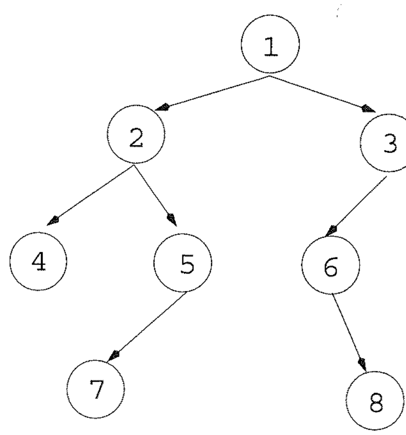
(i) delete 22



(ii) insert 34 (use the tree from step (i))

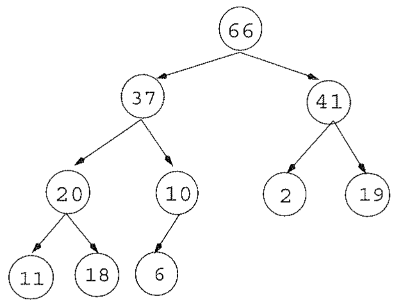


(2.10) Given the tree shown below, show the order in which nodes in the tree are processed by preorder traversal.

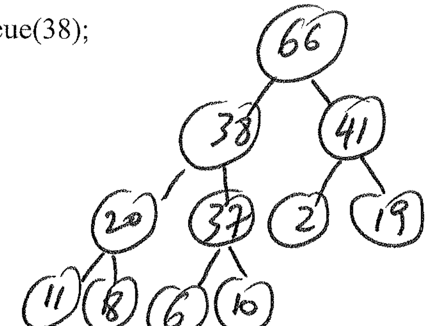


1 2 4 5 7 3 6 8

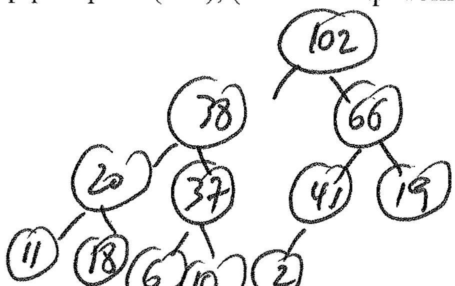
(2.11) A priority queue is implemented as a heap. Show how the heap shown below would look like after each of the following operations:



(i) pq.Enqueue(38);



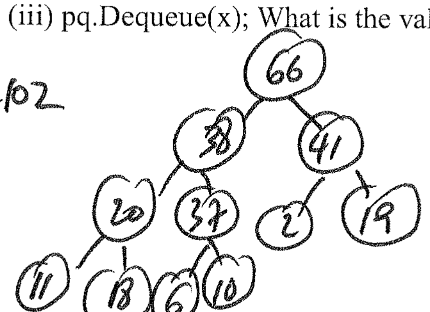
(ii) pq.Enqueue(102); (use the heap from step (i))



(2.12) Continue problem (2.11)

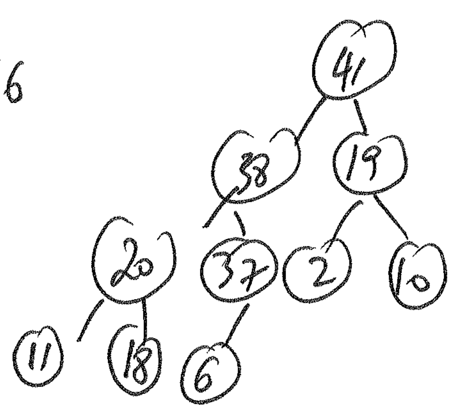
(iii) pq.Dequeue(x); What is the value of x ? (use the heap from step (ii))

X=102



(v) pq.Dequeue(y); What is the value of y ? (use the heap from step (iii))

Y=66

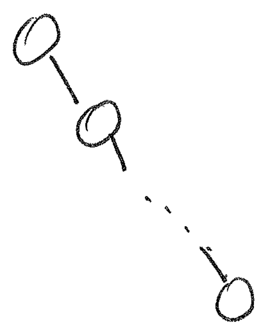


Complete tree!

(2.13) Heaps are usually implemented using arrays. Why ? (be specific). What property of heaps allow us to implement them using arrays ?

- Save memory
- we can compute the parent of a node easily
- the children nodes can be computed easily too.

(2.14) Suppose *N* elements are inserted in order, from smallest to largest, into a binary search tree. Describe the efficiency of searching for an element in the tree in terms of Big-O notation.



this will create a linked-list

O(N) time

(2.15) Trace the function below and describe what it does.

```
template<class ItemType>
int Mystery(TreeType<ItemType> *tree, int &n)
{
 if(tree != NULL) {
   n++;
   Mystery(tree->left, n);
   Mystery(tree->right, n);
 }
}
```

counts the # of nodes in a binary tree.

3. **Code**

(3.1) (10 pts) Write a function that returns the largest value in a binary search tree (full credit will be given only to the most efficient solutions).

```
template <class ItemType>
ItemType   TreeType<ItemType>:: Largest ()
{
    return    LargestValue (root);
}

template <class ItemType>
ItemType  LargestValue (TreeNode <ItemType>* tree)
{ if  (tree-> right  != NULL) { /* assumes
                                    that the
         tree  = tree-> right;      tree is
                                    not
         LargestValue (tree); }     empty */
   return  tree-> info;
}
```

(3.2) (10 pts) Write a boolean member function *IsBST* that determines if a binary tree is a binary search tree.

```
bool TreeType<ItemType>:: IsBST()
{ return IsTrue(TreeNode<ItemType> * tree);
}
- - - - - - - - - - - - - - - - - - - - -
bool IsTrue(TreeNode<ItemType> * tree)
{ if (tree == NULL)
        return true;
  else if (tree->left != NULL &&
           tree->left->info > tree->info )
        return false;
  else if (tree->right != NULL &&
           tree->right->info <= tree->info )
        return false;
  else
  return IsTrue(tree->left) && IsTrue(tree->right);
}
```

(3.3) (5 pts) Give the pseudo-code of the Depth-First-Search approach. How is it different from the Breadth-First-Search approach ?

```
found = false;
Stack.Push (Start Vertex)
do
  Stack.Pop (vertex)
  If vertex = = end Vertex
      found = true
  else
  Push all adjacent vertices onto Stack
While !Stack.IsEmpty() && ! found

if (! found)
   Write "Path does not exist"
```

BFS uses a queue instead of a stack !!

Look at all possible paths at the same depth before going to a deeper level!