3013 - Algorithms **Name:** ______________________

Fall 2021

Exam 1

02/09/2020

READ THE INSTRUCTIONS

- Use pencil only
- Write your name at the top of all pages turned in.
- Do not remove the staple from your test.
- Handwriting that is illegible (messy, small, not straight) will lose points.
- Indentation matters. Keep code aligned correctly.
- Answer all questions in the provided space directly on the test.
- **Failure to comply will result in loss of letter grade.**

This exam is 8 pages (without cover page) and 11 questions. Total of points is 173.

Grade Table (don't write on it)

Question	Points	Score
1	10	
2	10	
3	10	
4	24	
5	14	
6	20	
7	15	
8	25	
9	15	
10	20	
11	10	
Total:	173	

1. (10 points) Given the following list of numbers:

11 , 13 , 27 , 7 , 17 , 9 , 5 , 31 , 3 , 8

load them into an array based binary search tree starting with the leftmost number and moving right.

Answer:
Remember that this is an array representation of a binary search tree. It is highly likely there will be gaps between values (see actual tree below using the same values). Unlike the "heap" (far bottom) which maintains a "complete" tree, simply placing values in an array using Left $= 2 * i$ and Right $= 2 * i + 1$ to find child locations does NOT guarantee a nice complete tree.

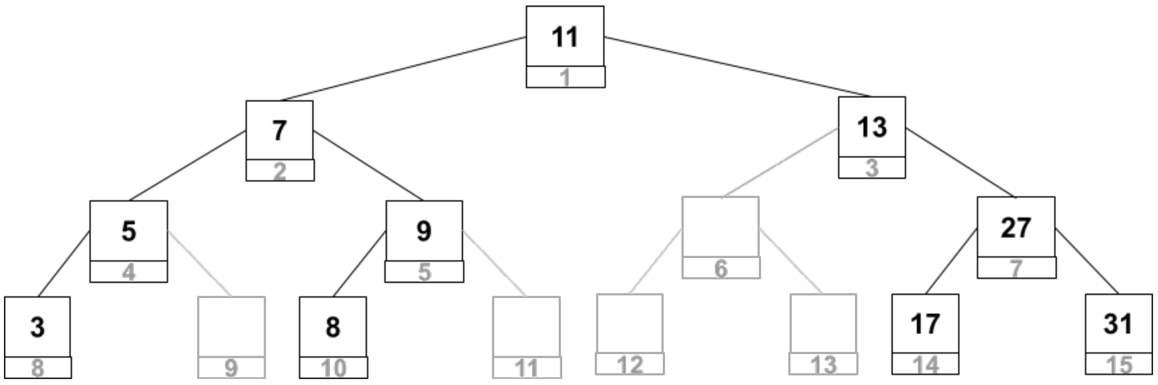
	11	7	13	5	9		27	3		8				17	31	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

2. (10 points) Given the following list of numbers:

11 , 13 , 27 , 7 , 17 , 9 , 5 , 31 , 3 , 8

load them into a binary search tree starting with the leftmost number and moving right. Use the image below and assume any nodes without values do not exist.

Answer:
Below is a binary search tree with the values loaded. I've included numeric indexes at each node to show the relationship with the array above.



3. (10 points) Given the following list of numbers:

11 , 13 , 27 , 7 , 17 , 9 , 5 , 31 , 3 , 8

load them into an the binary **MAX** heap starting with the leftmost number and moving right.

Answer:
Remember we add new values at the end of the array and bubble them up. In a MAX heap, if a value is larger than its parent (Parent $= i/2$) we swap with them, and continue swapping until its at the top or not larger than its parent. A heap maintains a **complete** tree, so there are no gaps in the array.

	31	27	13	17	11	9	5	7	3	8						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

4. (24 points) List the complexities from fastest to slowest.

Complexity Choices

$O(n!)$ $O(2^n)$ $O(1)$ $O(n\ lg\ n)$ $O(n^2)$ $O(n^n)$ $O(n)$ $O(log\ n)$

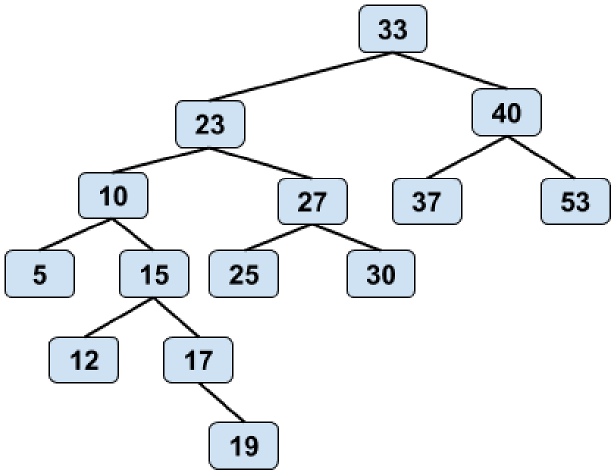
Answer:

$O(1)$	$O(log\ n)$	$O(n)$	$O(n\ lg\ n)$	$O(n^2)$	$O(2^n)$	$O(n!)$	$O(n^n)$
1	2	3	4	5	6	7	8

←Fastest Slowest →

---

5. (14 points) Given the following binary tree:



1. What is the height of this tree?
2. What is the root of this tree?
3. How many leaves does this tree have? List them.
4. How many descendants does 23 have?
5. How many siblings does 23 have?
6. Who is the predecessor of 23?
7. Who is the successor of 23?

Answer:

5 or 6	**33**	**7**	**9**	**1**	**19**	**25**
1	2	3	4	5	6	7

3) Leaves: 5, 12, 19, 25, 30, 37, 53

6. (20 points) Assign the correct complexity to each item below. Use the array below to write your answers.

Complexity Choices

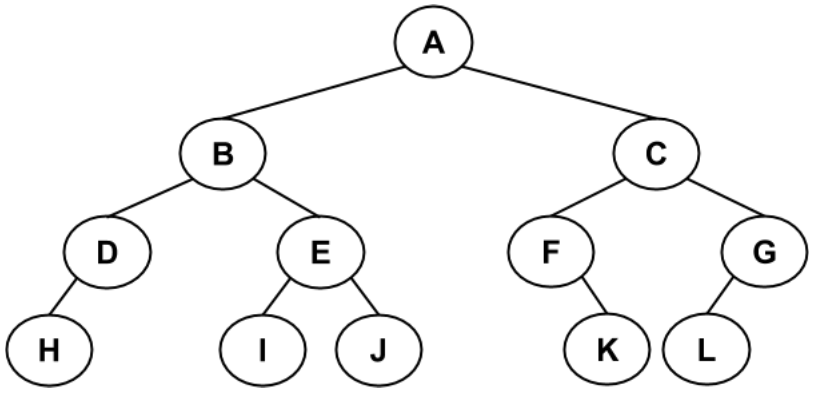
$O(n!)$ $O(2^n)$ $O(1)$ $O(n\ lg\ n)$ $O(n^2)$ $O(n^n)$ $O(n)$ $O(log\ n)$ None of These

A. Inserting an element into a balanced binary search tree.
B. Finding an element in a list.
C. Finding an element in an ordered list.
D. Removing an element from a binary heap.
E. Finding an element in a binary search tree.
F. Adding an element to a binary heap.
G. Building a binary heap given an array of values.
H. Building a binary heap given a linked list of values.
I. Remove an item from a linked list of values.
J. Insert an item into an ordered linked list of values.

Answer:

$O(log\ n)$	$O(n)$	$O(log\ n)$	$O(lg\ n)$	$O(log\ n)$ OR $O(H)$	$O(log\ n)$	$O(n)$	$O(n\ log\ n)$	$O(n)$	$O(n)$
A	B	C	D	E	F	G	H	I	J

7. (15 points) Do a pre-order, in-order, and post-order traversal of the following tree. Place your answers in the arrays below.



PRE	A	B	D	H	E	I	J	C	F	K	G	L
IN	H	D	B	I	E	J	A	F	K	C	L	G
POST	H	D	I	J	E	B	K	F	L	G	C	A

8. (25 points) Write a complete function that implements one of the $O(n^2)$ sorting algorithms. Name your function the same as the algorithm you are implementing.

```
void swap(int *xp, int *yp){
        int temp = *xp;
        *xp = *yp;
        *yp = temp;
}

// https://www.geeksforgeeks.org/bubble-sort/
void bubbleSort(int arr[], int n) {
        int i, j;
        bool swapped;
        for (i = 0; i < n-1; i++) {
                swapped = false;
                for (j = 0; j < n-i-1; j++) {
                        if (arr[j] > arr[j+1]) {
                                swap(&arr[j], &arr[j+1]);
                                swapped = true;
                        }
                }
                // IF no two elements were swapped by inner loop, then break
                if (swapped == false)
                break;
        }
}

//https://www.geeksforgeeks.org/selection-sort/
void selectionSort(int arr[], int n){
        int i, j, min_idx;

        // One by one move boundary of unsorted subarray
        for (i = 0; i < n-1; i++){
                // Find the minimum element in unsorted array
                min_idx = i;
                for (j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                min_idx = j;
                // Swap the found minimum element with the first element
                swap(&arr[min_idx], &arr[i]);
        }
}

//https://www.geeksforgeeks.org/insertion-sort/
void insertionSort(int arr[], int n){
        int i, key, j;
        for (i = 1; i < n; i++){
                key = arr[i];
                j = i - 1;

                /* Move elements of arr[0..i-1], that are  greater than key, to one position ahead of their current position */
                while (j >= 0 && arr[j] > key){
                        arr[j + 1] = arr[j];
                        j = j - 1;
                }
                arr[j + 1] = key;
        }
}
```

9. (15 points) Recursive tree traversal. Assume the following node structure.

```
struct Node{
        int data;
        Node* left;
        Node* right;
};
```

Write a complete function that traverses a binary search tree, and sums all the values. Name your function **TreeSum**. You can assume that you have access to the root of the tree (as if you were in the tree class).

```
int TreeSum(Node* root){
        if(!root){
                return 0;
        }
        return root->data + TreeSum(root->left) + TreeSum(root->right);
}
```

10. (20 points) Explain what a Binary Heap data structure can be used for. Discuss its complexity. How is the best complexity achieved? For example: Can we get the best performance from a binary heap by inserting items one at a time? Discuss the methods required to maintain a heap. Talk about what a "heap property" is. How is it different from a Binary Search Tree?

Answer:

Complexity: A binary heaps complexity revolves around $O(lg\ n)$ since its structure remains a complete binary tree. This implies that it is always balanced. So an insertion or removal of an item costs at most $O(lg\ n)$.

If we were to insert 1 item at a time into a binary heap it would cost $O(n\ lg\ n)$. Then to remove all the items would cost another $O(n\ lg\ n)$ hmmmm. What makes a binary heap so awesome? It's the heapify method: given an array of unordered items we can turn that into a heap in linear time ($O(n)$). This is the basis for the $O(n\ lg\ n)$ sort time that heap sort is known for.

Methods: Every data structure needs some kind of ***insert*** and a ***remove*** method. But what drives a binary heap are its ***bubble up*** and ***bubble down*** methods (aka sift-up/sift-down trickle-up/trickle-down etc.). Every time we insert a new value we use ***bubble up*** to place the value in its proper location so we can maintain the "heap property". Likewise when we remove an item, we swap the rightmost value in the array with the first array location and then call ***bubble down*** to place that value into its proper location.

Heap Property: The "heap property" is the ordering in which a heap maintains. Depending on whether you are a min or max heap you just need to know: "do I swap with my parent?" For a max heap: larger children swap with their parents (and vice versa for min heap). Basically a binary search tree has a total ordering $left\ child < parent < right\ child$ whereas a heap is $parent > children$ (for a max heap). There is no order between siblings. This partial ordering is what allows heapify to run in $O(n)$. We discussed this in class in a little more depth.

11. (10 points) BONUS: Write a recursive TreeHeight function. Assume the same things as in the recursive tree traversal question. This function would return the height of a binary tree.

```
int TreeHeight(Node* root){
    if(!root){
        return 0;
    }
    int LH = Height(node->left);
    int RH = Height(node->right);

    if(LH > RH){
        return LH + 1;
    }else{
        return RH + 1;
    }
}
```