Name: _____

5243 Algorithms Exam 1
Spring 2025

For the next three questions, use the following list of numbers: **11 , 13 , 27 , 7 , 17 , 9 , 5 , 31 , 3 , 8**

1. (10 points) Load the above list of numbers into an **array based** binary search tree starting with the leftmost number and moving right.

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

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

Combine that with whether they fundamentally used BST logic or not. If they didn’t follow BST rules at all, you might give them at most 2–3 points.

2. (10 points) Load the above list of numbers into an array based **binary MAX heap** starting with the leftmost number and moving right.

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

0	1	2	3	4	5	6	7	8	9

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.

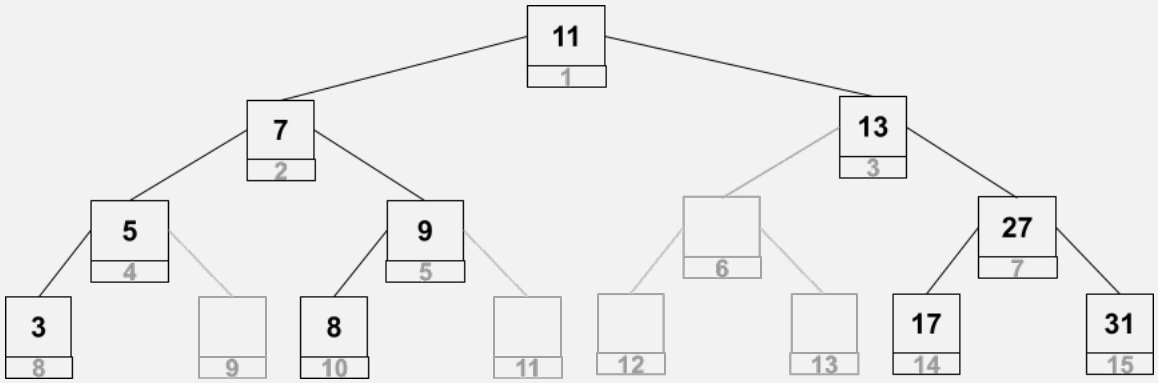
	31	**27**	**13**	**17**	**11**	**9**	**5**	**7**	**3**	**8**						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

3. (10 points) Load the above list of numbers into a **binary search tree** starting with the leftmost number and moving right.

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

Answer:

	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



4. (10 points) Write a recursive function that would count down from a given number, and print the following (assuming the number passed in was 5):

```
5
4
3
2
1
Blast Off!!
```

Answer:

```
void countdown(int n) {
  if (n <= 0) {
    cout << "Blast off!";
  } else {
    cout << n << endl;
    countdown(n - 1);
  }
}

// or

void countdown(int n) {
  if (n <= 0) {
    cout << "Blast off!";
    return;
  }
  cout << n << endl;
  countdown(n - 1);
}
```

5. (15 points) List vs Array based data structures. Read the statements below and place the most appropriate label from the following: **List**, **Array**, **Both**, **None**

1. Possible to directly access an element in this structure.

5. **Array**

2. Is bounded by size.

5. **Array**

3. Easy to insert and delete from while not having empty nodes / slots.

5. **List**

4. Easiest to implement.

5. **Array**

5. Has more overhead.

5. **List**

6. This structure can easily be sorted.

5. **Array**

7. Must be allocated in the heap.

5. **List**

8. This structure is expensive to resize.

5. **Array**

9. Which structure grows and shrinks easily.

5. **List**

10. Binary search can easily be performed on this sorted structure.

5. **Array**

11. Searching this structure can be done in $O(N)$.

5. **Both**

12. Can add items added to either end of this structure easily.

5. **List**

13. It is easier to manipulate the middle of this structure without big costs.

5. **List**

14. Can be used to represent a binary tree.

5. **Both**

6. (10 points) List the complexities from fastest to slowest.

A	B	C	D	E	F	G	H
$O(n!)$	$O(2^n)$	$O(1)$	$O(n \log n)$	$O(n^2)$	$O(n^n)$	$O(n)$	$O(\log n)$

Write the letters of the corresponding complexity in the table below with the fastest starting from the left.

Answer:

The spreadsheet below shows the growth of each choice based on the N value in column 1. The columns go from least cost on the left to greatest cost on the right

Fastest Slowest

	C	H	G	D	E	B	A	F
N	O(1)	O(lg N)	O(N)	O(N lg N)	O(N^2)	O(2^N)	O(N!)	O(N^N)
2	1	1	2	2	4	4	2	4
3	1	2	3	5	9	8	6	27
4.5	1	2	4.5	10	20.25	22.627417	24	869.8739234
6.75	1	3	6.75	19	45.5625	107.6347412	720	396096.0091
10.125	1	3	10.125	34	102.515625	1116.679918	3628800	15122538466
15.1875	1	4	15.1875	60	230.6601563	37315.82598	130767436800	8.78664E+17
22.78125	1	5	22.78125	103	518.9853516	7208411.797	1.124E+21	8.45889E+30
34.171875	1	5	34.171875	174	1167.717041	19353494041	2.95233E+38	2.56058E+52
51.2578125	1	6	51.2578125	291	2627.363342	2.6924E+15	1.55112E+66	4.35072E+87
76.88671875	1	6	76.88671875	482	5911.56752	1.39704E+23	1.88549E+111	9.92934E+144
115.3300781	1	7	115.3300781	790	13301.02692	5.22171E+34	2.92509E+188	6.36795E+237

[10] Implement either a bubble sort function or a selection sort function. You can assume that a swap function exists. The function header will look like:

```
void SortName(int A*, int size)
```

Answer:

Either one of the following:

```
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex])
                minIndex = j;
        }
        swap(arr[minIndex], arr[i]);
    }
}

void bubbleSort(int arr[], int n) {
    bool swapped;
    for (int i = 0; i < n - 1; i++) {
        swapped = false;
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
                swapped = true;
            }
        }
        if (!swapped) // If no swaps occurred, array is sorted
            break;
    }
}
```

[10] Use a binary search on the following array of numbers and search for the key with the value **3**

2	**3**	**5**	**7**	**11**	**13**	**17**	**19**	**23**	**29**	**31**	**37**	**41**
0	1	2	3	4	5	6	7	8	9	10	11	12

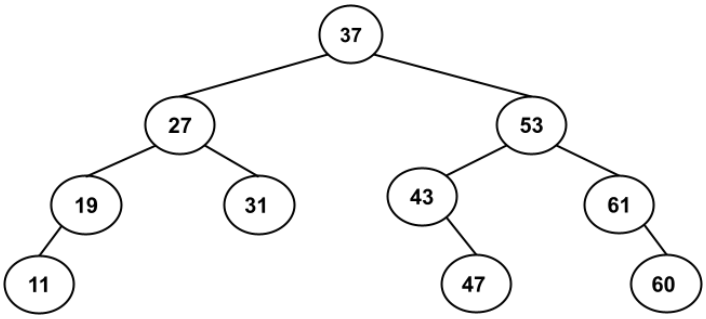
Answer:

This only required you to use two rounds even if placing L M & R in same positions without moving them to avoid double checking locations that we have searched already.

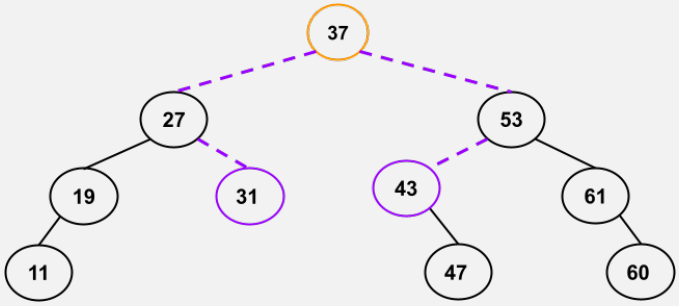
2	**3**	**5**	**7**	**11**	**13**	**17**	**19**	**23**	**29**	**31**	**37**	**41**
0	1	2	3	4	5	6	7	8	9	10	11	12

Step 1	L						M						R
Step 2		L		M		R							

[10] Identify the inorder successor and predecessor to the root of the tree.



Answer:
[10] Identify the inorder successor and predecessor to the root of the tree.



Answer the multiple choice questions below. Read them carefully, and answer them on this sheet like the example show below.

(2 points) How should you answer your multiple choice questions on this exam?

A

A. Follow this example. Look to the left and see the huge letter? Do that!
B. Don't answer because that's safer.
C. Simply guess, you have a 25% chance of getting it right.
D. Make up your own answer because that's not crazy.
E. Hail to the mystical beings of the order of the correct answer.

(1) (2 points) Which of the following data structures automatically resizes itself when elements are added or removed?

A. Static Array
B. Linked List
C. Heap
D. Stack
E. None of the above

(2) (2 points) What is the time complexity of accessing an element in an array-based data structure?

A. $O(1)$
B. $O(n)$
C. $O(logn)$
D. $O(n^2)$
E. None of the above

(3) (2 points) Which data structure uses LIFO (Last In, First Out) principle?

A. Queue
B. Stack
C. Array
D. Linked List
E. None of the above

(4) (2 points) In a linked list, each element is stored in a structure called:

A. Node
B. Array
C. Vertex
D. Edge
E. None of the above

(5) (2 points) Which sorting algorithm repeatedly swaps adjacent elements that are in the wrong order?

A. Insertion Sort
B. Bubble Sort
C. Quick Sort
D. Merge Sort
E. None of the above

(6) (2 points) Which of the following is not a characteristic of a binary search algorithm?

A. Requires a sorted array
B. Operates in O(log n) time complexity
C. Uses a divide and conquer approach
D. Works well with linked list data structures
E. None of the above

(7) (2 points) Where do statically declared variables typically reside?

A. Heap Memory
B. Stack Memory
C. Global Memory
D. Virtual Memory
E. None of the above

(8) (2 points) Which type of memory allocation is used for dynamically declared variables?

A. Stack Allocation
B. Heap Allocation
C. Static Allocation
D. Register Allocation
E. None of the above

(9) (2 points) What keyword is often associated with dynamically allocated memory in many programming languages?

A. static
B. const
C. new
D. auto
E. None of the above

(10) (2 points) Which of the following is a characteristic of heap memory?
A. Automatically manages memory allocation and deallocation
B. Typically faster access compared to stack memory
C. Used for global variable storage
D. Memory must be manually managed (allocated and deallocated)
E. None of the above

(11) (2 points) Which of the following scenarios is more likely to cause a memory leak?
A. Forgetting to deallocate memory allocated on the stack
B. Forgetting to deallocate memory allocated on the heap
C. Declaring too many local variables inside a function
D. Using static memory allocation for all variables
E. None of the above

(12) (2 points) In what scenario does storing a binary tree in an array result in inefficient use of space?
A. When the tree is complete
B. When the tree is full
C. When the tree is balanced
D. When the tree is sparse
E. None of the above

(13) (2 points) What is the time complexity of search in an unbalanced binary tree?
A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)
E. None of the above

(14) (2 points) Which data structure is best suited for efficiently finding the nth largest element assuming sorted values?
A. Linked List
B. Priority Queue
C. Stack
D. Array
E. None of the above

(15) (2 points) If I wanted to reverse the values in an array, what data structure would be the most help?
A. Linked List
B. Priority Queue
C. Stack
D. Array
E. None of the above

(16) (2 points) What is a primary advantage of using an array over a linked list?
A. Dynamic sizing
B. Lower memory overhead
C. Ease of insertion and deletion
D. Fast access to elements by index
E. None of the above

(17) (2 points) Which of the following statements is true about linked lists compared to arrays?
A. Linked lists have a fixed size.
B. Linked lists are more memory efficient.
C. Linked lists allow for easy resizing.
D. Linked lists provide faster access by index.
E. None of the above

(18) (2 points) Why might resizing an array be considered costly?
A. Because it requires additional memory for pointers.
B. Because it involves creating a new array and copying elements.
C. Because run time stack memory is slow.
D. There's never a need to resize an array.

E. None of the above

(19) (2 points) In what scenario would a linked list be preferred over an array?

A. When memory usage is a critical concern.
B. When frequent resizing of the data structure is required.
C. When fast access to random elements is needed.
D. When the data structure needs to be sorted.
E. None of the above

(20) (2 points) What is a drawback of linked lists compared to arrays?

A. Higher memory usage due to storage of pointers.
B. Inability to store multiple data types.
C. Fixed size.
D. Slower resizing operations.
E. None of the above

(21) (2 points) How does an array's bounded size impact its use?

A. It makes arrays faster than linked lists.
B. It limits the number of elements an array can hold.
C. It allows for dynamic resizing.
D. It reduces memory overhead.
E. None of the above

(22) (2 points) Which of the following is not a characteristic of a linked list?

A. Direct access to nodes.
B. No bounded size.
C. Overhead cost of managing pointers.
D. Easy growth and shrinkage.
E. None of the above

(23) (2 points) What makes linked lists more flexible in terms of size compared to arrays?

A. The use of contiguous memory allocation.
B. The fixed size.
C. The ability to add or remove elements without resizing the entire structure.
D. Faster access time to elements.
E. None of the above

(24) (2 points) In a list-based priority queue, what is the time complexity of finding the correct location to insert a new element with a given priority?

A. $O(lg\, n)$
B. $O(n)$
C. $O(n^2)$
D. $O(array\ size)$
E. $O(n\, lg\, n)$

(25) (2 points) Which of the following best describes a priority queue?

A. A data structure that allows for LIFO (Last In, First Out) access.
B. A collection that returns the highest or lowest element based on priority.
C. A type of queue where elements are processed alphabetically.
D. A data structure where elements are always sorted in ascending order.
E. None of the above

(26) (2 points) What is the time complexity of inserting an element in an array-based priority queue (not a binary heap) where the array is kept sorted?

A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)
E. None of the above

(27) (2 points) In a list-based priority queue, what is the time complexity of finding the correct location to insert a new element with a given priority?

A. $O(lg\, n)$
B. $O(n)$
C. $O(n^2)$
D. $O(array\ size)$
E. $O(n\, lg\, n)$

(28) (2 points) Which of the following operations tends to be more efficient in a list-based priority queue compared to an array-based priority queue?

A. Removing the highest-priority element

B. Searching for the proper location of the new element
C. Increasing the priority of an element
D. Inserting an element at the end
E. None of the above

(29) (2 points) What is the cost for removing an item from a binary heap?

A. $O(lg\, n)$
B. $O(n)$
C. $O(n^2)$
D. $O(array\ size)$
E. $O(n\, lg\, n)$

(30) (2 points) Can be used to implement a scheduler for tasks that need to be executed in a specific order.

A. Array
B. List
C. Priority Queue
D. Queues
E. Stacks

(31) (2 points) Elements can be of varying sizes and types (heterogeneous).

A. Array
B. List
C. Priority Queue
D. Queues
E. Stacks

(32) (2 points) What is the cost for finding an item in an ordered array using binary search?

A. $O(lg\, n)$
B. $O(n)$
C. $O(n^2)$
D. $O(array\ size)$
E. $O(n\, lg\, n)$

(33) (2 points) Follows a Last In, First Out (LIFO) principle.

A. Array
B. List
C. Priority Queue
D. Queues
E. Stacks

(34) (2 points) Ideal for scenarios like undo mechanisms in text editors.

A. Array
B. List
C. Priority Queue
D. Queues
E. Stacks

(35) (2 points) Insertion and deletion of elements at the beginning are typically faster.

A. Array
B. List
C. Priority Queue
D. Queues
E. Stacks

(36) (2 points) Memory overhead due to storing pointers to the next (and possibly previous) elements.

A. Array
B. List
C. Priority Queue
D. Queues
E. Stacks

(37) (2 points) Operates on a First In, First Out (FIFO) principle.

A. Array
B. List
C. Priority Queue
D. Queues
E. Stacks

(38) (2 points) Preferred for applications where memory layout and access speed are critical.

A. Array
B. List
C. Priority Queue
D. Queues
E. Stacks

(39) (2 points) Suitable for processing jobs in order like print requests.

A. Array
B. List
C. Priority Queue
D. Queues
E. Stacks