Name: _____

5243 Algorithms Exam 1
Spring 2026

1. (10 points) Load the list of numbers into an **array-based BST** (insert left-to-right).

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

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Step 1	Insert 11	**11**															
Step 2	Insert 13	11		**13**													
Step 3	Insert 27	11		13				**27**									
Step 4	Insert 07	11	**7**	13				27									
Step 5	Insert 17	11	7	13				27							**17**		
Step 6	Insert 09	11	7	13		**9**		27							17		
Step 7	Insert 05	11	7	13	**5**	9		27							17		
Step 8	Insert 31	11	7	13	5	9		27							17	**31**	
Step 9	Insert 03	11	7	13	5	9		27	**3**						17	31	
Step 10	Insert 08	11	7	13	5	9		27	3		**8**				17	31	
Step 1	Insert 11	Array is empty, so 11 goes into slot 0															
Step 2	Insert 13	13 > 11 (2*0)+2 = **2**															
Step 3	Insert 27	27 > 11 R=(2*0)+2 = **2**, 27 > 13 R=(2*2)+2 = **6**															
Step 4	Insert 07	7 < 11 L=(2*0)+1 = **1**															
Step 5	Insert 17	17 > 11 R=(2*0)+2 = **2**, 17 > 13 R=(2*2)+2 = **6**, 17 < 27 L=(2*6)+1 = **13**															
Step 6	Insert 09	9 < 11 L=(2*0)+1 = **1**, 9 > 7 R=(2*1)+2 = **4**															
Step 7	Insert 05	5 < 11 L=(2*0)+1 = **1**, 5 < 7 L=(2*1)+1 = **3**															
Step 8	Insert 31	17 > 11 R=(2*0)+2 = **2**, 17 > 13 R=(2*2)+2 = **6**, 31 > 27 R=(2*6)+2 = **14**															
Step 9	Insert 03	3 < 11 L=(2*0)+1 = **1**, 3 < 7 L=(2*1)+1 = **3**, 3 < 5 L=(2*3)+1 = **7**															
Step 10	Insert 08	8 < 11 L=(2*0)+1 = **1**, 8 > 7 R=(2*1)+2 = **4**, 8 < 9 L=(2*4)+1 = **9**															

---

2. (10 points) Load the 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

	Parents:		0	0	1	1	2	2	3	3	4	4	5	5	6	6	7
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Step 1	Insert 11	**11**															
Step 2	Insert 13	**13**	11														
Step 3	Insert 27	**27**	11	13													
Step 4	Insert 07	27	11	13	**7**												
Step 5	Insert 17	27	**17**	13	7	11											
Step 6	Insert 09	27	17	13	7	11	**9**										
Step 7	Insert 05	27	17	13	7	11	9	**5**									
Step 8	Insert 31	**31**	27	13	17	11	9	5	7								
Step 9	Insert 03	31	27	13	17	11	9	5	7	**3**							
Step 10	Insert 08	31	27	13	17	11	9	5	7	3	**8**						
Step 1	Insert 11	Array is empty, so 11 goes into index 0															
Step 2	Insert 13	At index 1, 13 > Parent:11 = Swap 13/11															
Step 3	Insert 27	At index 2, 27 > Parent:13 = Swap 27/13															
Step 4	Insert 07	At index 3, 7 < Parent:11 = Stay															
Step 5	Insert 17	At index 4, 17 > Parent:11 = Swap 17/11, 17 < Parent:27 = Stay															
Step 6	Insert 09	At index 5, 9 < Parent:13 = Stay															
Step 7	Insert 05	At index 6, 5 < Parent:13 = Stay															
Step 8	Insert 31	At index 7, 31 > Parent:07 = Swap 31/07, 31 > Parent:17 = Swap 31/17, 31 > Parent:27 = Swap 31/27, 31 at top															
Step 9	Insert 03	At index 8, 3 < Parent:17 = Stay															
Step 10	Insert 08	At index 9, 8 < Parent:11 = Stay															

3. (10 points) Binary search for key **3**. Show steps in the grid.

		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
V1	Step 1	L						M						R
V1	Step 2		L		M		R							
V2	Step 1	L						M						R
V2	Step 2	L			M			R						
V2	Step 3	L	M		R									

Version 1

Step 1 L=0, R=12, M=(0+12)/2=6

None of the index's are the key and key < middle we go left

Step 2 L=1, R=5, M=(1+5)/2=3

Key is at L index so we stop

Versio

Step 1 L=0, R=12, M=(0+12)/2=6

None of the index's are the key and key < middle we go left

Step 2 L=0, R=6, M=3

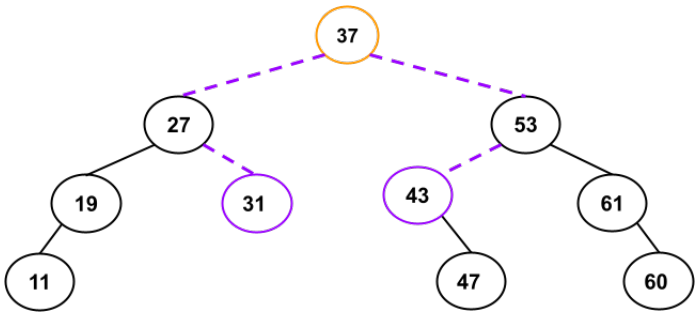
M does not find key, but key < middle, we go left

Step 3 L=0, R=3, M=1

M finds the key so we stop

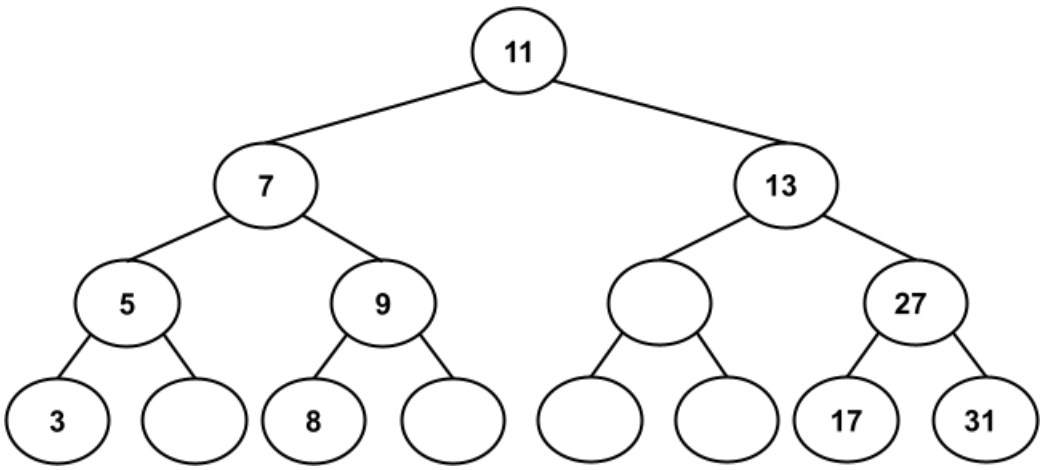
---

4. (10 points) Identify the inorder successor and predecessor of the root.



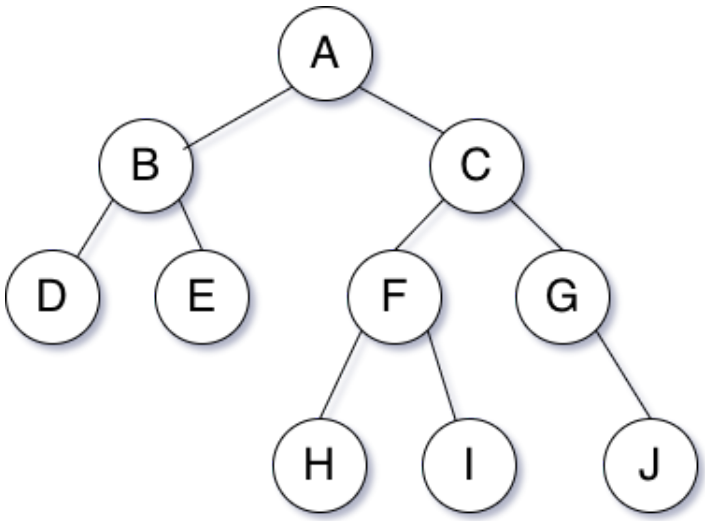
5. (10 points) Load the above list of numbers into a **binary search tree** starting with the leftmost number and moving right. Use the tree template below knowing that there will be some unused tree nodes.

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



---

6. (10 points) Write out the PreOrder PostOrder and InOrder traversals of the following binary tree:



PreOrder	A	B	D	E	C	F	H	I	G	J
InOrder	D	B	E	A	H	F	I	C	G	J
PostOrder	D	E	B	H	I	F	J	G	C	A

7. (10 points) List the complexities from fastest to slowest.

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

Write the letters of the corresponding complexity in the table below with the fastest starting from the left. Put your letters in the top row, along with the complexity in the bottom row.

C	G	F	D	E	B	A
O(1)	O(LG N)	O(N)	O(N LG N	O(N^2)	O(2^N)	O(N!)
1	2	3	4	5	6	7

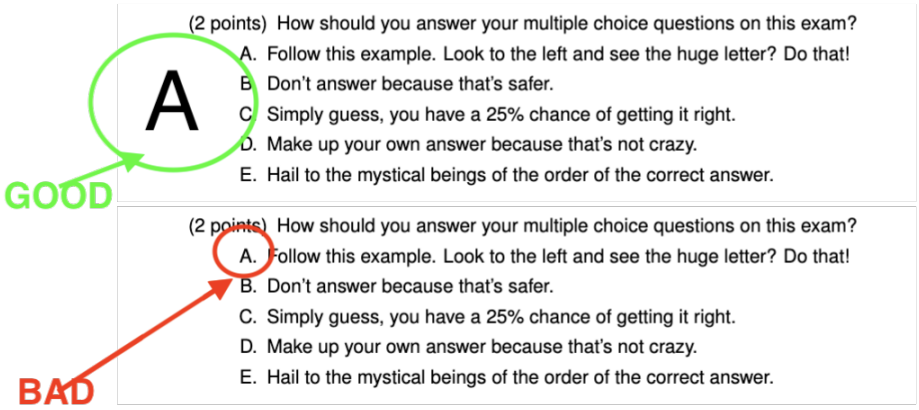
---

8. (10 points) Write a recursive function that counts down from n and prints `Blast Off!!` at the end.

```
void blastOff(int x){
    if(!x){
        cout<<"Blast Off!""<<endl;
    }else{
        cout<<x<<endl;
        blastOff(--x);
    }
}
```

9. (30 points) Label each statement as **List**, **Array**, **Both**, or **None**.
    1. **Array** Possible to directly access an element in this structure.
    2. **Array** Is bounded by size.
    3. **List** Easy to insert and delete from while not having empty nodes / slots.
    4. **Array** Easiest to implement.
    5. **List** Has more overhead.
    6. **Array** This structure can easily be sorted.
    7. **List** Must be allocated in the heap.
    8. **Array** This structure is expensive to resize.
    9. **List** Which structure grows and shrinks easily.
    10. **Array** Binary search can easily be performed on this sorted structure.
    11. **Both** Searching this structure can be done in $O(N)$.
    12. **List** Can add items added to either end of this structure easily.
    13. **List** It is easier to manipulate the middle of this structure without big costs.
    14. **Both** Can be used to represent a binary tree.

10. **Read each question them carefully, and answer them on this sheet like the example show below.**



CHOOSE ONLY 20 QUESTIONS TO ANSWER.

Don't answer any more, I will only grade the first 20 I see. So, there is no maximizing your chances ... sorry.

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