Officer(IT)- 04.10.2024 Solution
This Questions are important for BUET, BPSC, IBA or MIST Pattern

1	**You are required to convert a 32 bit digital number to an analogue voltage over the voltage range of 0 to 3.3V with a Digital to Analogue converter (DAC). What is the resolution of the analogue output ?**

Resolution = $\frac{Volage\ range}{Number\ of\ steps} = \frac{3.3}{2\text{^}12}$ = 0.00080586 Volt = 0.080586mV.

2	**Find the sets X and Y If XUY = {1,2,3,5,6,8,9,10}, X∩ $Y = \{1,5\}, Y - X = \{2,6,9,10\}$**

Using Bangla for Clarification:

Y-X এর মানে হলো একটি সেটের উপাদান যেখানে Y এ আছে কিন্তু X এ নেই।
X∩ Y, X এবং Y উভয়তেই আছে,
{2,6,9,10} U {1,5}
X অবশ্যই X∩ Y এ আছে এবং Y-X এ নাই। আমরা দেখেছি XUY = {1,2,3,5,6,,8,9,10} এ (Y-X) এর element এর মধ্যে {3,8} অনুপস্থিত তাই ,
(X∩ Y) U {3,8} = {1,5} U {3,8} = {1,3,5,8} Answer.

3	**Convert the infix expression P = 12/(7-3)+2 to postfix expression and evaluate it.**

SL No.	Expression	Stack	Postfix
0		(	
1	P	(	P
2	=	(	P =
3	12	(	P = 12
4	/	(/	P = 12
5	(	(/(	P = 12
6	7	(/(	P = 12 7
7	-	(/(-	P = 12 7
8	3	(/(	P = 12 7 3
9	)	(/	P = 12 7 3 -
10	+	(+	P = 12 7 3 - /
11	2	(+	P = 12 7 3 - / 2
12	)		P = 12 7 3 - / 2 +

Postfix Evaluation:

SL	Symbol	Stack
1	12	12
2	7	12,7
3	3	12,7,3
4	-	12,4
5	/	3
6	2	3,2
7	+	5
Result		5

DO NOT COPY

4	**Explain how encapsulation and inheritance are advantageous in Object Oriented Programming?**

Encapsulation that involves grouping related data and functions within a single unit called a class. This concept allows for data hiding, as the details of a class are hidden from the other parts of the program, and can only be accessed through methods specifically defined for that purpose.

Benefits:

By implementing encapsulation, we can achieve a modular and maintainable software structure. Encapsulation ensures that the internal workings of a class are protected from unintentional changes while providing a clearly defined interface to interact with the class.

To better understand encapsulation, let's explore some key programming concepts related to encapsulation:

- **Data hiding:** This is the concept of hiding the internal details of a class and only exposing the necessary functionality. This is achieved by using private, public and protected **access modifiers**.
- **Modifiers: These** are keywords that control the visibility of class members. In many object-oriented **programming languages**, such as **Java** and C++, they include:
  - **Public:** Members can be accessed from anywhere in the program.
  - **Private:** Members can only be accessed within the class they are declared.
  - **Protected:** Members can be accessed within the class and its derived classes.
- **Abstraction:** This is the process of simplifying complex systems by breaking them into smaller and more manageable components. Encapsulation supports abstraction by hiding the complexities of a class behind a simple interface.
- **Getters and setters:** Also known as accessor and mutator methods, getters and setters allow for the controlled access and modification of class variables while maintaining encapsulation.

DO NOT COPY

Key benefits of inheritance:

- Code Reusability:

By inheriting from a parent class, a child class automatically gains access to all its attributes and methods, eliminating the need to re-write the same code again, saving time and effort.

- Hierarchical Organization:

Inheritance establishes a clear hierarchy between classes, where a child class is a specialized version of its parent class, making the relationships between different classes easier to understand.

- Improved Maintainability:

If changes need to be made to common functionality shared across multiple classes, modifying the parent class updates all inheriting child classes simultaneously, ensuring consistency.

- Modularity:

Inheritance promotes modularity by breaking down complex systems into smaller, reusable components, allowing developers to focus on specific functionalities within each class.

- Extensibility:

Child classes can add new features or override existing methods from the parent class to customize behavior while still leveraging the base functionalit

For Example:

Imagine creating a class called "Vehicle" with basic attributes like "color" and "speed" and methods like "accelerate" and "brake." Then, you can create separate classes for "Car" and "Motorcycle" which inherit from "Vehicle," inheriting the core functionality and adding specific properties like "numberOfDoors" (for Car) or "handlebarType" (for Motorcycle).

5	**If you throw two unbiased dice (each with six sides) together, what is the proabability that sum of two upward faces will be 7? Explain your answer.**

n(S) = 36
Let A = sum of numbers is 7 = { (1, 6)(2, 5)(3, 4) (4, 3) (5, 2) (6, 1)}
n(A) = 6
P (Sum of numbers is 7) = $\frac{n(A)}{n(S)}$
= $\frac{6}{36} = \frac{1}{6}$
Therefore, the probability of rolling two dice and getting a sum of 7 is $\frac{1}{6}$.

6	**How many total bits are required of a Direct-Mapped cache with 16 KB of data and 4-word blocks, assuming a 32-bit address (word addressable) and 32-bit words? [BUET MSC 21] [Officer IT'24] [B.B AME'2023]**

DO NOT COPY

Total Cache Size = 2^n* (block size 2^m*32 + tag size + valid field size)
= 2^n * (2^m*32 + 32 – (n+m+z)+1) bits
No of words = 16 kb/4 blocks= 2^{12} words (as 1 word = 1B byte addressable)
No of blocks = 2^{12} /2^2 = 2^{10} blocks
Tag = 32 – (10+2+2) = 18 bits (n = 10, m =2)
Valid = 1 bit
Total Cache Size = 2^{10} * (4*32+18+1) = 2^{10}*147=147KB
Total 147Kb are required of a direct mapped cache.

7	**Give the necessary condition for deadlock occur. Is it possible to have deadlock involving only a single process ? Explain your answer?**

Necessary condition for deadlock:
Mutual Exclusion
Circular Wait
Hold and Wait
No Preemption

No, it is not possible to have a deadlock with only a single process. Here's why:

- In a deadlock, **circular wait** is a necessary condition. A single process cannot create a circular wait, as there is no other process for it to form a cycle with.
- Deadlock typically involves **multiple processes** where each process holds some resources and is waiting for others, creating a cycle of dependencies.

8	**Consider the following buy a product description. Customer browses catalog, selects items to buy and then goes to check out. Customer fills in shipping information (address, receive time). System presents full pricing information and customer fills in credit card information. System authorizes purchase,**

	confirms sale and sends confirming email to customer. Draw a use case diagram for the above system.

Actors:

1. **Customer** – Browses catalog, selects items, provides shipping info, and completes purchase.
2. **System** – Processes the payment, provides pricing, and sends email confirmation.

 Browse **Catalog** – The customer views available products.
 Select **Items** – The customer adds products to the cart.
 Checkout – The customer proceeds to payment and shipping information.
 Provide **Shipping Information** – Customer enters their address and desired receive time.
 View **Pricing Information** – The system shows the total price, including shipping, taxes, etc.
 Provide **Payment Information** – Customer enters credit card information.
 Authorize **Payment** – The system processes the payment with the credit card.
 Confirm **Sale** – The system confirms the purchase and displays a confirmation.
 Send **Confirmation Email** – The system sends an email to the customer.

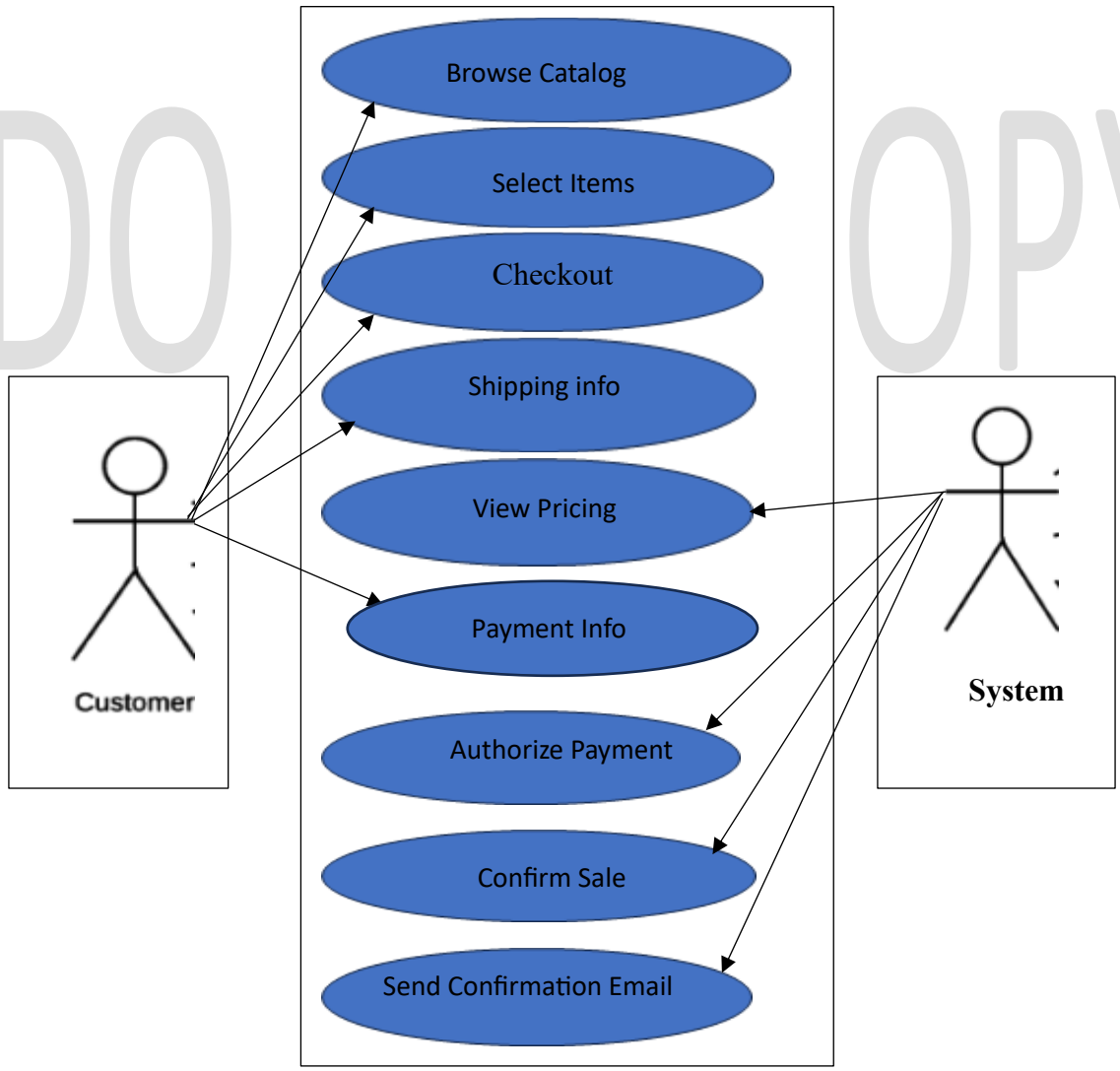


Figure-1: Use Case Diagram of Online Shopping

	Suppose we want to develop software for a graphic package and we are given the task to implement circle class. The circle class has to be translatable from its origin. And it should also be able to give perimeter and area of the circle. Identify the data and method requirements for the class and give the data flow of translation method.

Answer:

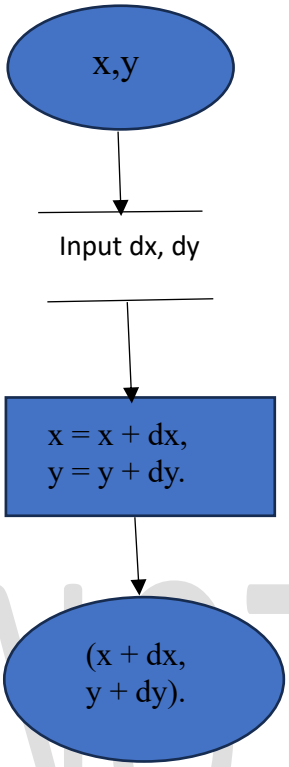
```
#include <iostream>
#include <cmath>
using namespace std;
class Circle {
private:
   float x, y;
   float radius;

public:
   Circle(float x = 0, float y = 0, float r = 1) {
      this->x = x;
      this->y = y;
      this->radius = r;
   }
   void translate(float dx, float dy) {
      x += dx;
      y += dy;
   }
   float getPerimeter() const {
      return 2 * M_PI * radius;
   }

   float getArea() const {
      return M_PI * radius * radius;
   }
   void printInfo() const {
      cout << "Circle Center: (" << x << ", " << y << ")" << endl;
      cout << "Radius: " << radius << endl;
      cout << "Perimeter: " << getPerimeter() << endl;
      cout << "Area: " << getArea() << endl;
   }
};
int main() {
   Circle circle(5, 5, 10);
   cout << "Initial Circle Information:" << endl;
   circle.printInfo();
   circle.translate(3, 4);
   cout << "\nAfter Translation (by 3, 4):" << endl;
   circle.printInfo();
   return 0;
}
```

DO NOT COPY

Start: Current center coordinates (x, y).
Input: Translation values dx and dy.
Process: Update x and y to new values: x = x + dx, y = y + dy.
End: Circle moves to new coordinates (x + dx, y + dy).



Data Flow of Translation Method:

DO NOT COPY

	Using an explanation of the difference between flow-control and congestion control, Discuss the impact of a stable end-to-end latency.

Flow Control and Congestion Control are mechanisms used to manipulate data transmission in networks, but they operate on exceptional tiers and for different purposes:

Flow Control:

Ensures that the sender does now not crush the receiver with extra records than it can deal with.

It works through coping with the rate at which facts is despatched to suit the receiver's potential. For instance, the receiver would possibly ship remarks to the sender indicating how a great deal more records it is able to take delivery of. This mechanism is generally carried out at the link layer or shipping layer (like TCP with window size).

Impact of Latency: In a machine with stable quit-to-quit latency, flow manage may be more predictable. Since latency influences the round-journey time (RTT) of acknowledgments, a solid latency way float manipulate mechanisms can modify sending prices successfully, making sure the receiver isn't crushed. Fluctuations in latency might complicate this, as senders might mistakenly assume the receiver is ready for extra information whilst it isn't always.

Congestion Control:

Manages the overall community traffic to keep away from overwhelming network sources, inclusive of routers and intermediate switches. Congestion manage ensures that the network doesn't emerge as overloaded, which could cause packet loss, accelerated delays, or throughput collapse.
Commonly applied in protocols like TCP (e.G., with algorithms like TCP Reno or TCP Cubic), congestion manage dynamically adjusts the statistics transmission rate based totally on signs and symptoms of congestion within the community, which includes packet loss or growing delays.
Impact of Latency: Stable cease-to-cease latency can imply that the network isn't experiencing congestion. If latency begins to vary or growth extensively, it could be a sign that congestion is building up. Therefore, congestion manage algorithms rely heavily on the observation of latency variations (along with other factors like packet loss) to adjust the transmission rate and prevent congestion.

Impact of Stable End-to-End Latency
A solid end-to-stop latency has a tremendous impact on both drift manipulate and congestion manipulate mechanisms for the following reasons:

Efficiency of Flow Control:

DO NOT COPY

With solid latency, the round-ride time (RTT) for acknowledgments is predictable, allowing the sender to tempo its transmissions correctly. A stable RTT enables the glide manipulate window (e.G., TCP window size) to be adjusted extra as it should be. This minimizes the probabilities of buffer overflows on the receiver, leading to fewer retransmissions and higher typical throughput.
Congestion Detection and Avoidance:

In congestion manipulate, adjustments in latency are regularly used as early caution signs of congestion. When latency is stable, it shows that there's no congestion or minimal congestion in the community. On the alternative hand, growing latency generally indicators congestion, prompting congestion manage algorithms to gradual down the transmission fee. A strong latency surroundings reduces the chance of congestion episodes, allowing for more constant and higher information throughput.
Predictable Transmission Rates:

A stable latency lets in for extra predictable data transmission quotes. Congestion control algorithms, particularly the ones the usage of latency as a metric, can perform extra effectively on account that they don't want to modify for constantly fluctuating community situations. This ends in more top of the line community utilization and less common want for congestion mitigation movements like decreasing the transmission charge.
Lower Packet Loss:

In environments in which stop-to-cease latency is strong, there may be less likelihood of packet loss because of congestion. This enhances the overall reliability of facts transmission, as fewer packets need to be retransmitted, main to more efficient use of network resources.

	Explain the difference between a singly linked list and a doubly linked list.

No.	**Singly linked list**	**Doubly linked list**
1	For singly linked lists, the complexity of insertion and deletion is O(n)	For doubly linked lists, the complexity of insertion and deletion is O(1)
2	The Singly linked list comprises two parts: data and link.	The doubly linked list consists of three parts: data and two pointers.
3	It allows for one-way traversal of elements.	It allows for two-way traversal.
4	Singly linked lists are often used to implement stacks.	Doubly linked lists can be used to implement binary trees, heaps, and stacks.
5	Singly linked lists are preferred when memory conservation is important, and search operations are not required.	Doubly linked lists are preferred for better implementation during search operations.
6	Singly linked lists use less memory compared to doubly linked lists.	Doubly linked lists use more memory compared to singly linked lists.

DO NOT COPY

	Describe and estimate the costs of procedure to insert a new item into an existing binary max heap.

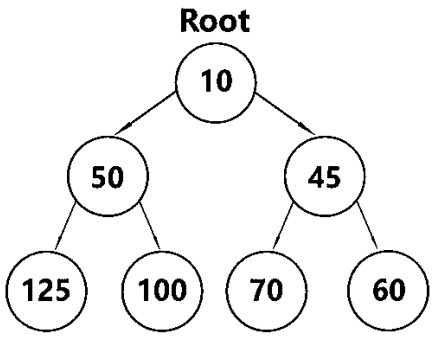
here is the pseudocode for **Max-Heapify algorithm**
A is an array , index starts with 1. and i points to root of tree.

```
1. Max-Heapify(A, i)
2. {
3. left = 2*i;
4. right = 2*i +1;
5. if(left <= A.heap_size() and A[left] > A[i])
6.         largest = l;
7. else largest = i;
8.
9. if(right <= A.heap_size() and A[right] > A[largest])
10.              largest = r;
11.
12. if(largest != i)
13.              swap(A[largest],A[i])
14.              Max-Heapify(A,largest)
15. }
```

It's time complexity is **O(logn).** where n is the number of elements in the subtree for which i is root.
i.e, **n = A.heap_size − i.**

inserting one by one element of a empty heap/ inserting an element at the end takes O(1) time.

10	50	45	125	100	70	60

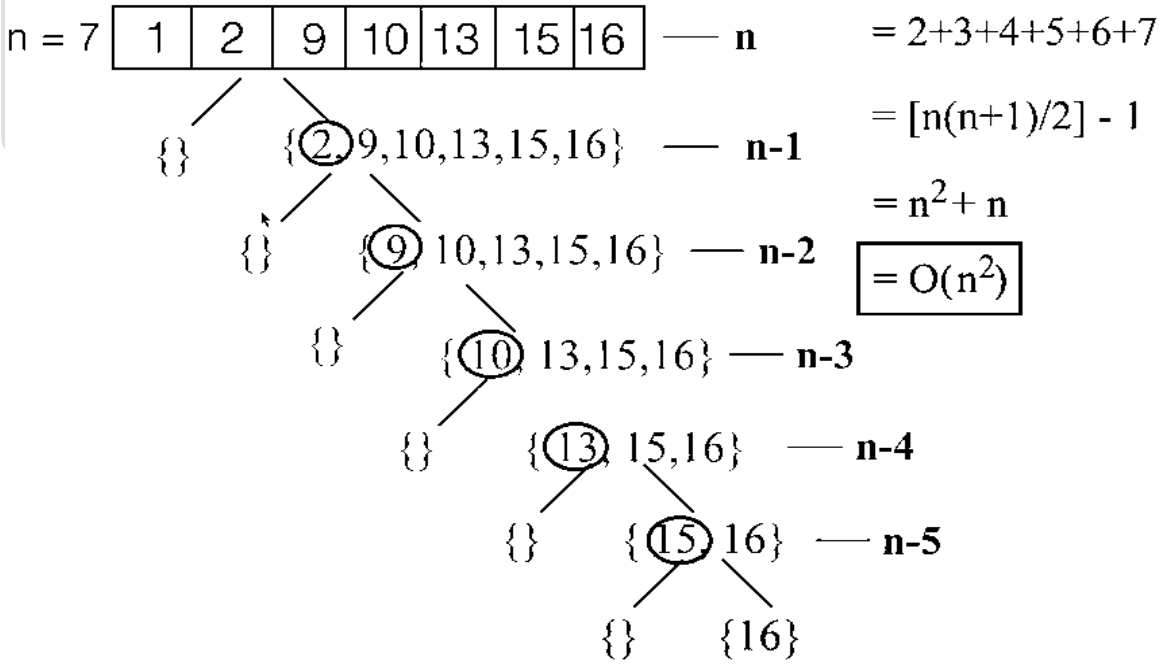


	What is the worst-case time and space complexity of quicksort algorithm. Explain how this worst case behavior can occur?

Worst Case time complexity is $O(n^2)$.

Worst-case scenario: O(n) due to unbalanced partitioning leading to a skewed recursion tree requiring a call stack of size O(n).
Best-case scenario: O(log n) as a result of balanced partitioning leading to a balanced recursion tree with a call stack of size O(log n).

When the array is already sorted worst case time complexity can occur. The picture below depicted how this can be calculated.



	Write a function which receives an array of integers as parameter and print the numbers divisible by 3 in the array.

#include <iostream>

```
using namespace std;
void printDivisibleBy3(int arr[], int size) {
   cout << "Numbers divisible by 3: ";
   for (int i = 0; i < size; i++) {
      if (arr[i] % 3 == 0) {
         cout << arr[i] << " ";
      }
   }
   cout << endl;
}
int main() {
   int arr[] = {10, 12, 15, 23, 30, 33, 40};
   int size = sizeof(arr) / sizeof(arr[0]);
   printDivisibleBy3(arr, size);
   return 0;
}
```

Output: Numbers divisible by 3: 12 15 30 33

DO NOT COPY

	Consider you are given a database of a pet society with the following relations: **Animals (ID: integer, Name: string, PrevOwner: string, DateAdmitted: date, Type: string** **Adopter (PSIN: integer, Name: string, Address: string, OtherAnimals: integer** **Adoption(AnimalID: integer, PSIN: integer, AdoptDate: date, chipNo: integer)** **Give a sql query that list total number of adoptions on June 30, 2024 for each animal type**

```
SELECT A.Type, COUNT(*) AS TotalAdoptions
FROM Animals A
JOIN Adoption AD ON A.ID = AD.AnimalID
WHERE AD.AdoptDate = '2024-06-30'
GROUP BY A.Type;
```