SH1 2020 Promo Paper 1 Sample Solutions

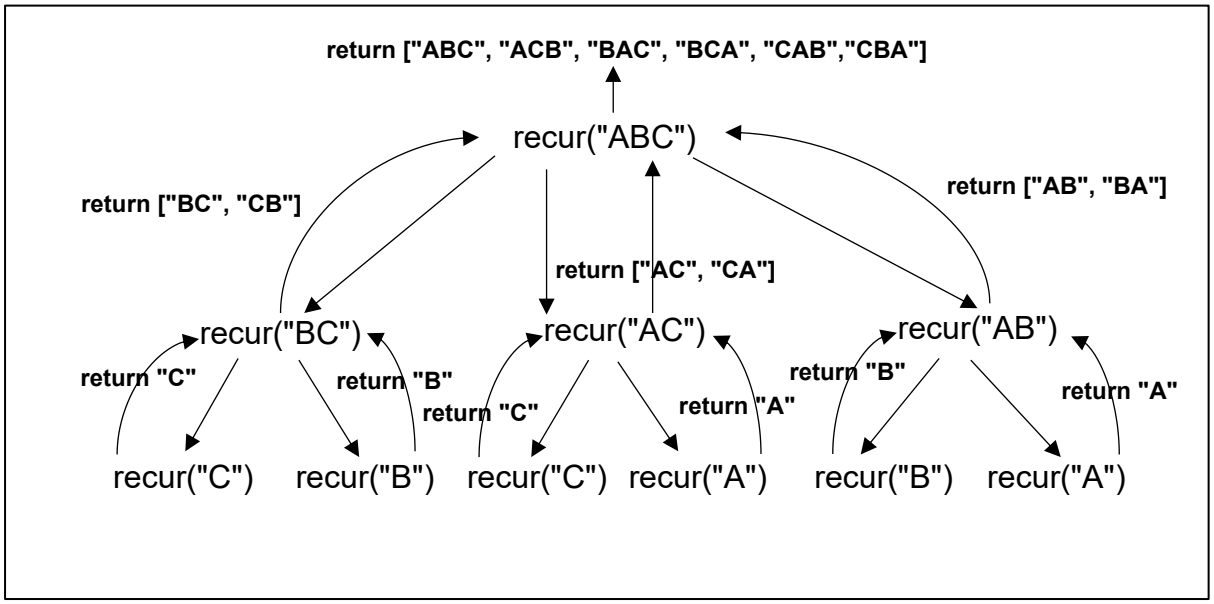
Q1a[2]

- **Calls itself after reducing the problem size which will eventually converges to a base case**
- **Has a base case that returns a solution**

Q1b[2]

- **iterative solution uses a loop which is repeatedly executed until a condition is met while a recusrive solution calls itself till a consition (base case) is met.**
- **OR recusrion requires multiple fraames in the call stack, iterative solution excutes within a single frame in the call stack.**

Q1c[4]



1st level call [1]

2nd level call [1]

Correct return values from all 3 1st level recursive calls [1]

Correct return values for final result [1]

Q1d [1]

Return all the permutations of the string "ABC"

Return all the possible 3 letters combination for the letters "A","B","C"

Any [1]

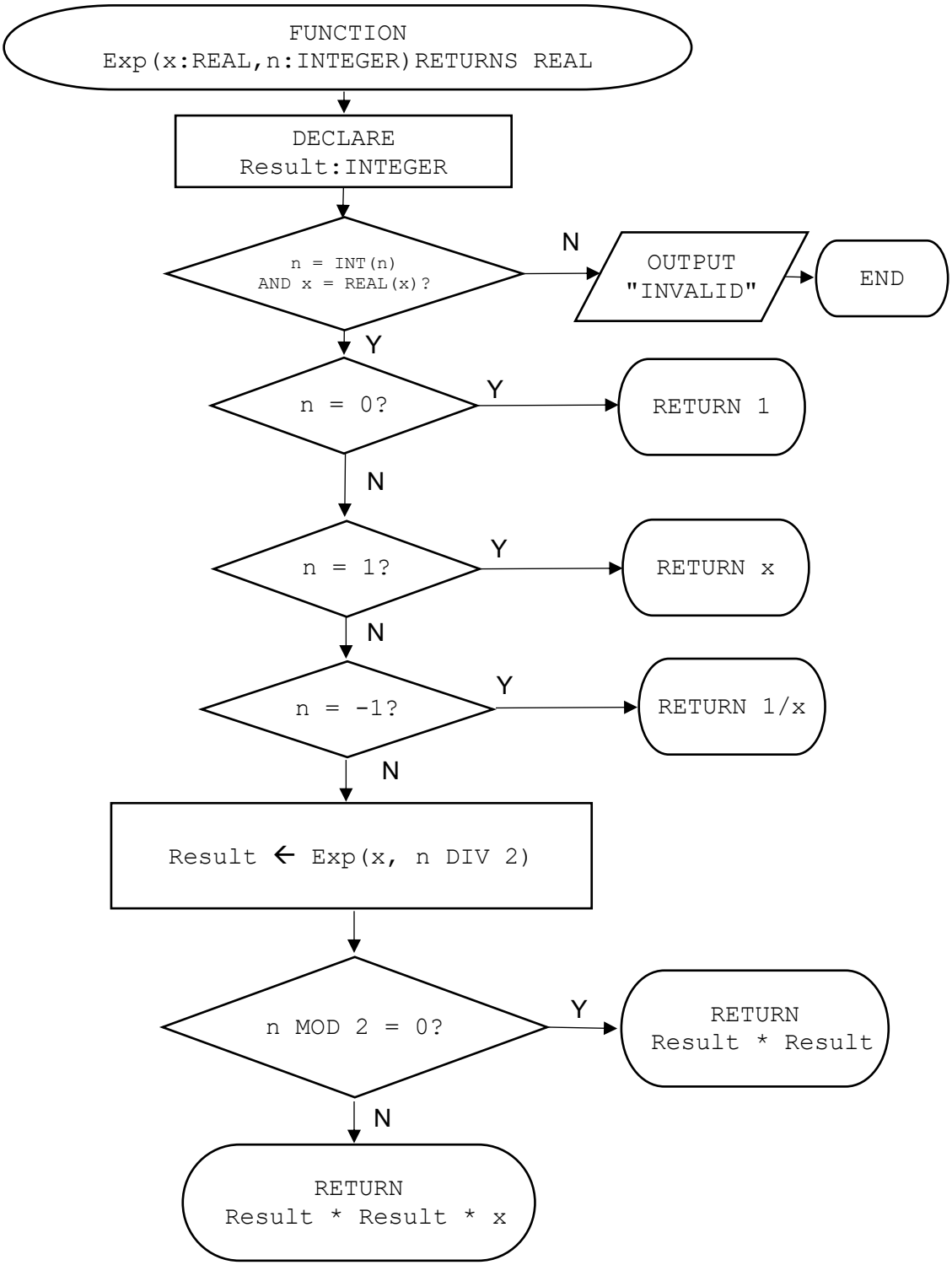
Q1e [2]

Run time complexity O(n!) [1]

The first call generates n recursive calls with n-1 letters, the second recursive call generates n-2 letters, and so on till the base case returns a single letter.

Therefore, total calls is n * (n-1) * (n-2) *..1 = n! [1]

Q2a[5] Solution is in a Flow Chart instead of PseudoCode. Convert to Pseudo code yourself



Base cases for 0, 1 [2],

Base case for -1 [1]

Optimal solution

- Use of n DIV 2 to reduce the number of recursive calls [1]
- Check for odd n or even n to return the correct result [1]

OR

```
def exp(x, n):
    if n == 1:
        return x
    else:
        return x * exp(x, n - 1)
```

For **non-optimal** working solution with O(n)

- Base case [1]
- reduce n by 1 [1]
- recursive call with return result [1]

(b)[2]

Time Complexity is O(log n). or correct notation for non-optimal solution [1]

This is better tha O(n) if we just do n * exp(n-1)

Save recursive calls by calling Exp(n Div 2) recursively [1]

Q3

(a) MYS, AUS, CAN, SGP, JPN. 1 mark for each correct position inserted

Insert AUS: AUS MYS

Insert CAN: AUS CAN MYS

Insert SGP: AUS CAN MYS SGP

Insert JPN: AUS CAN JPN MYS SGP

(b)

A: UpperBound – 1

B: List [Posn + 1]

C: False

D: Temp

Q4

- **[4]**

AI : Annual Income

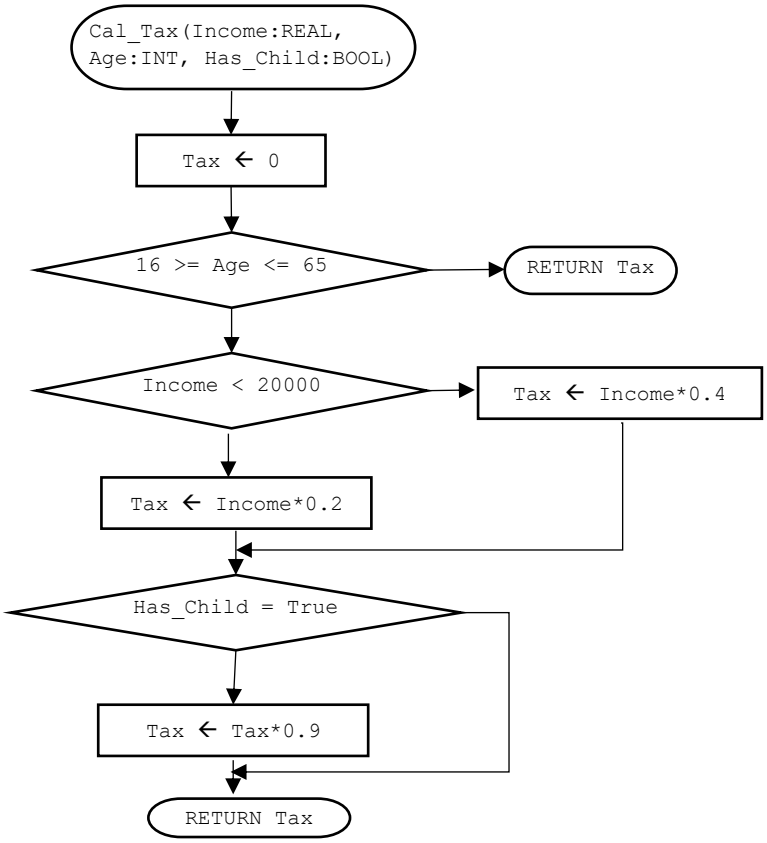
Conditions		**Rules**							
	16<=age<=65	**Y**	**Y**	**Y**	**Y**	**N**	**N**	**N**	**N**
	AI < 20K	**Y**	**Y**	**N**	**N**	**Y**	**Y**	**N**	**N**
	Has Children	**Y**	**N**	**Y**	**N**	**Y**	**N**	**Y**	**N**
Actions	Pay 20%xAI	**x**	**x**						
	Pay 40% x AI			**x**	**x**				
	Reduce Tax by 10% x AI	**x**		**x**					
	No need to pay					**x**	**x**	**x**	**x**

(b)[1]

Conditions		**Rules**							
	16<=age<=65	**Y**	**Y**	**Y**	**Y**	**N**			
	AI < 20K	**Y**	**Y**	**N**	**N**	**-**			
	Has Children	**Y**	**N**	**Y**	**N**	**-**			
Actions	Pay 20%xAI	**x**	**x**						
	Pay 40% x AI			**x**	**x**				
	Reduce Tax by 10% x AI	**x**		**x**					
	No need to pay					**x**			

(c)

Flow Chart



Q5(a)

- **Data is stored in a node which has a pointer that points to another node**
- **The first node in the Linked List is maintain using a pointer**

(b)

- **Dynamic storage vs Fix sized**
- **Direct access using index vs Traversing the List**
- **When the amount of nodes cannot be determined at run time.**

(c),(d)

- **Start, End pointer attributes [2]**
- **InsertInOrder, O(n)**
- **InsertBack,O(1) [1]**
- **InsertFront, O(1)**
- **RemoveFront,O(1) [1]**
- **RemoveBack(1)**

(e)

- **Loop while len(P) and len(Q) are both not 0 [1]**
- **Compare first Node of P and Q**
- **Insert smaller Node from P,Q to a New Linked List, Remove Node from P/Q**
- **End Loop**
- **If len(P) = 0 Then insert rest of nodes in Q Else if len(Q) = 0 insert rest of nodes in P to New Linked list**

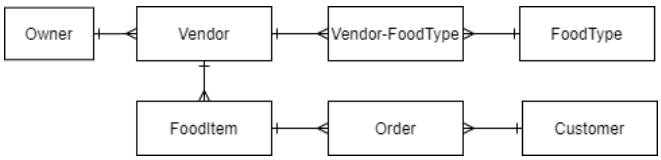
(f)

- **UML [1]**
- **Enqueue, Dequeue [2]**
- **Need InsertBack and RemoveFront [1]**

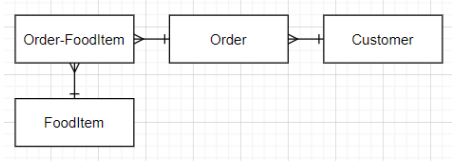
Q6

(a)

[6]



Owner(Optional),Vendor, Customer [1]
FoodItem, 1-M with Vendor [1]
Vendor-FoodType [1] , M-1 relationships with Vendor, FoodType[1]
Order[1], M-1 Relationships with FoodItem, Customer [1]
Alternative: Order-Fooditem



(b) **Customer** (Name : TEXT, Address:TEXT, Contact:TEXT) ID is accepted
[6]

Owner(Name:TEXT,Email:TEXT,Contact:TEXT) *[Accept this as attributes in Vendor table]*

Vendor (StoreName: TEXT, StoreAddress:TEXT, RegNo: TEXT, OwnerContact*:TEXT)

FoodItem(VendorRegNo*: TEXT, ItemNo: INTEGER, ItemName: TEXT, Description: TEXT, Price: REAL)

FoodType(Id: INTEGER, Type: TEXT)

Vendor-FoodType (RegNo*: TEXT, Id*: INTEGER)

Order (Contact*: TEXT, VendorRegNo*: TEXT, ItemNo*: INTEGER, Timestamp: DATETIME, Quantity: INTEGER)

*The order must capture vendor + food item and timestamp
1m- each, all attributes in web form must be captured, pri/foreign keys are correct. Accept Owner fields as attributes of Vendor

(c) User Experience –
[3]

- determines whether a user is able to achieve his objective or needs when using the web app.
- determines whether a user will revisit the web app
  Any one **[1]**
- UI/UX are **interrelated**, a **good UI will contribute to a good UX**, BUT a **good UI does not mean it will have a good UX**
- **OR** The intended user experience will determine the UI used. [**2]**

(d) All data in the database design are captured **[2]** -1m for 1 missing data
[4] Demonstrates any 2 of the following: **[2]**
Visual Hierarachy, Affordance, Consistentncy, Responsive