REVIEW QUESTIONS FOR DATA STRUCTURES

Give two reasons why dynamic memory allocation is a valuable device.

- Dynamic memory allocation is valuable because it allows programs to utilize memory more efficiently by allocating memory as needed during runtime.
- It provides flexibility, as it adapts to the changing memory requirements of a program and avoids fixed allocation limitations in static memory allocation.

Is it easier to insert a new node before or after a specified node in a linked list? Why?

- It is generally easier to insert a node after a specified node_in a linked list because:
- To insert after, only the pointer of the specified node needs to be updated to point to the new node, and the new node's pointer is set to the next node. Inserting before requires traversing the list to find the previous node, which adds complexity and time unless the previous node is already available in special cases.

What are the differences between a linked list and an array? Why would you choose one or the other? Your answer must include the following points:

- **a. Insertion performance**
- **Array**: Insertion can be inefficient as it may require shifting elements, especially for insertion at the beginning or middle.
- **Linked List**: Insertion is efficient as you only need to update pointers, regardless of the position.
- **b. Memory allocation**
- **Array**: Requires contiguous memory allocation, which might lead to memory shortages.
- **Linked List**: Uses dynamic memory allocation and is stored non-contiguously, which consumes memory for the pointers but provides flexibility.
- **c. Impact of removing elements (beginning/middle/end)**
- **Array:** Removal at the beginning or middle shifts all subsequent elements, making it slower.
- **Linked List:** Removal is efficient, as only the pointers need to be updated. No element shifting is required.

Use **array** when faster direct access (indexing) is needed and the size is predefined.

Use **linked list** when frequent insertions and deletions occur or the size is dynamic.

If the items in a list are floats taking 4 words each, compare the amount of space required altogether if

- **a.) the list is kept contiguously in an array 90 percent full:**
  Space = Array size × Word size = 1.11 × Actual items × 4 words.
- **b.) the list is kept contiguously in an array 60 percent full:**
  Space = 1.66 × Actual items × 4 words.
- **c.) the list is kept as a linked list (where the pointers take one word each)**
  Space = Items × (Data size + Pointer size) = Actual items × (4 + 1) = Actual items × 5 words.

If the items in a list are structures or records taking 200 words each, compare the amount of space required altogether if

- **(a) the list is kept contiguously in an array 85 percent full:** Space = Array size × Word size = 1.18 × Actual items × 200 words.
- **(b) the list is kept contiguously in an array 50 percent full:** Space = 2 × Actual items × 200 words.
- **(c) the list is kept as a linked list (where the pointers take one word each):** Space = Items × (Data size + Pointer size) = Actual items × (200 + 1) = Actual items × 201 words.

When it is appropriate to use the data structure Array?

- Arrays are appropriate when you need to store a fixed-size collection of elements of the same type and when you need fast access to elements by index.

What are the major disadvantages of the Array?
- Fixed size (can't resize after creation), inefficient for insertions or deletions in the middle or beginning.

Complete the following sentences.

i. **Front** refers to the first item in a queue.

ii. In **Stack**, insertion and deletion is performed at only one end called **the top**.

iii. Time complexity of linear search in the average case for n data items is **O(n).**

Dynamic memory allocation has overcome the drawbacks of static memory allocation. How?

- Dynamic memory allocation allows memory to be allocated at runtime, enabling flexible and efficient memory usage. Unlike static memory allocation, where the size of the data structure is fixed, dynamic memory allocation can grow or shrink as needed, preventing wastage or overflow.

What is the major disadvantage of linked lists in comparison with contiguous lists?

- Linked lists use extra memory for storing pointers, and they require more time for access because you need to traverse the list node by node. In contrast, contiguous lists (like arrays) provide direct access to elements.

How to overcome the false overflow problem? When this problem occurs?

- False overflow occurs in data structures like stacks or queues when they are implemented with static memory, but the structure is not actually full, just incorrectly marked as full due to limitations in how the memory is managed (like with circular buffers). To overcome this, ensure proper management of memory and use dynamic allocation or implement circular data structures correctly to prevent false overflow.

How a node can be inserted in the beginning, middle, and end of a linked list?

- **Beginning:** Create a new node, point it next to the current head, and update the head to this new node.

- **Middle:** Traverse to the desired position and adjust the pointers of the previous node and the new node so the new node points to the next node.

- **End:** Traverse to the last node, create a new node, and set the last node's next to point to this new node. If doubly linked, also update the new node’s previous pointer.

Using stack, evaluate the postfix expression A B C D - * +, where A = 25, B = 2, C = 18 and D=13?

Final result: 35

To evaluate a postfix expression, you follow these steps:
Start from the left, and push operands (A, B, C, D) onto the stack.
When you encounter an operator, pop the required operands from the stack, perform the operation, and push the result back onto the stack.
Postfix expression: A B C D - * +

Push A = 25 onto the stack.
Push B = 2 onto the stack.
Push C = 18 onto the stack.
Push D = 13 onto the stack.
Encounter -: Pop C = 18 and D = 13, calculate 18 - 13 = 5, and push the result (5) onto the stack.
Encounter *: Pop B = 2 and the result of the subtraction (5), calculate 2 * 5 = 10, and push the result (10) onto the stack.
Encounter +: Pop A = 25 and the result of the multiplication (10), calculate 25 + 10 = 35.

Is it easier to insert a new node before or after a specified node in a linked list? Why? It is easier to insert a new node after a specified node in a linked list because you only need to update the pointer of the previous node to point to the new node, and the new node will point to the next node. For insertion before a node, you need to traverse to the previous node, which requires more effort.

List three operations that are possible in Array and Linked list but are not allowed in Stack.

- **Random Access (Array):** Arrays allow direct access to any element by its index, whereas stacks do not.
- **Insertion at Specific Positions (Array, Linked List):** Inserting at any position is allowed in arrays and linked lists but not in stacks (which only allows insertion at the top).
- **Traversal (Array, Linked List):** Both arrays and linked lists allow traversal through all elements, while stacks restrict access to only the top element.

What is a queue? Show two possible techniques of memory allocations for a queue. Discuss their advantages and disadvantages.

A **queue** is a linear data structure that follows the FIFO (First In, First Out) principle. Elements are added at the rear and removed from the front.

Two techniques for memory allocation in a queue:

Static Array Allocation:

Advantages: Simple to implement and provides fast access.

Disadvantages: Fixed size; if the queue is full, no more elements can be added. Wastage of memory may occur if the queue size is too large.

Dynamic (Circular) Array Allocation:

Advantages: Allows efficient use of memory, and the array size can be adjusted as needed. Circular allocation avoids memory wastage by reusing empty spaces when the front elements are dequeued.

Disadvantages: More complex to implement and resizing or managing the wrap-around behavior adds extra overhead.

When linear search algorithm is better than binary search algorithm? Why?

Linear search is better when:

The data is unsorted. Binary search requires sorted data, whereas linear search works with both sorted and unsorted data.

You have a small dataset, where the overhead of sorting or maintaining the order for binary search isn't worth it.

Compare linear search algorithm with Binary search algorithm (Refer to their big O notations)

Linear Search:

Time Complexity: O(n), where n is the number of elements.

It checks each element one by one, so it may need to scan the entire list.

Binary Search:

Time Complexity: O(log n), where n is the number of elements.

It works by repeatedly dividing the list into half and checking only a specific range of elements. This makes it much faster than linear search, but the list must be sorted beforehand.

What is the difference between pointer p = nil and p is undefined?

pointer p = nil: This means that the pointer p is explicitly set to nil (or null), meaning it is intentionally pointing to nothing. It is a known state.
p is undefined: This means the pointer p has not been assigned any value yet. It is in an undefined state and its value is unknown, leading to potential issues like segmentation faults or unpredictable behavior.

What are the impacts of storing huge number of integers in array?

Memory consumption: Storing a large number of integers will require a significant amount of memory. Each integer typically consumes 4 bytes of memory, so large arrays can use a substantial amount of system memory.
Performance issues: If the array becomes too large, it could lead to slower performance due to memory access times, especially if the system is running out of available memory (paging or swapping).
Cache inefficiency: Large arrays may not fit in CPU cache, causing slower access times.

Which data structure you should use if a program keeps track of patients as they check into a medical clinic, assigning patients to doctors on a first-come, first-served basis. The **queue** data structure should be used because it follows the FIFO (First-In, First-Out) principle, which perfectly fits the requirement of handling patients in the order they check in.

Suppose we have an integer-valued stack S and a queue Q. What are final values in the stack S and in the Q after the following operations. Show contents of both S and Q at each step indicated by the line.

```
Stack S;
Queue Q;
int x =10, y=20;
```

S.push(x); S.push(y); S.push(S.pop()+S.pop()); Q.enqueue(x); Q.enqueue(y);
Q.enqueue(S.pop());
S.push(Q.dequeue()+Q.dequeue());

Initial State:

Stack S: (empty)
Queue Q: (empty)
S.push(x): Push 10 onto stack S.
Stack S: 10
Queue Q: (empty)
S.push(y): Push 20 onto stack S.
Stack S: 10, 20
Queue Q: (empty)
S.push(S.pop() + S.pop()):

Pop 20 and 10 from **the stack, perform 10 + 20 = 30, then push the result (30) back onto the stack.**

Stack S: 30
Queue Q: (empty)
Q.enqueue(x): Enqueue 10 onto queue Q.
Stack S: 30

Queue Q: 10
Q.enqueue(y): Enqueue 20 onto queue Q.
Stack S: 30
Queue Q: 10, 20
Q.enqueue(S.pop()): Pop 30 from the stack, then enqueue 30 onto queue Q.
Stack S: (empty)
Queue Q: 10, 20, 30
S.push(Q.dequeue() + Q.dequeue()):
Dequeue 10 and 20 from the queue, perform 10 + 20 = 30, then push 30 onto the stack.
Stack S: 30
Queue Q: 30

Sort the given values using Bubble Sort, indicating the number of passes and the number of comparisons.

70	75	85	60	55	50

Initial Values:
70, 75, 85.60, 55.50

Pass 1:

Compare 70 and 75: no swap needed (70 < 75).
Compare 75 and 85.60: no swap needed (75 < 85.60).

Compare 85.60 and 55.50: swap (85.60 > 55.50).
New array: 70, 75, 55.50, 85.60
After Pass 1, we have bubbled the largest value (85.60) to the end.

Pass 2:

Compare 70 and 75: no swap needed (70 < 75).
Compare 75 and 55.50: swap (75 > 55.50).
New array: 70, 55.50, 75, 85.60
After Pass 2, 75 is in its final position.

Pass 3:

Compare 70 and 55.50: swap (70 > 55.50).
New array: 55.50, 70, 75, 85.60
After Pass 3, the array is fully sorted.

Summary:
Total Passes: 3
Total Comparisons: 6 (3 comparisons per pass for 3 passes)

Sorted Array:

55.50, 70, 75, 85.60

Convert the expression ((A + B) * C - (D - E) ^ (F + G)) to equivalent Prefix and Postfix notations.

Prefix : - * + A B C ^ - D E + F G **Postfix:** A B + C * D E - F G + ^ -

Consider the following stack of characters, where STACK allocates memory size for 8 characters. STACK: A, C, D, F, K, ___, ___, ___ where "___" denotes an empty stack cell. Describe the stack as the following operations take place:

Initial Stack:

STACK: A, C, D, F, K, ___, ___, ___

There are 5 elements in the stack initially (A, C, D, F, K), and 3 empty spaces.

(a) POP():
This operation removes the top element from the stack, which is K.
STACK: A, C, D, F, ___, ___, ___, ___

(b) POP():
This operation removes the next top element, which is F.
STACK: A, C, D, ___, ___, ___, ___, ___

(c) PUSH(L):
This operation pushes L onto the top of the stack.
STACK: A, C, D, L, ___, ___, ___, ___

(d) PUSH(P):
This operation pushes P onto the top of the stack.
STACK: A, C, D, L, P, ___, ___, ___

(e) POP():
This operation removes the top element, which is P.
STACK: A, C, D, L, ___, ___, ___, ___

(f) PUSH(R):
This operation pushes R onto the top of the stack.
STACK: A, C, D, L, R, ___, ___, ___

(g) PUSH(S):
This operation pushes S onto the top of the stack.
STACK: A, C, D, L, R, S, ___, ___

(h) POP():
This operation removes the top element, which is S.
STACK: A, C, D, L, R, ___, ___, ___

Final Stack:
STACK: A, C, D, L, R, ___, ___, ___

Suppose STACK is allocated memory space for 6 integers and initially its top = -1. Show stack behavior and find the output of the following pseudo code:

```
1. a := 2; b := 5;
2. push (a);
3. push (b+2);
4. push (9);
5. while (top <> -1)
6. { pop ( item);
7. print item; }
```

Answer:

1. a := 2; b := 5;
2. a = 2
3. b = 5
4. push(a);
5. Stack: [2]
6. Top: 0
7. push(b + 2);
8. Stack: [2, 7]
9. Top: 1
10. push(9);
11. Stack: [2, 7, 9]
12. Top: 2
13. while (top <> -1) { pop(item); print item; }

While Loop Execution:

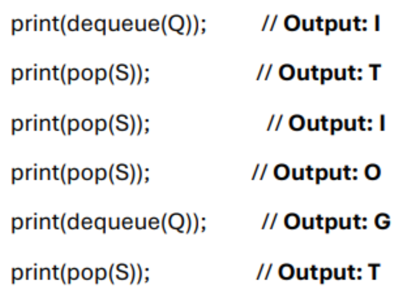
1. First Iteration:
2. Pop: 9
3. Print: 9
4. Stack: [2, 7]
5. Top: 1
6. Second Iteration:
7. Pop: 7
8. Print: 7
9. Stack: [2]
10. Top: 0
11. Third Iteration:
12. Pop: 2
13. Print: 2
14. Stack: []
15. Top: -1

Output: 9 7 2

Assume that you have a stack S, a queue Q, and the standard stack - queue operations: push, pop, enqueue and dequeue. Assume that print is a function that prints the value of its argument. Execute the operations below to show only the output of each print function.

```
push(S, ‘T’);
enqueue(Q, ‘I’);
push(S,dequeue(Q));
enqueue(Q, ‘I’);
enqueue(Q, ‘G’);
print(dequeue(Q));
enqueue(Q, T);
push(S, ‘I’);
push(S, dequeue(Q));
print(pop(S));
enqueue(Q, pop(S));
push(S, ‘O’);

print(pop(S));
enqueue(Q, ‘O’);
print(dequeue(Q));
enqueue(Q, pop(S));
push(S, dequeue(Q));
print(pop(S));
print(pop(S));
```



```
print(dequeue(Q));      // Output: I
print(pop(S));          // Output: T
print(pop(S));          // Output: I
print(pop(S));          // Output: O
print(dequeue(Q));      // Output: G
print(pop(S));          // Output: T
```

Consider the following queue QUEUE is allocated 6 memory cells:

FRONT= 1, REAR = 4 and

QUEUE: ___, London, Berlin, Rome, Paris, ___.

Describe queue’s behavior, including FRONT and REAR, as the following operations take place:

(a) "Athens" is added   (b) Two cities are deleted

(c) "Moscow" is added   (d) "Madrid" is added

(a) "Athens" is added (Enqueue operation)

Operation: Add "Athens" to the queue.
Update: The value is added to the position after Paris (i.e., at index 5).
FRONT = 1, REAR = 5
Result: QUEUE: ___, London, Berlin, Rome, Paris, Athens

(b) Two cities are deleted (Dequeue operation)

Operation: Remove two cities from the front of the queue (starting from FRONT = 1).
First dequeue: Remove London from the queue.
Second dequeue: Remove Berlin from the queue.
FRONT = 3, REAR = 5
Result: QUEUE: ___, ___, Rome, Paris, Athens, ___

(c) "Moscow" is added (Enqueue operation)

Operation: Add "Moscow" to the queue.
FRONT = 3, REAR = 6
Result: QUEUE: ___, ___, Rome, Paris, Athens, Moscow

(d) "Madrid" is added (Enqueue operation)

Enqueue Operation: Add "Madrid" to the queue. However, since the queue is already full (6 memory cells), this operation will fail. A queue overflow occurs in this case.

Give some examples where the data structures Stack and Queue are used in computer when executing a program.

Examples where Stack is used:

– **Function Call Stack:** When a function is called in a program, the system stores its information (parameters, return address, etc.) on the call stack. When the function returns, this information is popped off the stack.
– **Undo Operations in Text Editors**: Text editors like Word use a stack to store actions (such as typing or deleting text), allowing users to undo or redo those actions.

Examples where Queue is used:

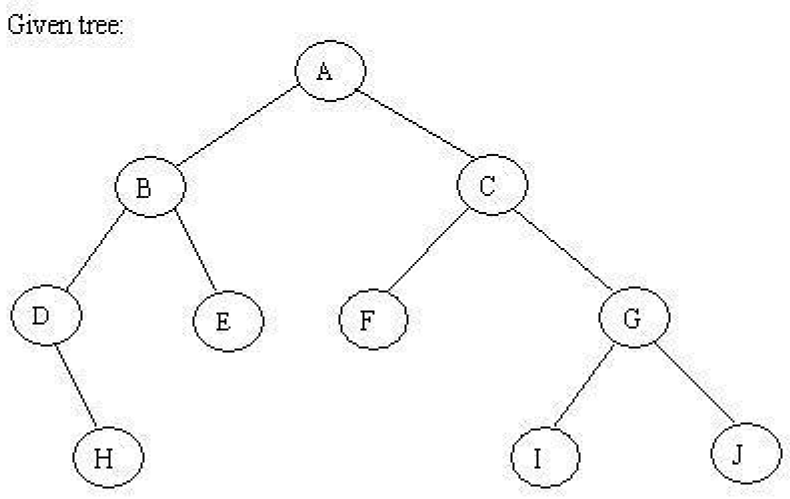
– **Task Scheduling:** Operating systems use queues to manage tasks in a multi-tasking environment. Processes are scheduled to run in a FIFO (First In, First Out) manner.
– **Print Spooling:** In a printer queue, print jobs are processed in the order they arrive, ensuring that the first job is printed first.

How many different trees are possible with 3 nodes ?

$$\boxed{C(3) = \frac{1}{3+1}\binom{6}{3} = \frac{1}{4} * 20 = 5}$$

There are **5 different binary trees** possible with 3 nodes.

Traverse the given following tree using Inorder, Preorder and Postorder traversals.



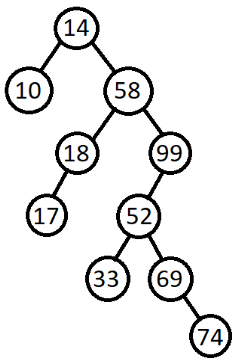
Inorder Traversal: **<u>D, H, B, E, A, F, C, I, G, J</u>**

Preorder Traversal: **<u>A, B, D, H, E, C, F, G, I, J</u>**

Postorder Traversal: **<u>H, D, E, B, F, I, J, G, C, A</u>**

Draw the binary search tree that would result from the insertion of the following integer keys:

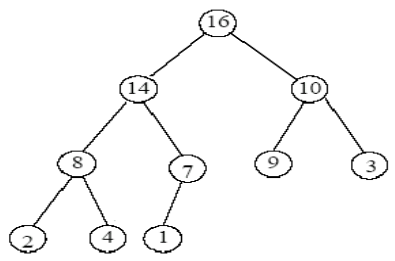
14, 58, 18, 10, 99, 52, 33, 69, 74, 17



What is the maximum number of nodes of a binary tree with height 3? **<u>15</u>**

<u>Maximum number of nodes = 2^ (3+1) - 1 = 2^4 - 1 = 16 - 1 = 15</u>

Given the following tree T, answer the following questions and give a reason for your answer in case of (Yes/ No)



1. Is T a binary tree?

Answer: **Yes**, T is a binary tree. A binary tree is a tree in which each node has at most two children, and T follows this property

2. Is T a binary search tree?

Answer: **No**, T is not a binary search tree. In a BST, for each node, the value of its left subtree nodes must be smaller, and the value of its right subtree nodes must be greater. In T, the left and right subtrees of some nodes violate this condition, so it is not a BST.

3. Is T a max-heap?

Answer: **No**, T is not a max-heap. In a max-heap, for every node, the value of the node must be greater than or equal to the values of its children. In T, the parent nodes (16, 14, 10, etc.) are not always greater than their children, so it doesn't satisfy the max-heap property.

4. Is T full?

Answer: **No**, T is not a full binary tree. A full binary tree is one where every node has either 0 or 2 children. In T, the node with value 9 has only one child (3), so it is not full.

5. Is T complete?

Answer: **No**, T is not a complete binary tree. A complete binary tree is one where all levels are filled except possibly for the last level, which should be filled from left to right. The tree does not satisfy this condition.

6. Represent data nodes of T using an array data structure.

Answer: int T[10] = [16, 14, 10, 8, 7, 9, 3, 2, 4, 1];

Consider the following contiguous implementation of a tree. Answer the following questions:

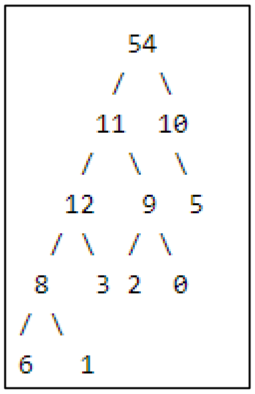
a. What is the left child of 2? Left Child of 2 is 3

b. What is the parent of 2? Parent of 2 is 8

$\boxed{15}\ \boxed{8}\ \boxed{20}\ \boxed{2}\ \boxed{11}\ \boxed{30}\ \boxed{27}\ \boxed{13}\ \boxed{6}\ \boxed{18}\ \boxed{12}$

Build the heap from the following items. Show should be done to keep the heapness (structure and order) properties of the tree.

11 54 10 2 9 5 8 3 12 0 6 1



```
          54
         /  \
       11   10
      /  \   \
    12    9   5
   / \   / \
  8   3 2   0
 / \
6   1
```

Draw the parse tree of the expression R = A * (B – C) + 2 * (3 + B - C)

```
        =
      /   \
    R       +
          /   \
        *       *
      / \     / \
     A   -   2   +
        / \     / \
       B   C   3   -
                  / \
                 B   C
```

Here is a small binary tree of integers that is needed for the next seven questions.

```
        14
      /    \
     2      19
    / \     / \
   1   5   15  30
        \     /  \
       22   40
```

a. The node with the value 12 is an parent of the node with the value 2 (True/False). False
b. The node with the value 40 is a child of the node with the value 15 (True/False). True
c. The tree is a complete tree (True/False). False
d. The tree is a complete tree (True/False). False
e. What is the depth of the tree? 4
f. What is the order of nodes visited using a pre-order traversal? 14, 2, 1, 5, 12, 19, 15, 30, 22, 40
g. What is the order of nodes visited using a post-order traversal? 1, 12, 5, 2, 15, 22, 40, 30, 19, 14
h. We remove the root, what will be the new tree?

```
         19
      /    \
     2      30
    / \     / \
   1   5   15  40
         \    /
        12 22
```

i. We add the element 21, what will be the new tree?

```
     14
   /    \
  2      19
 / \     / \
1   5   15  30
     \     /  \
     12   22  40
           /
         21
```

What is the complexity of binary search algorithm? Justify.

<u>$O(\log_2 n)$</u>

Binary search divides the input array into half at every step and discards one half of the elements, leaving only the relevant half to search in.The number of steps required to reduce n elements to 1 is logarithmic to the base 2, i.e., $\log_2 n$.This makes the time complexity **$O(\log_2 n)$**, where n is the size of the input.

Compute the time complexity of the following code:

```
sum := 0
for (int i = 0 ; i < n ; i++)
      for (int j = 0 ; j < m ; j++)
            sum := sum + i + j
```

Answer: $O(n * m)$

The outer loop runs n times. For each iteration of the outer loop, the inner loop runs m times. Inside the inner loop, there is a constant time operation (sum := sum + i + j). Total operations = n * m. Thus, the time complexity is $O(n * m)$.

- **Arrange the following functions, often used to represent complexity of algorithms in order from slowest to fastest**

$O(1)$, $O(n)$, $O(n*\log_2 n)$, $O(\log_2 n)$, $O(n^2)$, $O(2^n)$

$O(2^n)$, $O(n^2)$, $O(n * \log_2 n)$,$O(n)$,$O(\log_2 n)$,$O(1)$

How the performance of an algorithm is measured?

The performance of an algorithm is typically measured based on:

- **Time Complexity**: The time an algorithm takes to complete relative to the size of the input (n)
- **Space Complexity:** The amount of memory or storage an algorithm uses during execution. Both are expressed using **Big-O notation**, which gives an upper-bound performance estimate.

Why time and space are the most important considerations in computer operation?

- **Time:** Efficient algorithms reduce execution time, improving responsiveness and productivity, especially in large-scale or real-time applications.
- **Space:** Memory is limited, and inefficient algorithms may exhaust memory or require more resources, leading to failures.

Why is the complexity of an algorithm generally more important than the speed of the processor?

- **Processor speed is fixed**: **Processor speed is fixed:** Even the fastest processor cannot compensate for an algorithm with poor complexity. For example, an $O(2^n)$ algorithm will remain slow for large inputs, regardless of processor speed.

What are the factors you will consider to select the proper data structures and memory allocation mechanism when building an algorithm?

- **Nature of the data:** Whether the data is static or dynamic, structured, or unstructured.
- **Operations required:** The type of operations needed, like insertion, deletion, searching, sorting, or traversal.
- **Time constraints:** How quickly the algorithm must execute for acceptable performance.
- **Space constraints:** The amount of memory available for the data structure and algorithm.
- **Complexity of access:** Whether random access (e.g., arrays) or sequential access (e.g., linked lists) is needed.
- **Scalability:** The ability to handle increasing input sizes without significant performance degradation.

Sample for MCQ

__C___ 1.Two main measures for the efficiency of an algorithm are_

A. Processor and memory

B. Complexity and capacity

C. Time and space

D.	Data and space

__B___ **2. The time factor when determining the efficiency of algorithm is measured by**

A.	Counting microseconds

**B.	Counting the number of key operations**

C.	Counting the number of statements

D.	Counting the kilobytes of algorithm

___A__ **3. The space factor when determining the efficiency of algorithm is measured by**

**A.	Counting the maximum memory needed by the algorithm**

B.	Counting the minimum memory needed by the algorithm

C.	Counting the average memory needed by the algorithm

D.	Counting the maximum disk space needed by the algorithm

___D__ **4. Which of the following case does not exist in complexity theory**

A.	Best case

B.	Worst case

C.	Average case

**D.	Null case**

___D__ **5. The Worst case occur in linear search algorithm when_**

A.	Item is somewhere in the middle of the array

B.	Item is not in the array at all

C.	Item is the last element in the array

D.   Item is the last element in the array or is not there at all

__A___6. The Average case occur in linear search algorithm_

A.   When Item is somewhere in the middle of the array

B.   When Item is not in the array at all

C.   When Item is the last element in the array

D.   When Item is the last element in the array or is not there at all

___A__7. The complexity of the average case of an algorithm is

A.   Much more complicated to analyze than that of worst case

B.   Much more simpler to analyze than that of worst case

C.   Sometimes more complicated and some other times simpler than that of worst case

D.   None or above

__A___8. The complexity of linear search algorithm is

A.   O(n)

B.   O(log n)

C.   O(n2)

D.   O(n log n)

__B___9. The complexity of Binary search algorithm is

A.   O(n)

B.   O(log n)

C.   O(n2)

D.   O(n log n)

___**C**___10. The complexity of Bubble sort algorithm is

A. O(n)

B. O(log n)

C. O(n2)

D. O(n log n)

___**D**___11. The complexity of merge sort algorithm is

A. O(n)

B. O(log n)

C. O(n2)

D. O(n log n)

___**D**___12.Which of the following data structure is not linear data structure?

A. Arrays

B. Linked lists

C. Both of above

D. None of above

___**C**___13.Which of the following data structure is linear data structure?

A. Trees

B. Graphs

C. Arrays

D. None of above

___**D**___**14.The operation of processing each element in the list is known as**

A. Sorting

B. Merging

C. Inserting

D. Traversal

___**B**___**15.Finding the location of the element with a given value is:**

A. Traversal

B. Search

C. Sort

D. None of above

___**A**___**16.Arrays are best data structures**

A. for relatively permanent collections of data

B. for the size of the structure and the data in the structure are constantly changing

C. for both of above situation

D. for none of above situation

___**B**___**17.Linked lists are best suited**

A. for relatively permanent collections of data

B. for the size of the structure and the data in the structure are constantly changing

C. for both of above situation

D. for none of above situation

___C___18.Each array declaration need not give, implicitly or explicitly, the information about

A. the name of array

B. the data type of array

C. the first data from the set to be stored

D. the index set of the array

___A___19.The elements of an array are stored successively in memory cells because_

A. by this way computer can keep track only the address of the first element and the addresses of other elements can be calculated

B. the architecture of computer memory does not allow arrays to store other than serially

C. both of above

D. none of above

___A___20. Inserting an item into the stack when stack is not full is called …………. Operation and deletion of an item form the stack, when stack is not empty is called ……….operation.

A) push, pop

B) pop, push

C) insert, delete

D) delete, insert

___B___ 21. Is a pile in which items are added at one end and removed from the other.

A) Stack

B) Queue

C) List

D) None of the above

___**A**___22. is very useful in situation when data have to stored and then retrieved in reverse order.

A) Stack

B) Queue

C) List

D) Link list

___**A**___23. Stack is also called as

A) Last in first out

B) First in last out

C) Last in last out

D) First in first out

___**D**___24.The five items: A, B, C, D and E are pushed in a stack, one after the other starting from A. The stack is popped four times, and each element is inserted in a queue. Then two elements are deleted from the queue and pushed back on the stack. Now one item is popped from the stack.

The popped item is.

A) A

B) B

C) C

D) D

___**A**___25. If post order traversal generates sequence xy-zw*+, then label of nodes 1,2,3,4,5,6,7 will be

A. +, -, *, x, y, z, w

B. x, -, y, +, z, *, w

C. x, y, z, w, -, *, +

D. -, x, y, +, *, z, w