3013 - Algorithms **Name:** ____________________
Spring 2024
Exam One
February 14, 2024

1. Multiple Choice

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) What is the run time stack primarily used for?
    A. Storing dynamically allocated variables
    B. Storing temporary variables created by each function
    C. Managing the memory used by operating system processes
    D. Storing the program's executable code
    E. None of the above

12. (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

13. (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

14. (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

15. (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

16. (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

17. (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

18. (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

19. (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

20. (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

21. (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

22. (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

23. (2 points) What aspect of arrays makes them easy and fast to use in many languages?
    A. The ability to store elements of different data types.
    B. The dynamic resizing capability.
    C. The contiguous memory allocation.
    D. The automatic management of pointers.
    E. None of the above

24. (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

25. (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

26. (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

27. (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

28. (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(1)
    - B. O(log n)
    - **C. O(n)**
    - D. O(n log n)
    - E. None of the above

29. (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

30. (2 points) Why might a list-based priority queue be preferred over an array-based priority queue in certain scenarios?
    - **A. Lower overhead when frequently resizing the data structure**
    - B. Guaranteed constant time insertion
    - C. No need to maintain a sorted order
    - D. More efficient memory usage in sparse queues
    - E. None of the above

31. (2 points) Better suited for scenarios where frequent insertions and deletions occur at various positions.
    - A. Array
    - **B. List**
    - C. Priority Queue
    - D. Queues
    - E. Stacks

32. (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

33. (2 points) Commonly used in breadth-first search algorithms in graph theory.
    - A. Array
    - B. List
    - C. Priority Queue
    - **D. Queues**
    - E. Stacks

34. (2 points) Data structure with elements stored in contiguous memory locations.
    - **A. Array**
    - B. List
    - C. Priority Queue
    - D. Queues
    - E. Stacks

35. (2 points) Efficient random access of elements by their index.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks
36. (2 points) Elements are accessed sequentially starting from the head.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks
37. (2 points) Elements are processed in the order they arrive unless one has higher priority.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks
38. (2 points) Elements can be of varying sizes and types (heterogeneous).
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks
39. (2 points) Elements with higher priority are moved to the front of the queue.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks
40. (2 points) Fixed size, determined at the time of allocation.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks
41. (2 points) Follows a Last In, First Out (LIFO) principle.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks

42. (2 points) Ideal for scenarios like undo mechanisms in text editors.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks

43. (2 points) Insertion and deletion of elements at the beginning are typically faster.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks

44. (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

45. (2 points) Operates on a First In, First Out (FIFO) principle.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks

46. (2 points) Preferred for applications where memory layout and access speed are critical.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks

47. (2 points) Size can dynamically grow or shrink as elements are added or removed.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks

48. (2 points) Suitable for queueing requests for a resource like a printer.
    A. Array
    B. List
    C. Priority Queue
    D. Queues
    E. Stacks

49. (2 points) Used for managing tasks in order of importance, not just arrival time.

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

50. (2 points) Utilized in balancing parentheses in compilers.

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

2. Short Answer

51. (10 points) List the complexities from fastest to slowest.

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

Solution: 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

52. (10 points) Given the following list of numbers, load them into an array based binary search tree by reading the numbers from left to right.

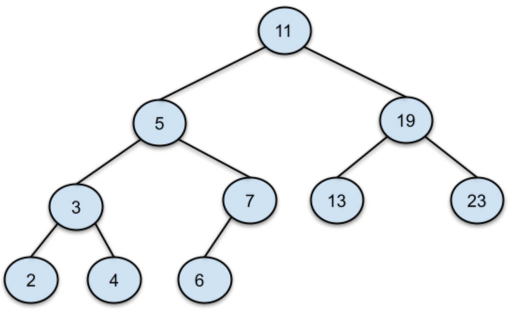
11, 5, 19, 3, 7, 13, 23, 2, ~~17, 29~~ 4, 6

Solution:

- Left Child $= 2 * i$ (2 times the index)
- Right Child $= 2 * i + 1$ (2 times the index + 1)

X	11	5	19	3	7	13	23	2	4	6
0	1	2	3	4	5	6	7	8	9	10

BST Drawing (optional



53. (15 points) Write a selection sort function with the following function header:

Solution:
1) Don't change my variable names.
2) Outer Loop: 0 to size -1.
3) Inner Loop: i + 1 to end.
3) Find minindex on each pass.
4) Do a swap.

```
void selectionSort(int *A, int size) {
    for (int i = 0; i < size - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < size; j++) {
            if (A[j] < A[minIndex]) {
                minIndex = j;
            }
        }
        std::swap(A[i], A[minIndex]);
    }
}
```

}