2021 SH2 Prelim Sample Solutions

Q1

(a) [2] Any 2 of

- **Sorted vs Unsorted**
- **Time complexity**
- **Search space**

(b) [2]

- **Linear search, O(n)**
- **Binary Search, sort O(NlogN) + O(logN)**
- **Linear Search**

(c) [4]

```
FUNCTION LinearSearch(A: Array[1:N] OF INTEGER, k:INTEGER) RETURNS
INTEGER
 val ← MAXINT // Largest Integer Value ,1m
 pos ← -1 //1m
 FOR i = 1 TO N:
  IF A[i] > k and A[i] < val: //2m
            val ← L[i]
            pos ← i
 RETURN pos
ENFUNCTION
```

(d) [2]

Recursive function has very high overheads in memory and computing resources.

Every recursive function pushes state data into the call stack memory, if the depth of the binary serach exceeds the call stack memory it will result in a run time stack overflow error.

Recursive function will need more processing time to pass data to and from the call frame.

Iterative solution is more appropiate as there are no function calls and the all data state changes within a local function scope.

(e) [4] find index of value k

NOTE THAT Any solution involving truncating of Array will result in incorrect index returned

```
FUNCTION Helper(A: ARRAY[1:N] OF INTEGER, k: INTEGER, lb: INTEGER,
ub:INTEGER)
 IF lb > ub THEN
  RETURN -1 //1m
 ENDIF
 mid ← (lb + ub) DIV 2
 IF A[mid] = k THEN
  RETURN mid //1m
 ENDIF
 IF k < A[mid] THEN //1m
  RETURN Helper(A, k, lb, mid-1)
 ELSE:
```

```
  RETURN Helper(A, k, mid+1, ub)
 ENDIF
ENDFUNCTION

FUNCTION BinSearch_1(A: ARRAY[1:N] OF INTEGER, k: INTEGER )
RETURNS INTEGER
 RETURN Helper(A, k, 1, N) //1m
ENDFUNCTION
```

(f) [4] largest value smaller than k

- **Init lb,ub**
- **Terminating condition in loop, make sure terminating condition exists**
- **Update lb, ub**
- **Return based on one of lb,ub, temp variable**

```
//using a variable ret
FUNCTION BinSearch_2(A: ARRAY[1:N] OF INTEGER, k: INTEGER ) RETURNS
INTEGER
 lb ← 1
 ub ← N
 ret ← -INF //smallest numerical value
 WHILE lb <= ub //1m
  mid ← (lb+ub) DIV 2
  IF A[mid] < k THEN //1m
      IF A[mid] > ret THEN
       ret = A[mid]
      ENDIF
   lb ← mid + 1
  ELSE
   ub ← mid – 1 ## final ret will be
  ENDIF

  IF ret > -INF THEN
   RETURN ret
  ELSE
   RETURN -1
  ENDIF
ENDFUNCTION

OR
//
FUNCTION BinSearch_2(A: ARRAY[1:N] OF INTEGER, k: INTEGER ) RETURNS
INTEGER
 lb ← 1
 ub ← N
 WHILE lb <= ub //1m
  mid ← (lb+ub) DIV 2
  IF k <= A[mid] //1m , the result will be ub eventually since the
search space now contains all values smaller than k
   ub ← mid - 1
  ELSE
   lb ← mid + 1
  ENDIF

  IF ub >= 1 AND ub <= N THEN //1m
   RETURN A[ub]
  ELSE
   RETURN -1 //1m
  ENDIF
 ENDWHILE
ENDFUNCTION
```

(g) [4],1 valid, 1 invalid, any 2 boundary

Category	Test Case	Expected Result
Valid	BinSearch_2([2,4,6,8], 5)	4
Boundary	BinSearch_2([2,4,6,8], 1)	-1
Boundary	BinSearch_2([2,4,6,8], 2)	-1
Boundary	BinSearch_2([2,4,6,8], 10)	8
Boundary	BinSearch_2([2,4,6,8], 8)	6
Invalid	BinSearch_2([2,4,6,8], "a")	Runtime error

(h) [8]

- **Recursive search leftmost index, rightmost index**
- **Use a variable index to keep the leftmost and rightmost index found**

```
FUNCTION Helper(A:ARRAY[1:N] OF INTEGER, k: INTEGER, lb: INTEGER,
ub: INTEGER, flag: STRING, index: INTEGER) RETURNS INTEGER

 IF lb > ub THEN //1m base case
  RETURN index
 ENDIF

 mid ← (lb+ub)DIV 2 //1m recursive step

 IF k = A[mid] THEN
  index ← mid //1m for key found

   IF flag = "L":
    RETURN Helper(A, k, lb, mid-1, flag, index) // 1m search left
sublist
   ELSE: //1m search right sublist
    RETURN Helper(A, k, mid+1, ub, flag, index)
   ENDIF

 ELSE IF k < array[mid] THEN //1m
  RETURN Helper(A, k, lb, mid-1, flag, index)
 ELSE: //1m
  RETURN Helper(A, k, mid+1, ub, flag, index)
 ENDIF

END FUNCTION

FUNCTION BinSearch_3(A:ARRAY[1:N] OF INTEGER, k: INTEGER) RETURNS
ARRAY[1:2] OF INTEGER
//bootstrap 1m

 DECLARE result: ARRAY[1:2] OF INTEGER
 result[1] ← Helper(A, k, 1, N, "L", -1)
 result[2] ← Helper(A, k, 1, N, "R", -1)
 RETURN result

ENFUNCTION
```

OR using "global variables"

```
FUNCTION BinSearch_3 (A:ARRAY[1:N] OF INTEGER, k: INTEGER) RETURNS
ARRAY[1:2] OF INTEGER

    FUNCTION Helper(lb: INTEGER, ub: INTEGER) RETURNS INTEGER
        IF lb > ub THEN
            RETURN
        ENDIF
        mid ←(lb+ub)DIV 2
        IF k = A[mid] THEN
           IF mid < low THEN
```

```
                low ← mid
            ENDIF
            IF mid > high THEN
                high ← mid
            ENDIF
            Helper(lb, mid-1) //search left sublist
            Helper(mid+1,ub) //search right sublist
        ELSE IF k < A[mid] THEN
            Helper(lb,mid-1)
        ELSE:
            Helper(mid+1, ub)
        ENDIF

    low = N + 1
    high = 0
    lb, ub = 1, N
    Helper(lb, ub)
IF low = N + 1 THEN
 low ←  -1
ENDIF
    DECLARE result: ARRAY[1:2] OF INTEGER
    result[1] ← low
result[2] ← high

    RETURN result
```

Q2

(a) [2]

Source	Directly connected neighbours
Node(1)	[Node(2), Node(3)]
Node(2)	[Node(4)]
Node(3)	[Node(2)]
Node(4)	[Node(3)]

(b) [3]

- **Append to List**
- **Iterate over List, getHead, getNext**
- **Existence in List**
- **Size of List**

List
- **Hidden (N.A)**
+ **constructor()** **+ getHead(): Object // returns the first node in the List ,None if empty List** **+ size(): INTEGER // size of List** **+ append (node: Object) : None // add node to the end of the List [1m]** + remove(): None // remove the fist node in List **+ inList(n: Object) : Boolean // returns True if object n is in the List [1m]** **+ getNext(n: Object): Object // returns the next node from the List after n , returns None if n is the last node in the List [1m]**

(c)[3]
Dictionary
Any 1,1m

- **The set of nodes in the graph do not not have duplicates, therefore we can use the IDs of the nodes as keys in the Dictionary and a List of Node objects as its value.**
- **Provides O(1) access time to the List, using a integer key**
- **Provides O(1) storing of the List using a integer key**
- **Can store any objects with a key**

Dictionary
- **Hidden (N.A)**
+ get(key: INTEGER) : List //1m returns a List **+ set(key: INTEGER, data: List) : None** **+ getKeys(): ARRAY [1:N] of INTEGERS // 1m array of the keys in Dictionary**

(d)[6]

Structued English Solution

- **A List,** `visited` **to keep the nodes found, initially empty**
- **Call recursive function to lookup** `source` **and return visited**
    - **Base case , if** `source` **is in** `visited` **then return visited**
    - **Recursive step:**
        - **Add** `source` **to visited**
        - **Lookup** `source` **from graph for a List of neighbour nodes, Call recusive function for each neighbour node as source**

Pseudocode Solution

```
FUNCTION Helper(graph: OBJECT, source: INTEGER, visited: List)
RETURNS List
 DECLARE neighbours: List

 IF visited.inList(source) THEN // 1m Base case
  RETURN visited
 ENDIF

 visited.append(source)  //1m
 neighbours ← graph.get(source) //1m
 node ← neighbours.getHead()

 WHILE node <> None DO //1m iterate over List, implicit base case
for empty neighbours list
  visited ← Helper(graph, node.getID(), visited) //1m
  node ← neighbours.getNext(node)
 ENDWHILE

 RETURN visited
ENDFUNCTION

FUNCTION FindPath(graph: OBJECT, source: INTEGER) RETURNS List
 DECLARE visited: List
 visited ← List()
 Helper(graph, source, visited) //1m, need a List to store the nodes
found
 RETURN visited
ENFUNCTION
```

(e) [3]

- **Iterate over the set of nodes in graph**
- **Check if nodes visited is same size as the nodes in the graph**

```
nodes = graph.getKeys() // 1m number of nodes in graph
FOR i = 1 to LEN(nodes) //LEN returns size of array
 paths = FindPath(graph, nodes[i] ) //1m
 if paths.size() = LEN(nodes) THEN //1m
  OUTPUT( "Node", nodes[i] )
```

```
ENDFOR
```

OR

- **Get a set of nodes in graph, s**
- **For each node in graph, get the List of visted nodes using FindPath and compare with s, if same set then that node can reach all nodes**

```
nodes = graph.getKeys() // 1m number of nodes in graph
s = set(nodes) // s is a set of nodes
FOR i = 1 to LEN(nodes) //LEN returns size of array
 paths = FindPath(graph, nodes[i] ) //1m
 if s = set(paths) THEN //1m
  OUTPUT( "Node", nodes[i] )
ENDFOR
```

Q3

(a)[2]

Hash Table: Store and Find data items in O(1)/O(n) time.

Hash(input data value) → address

address	**data value**

Any 2

- Given an input data , a hash function should uniformly distribute the input data across the range of addresses.
- The hash function must use all the input data to generate the address, ie if data = d_1 d_2 d_3 d_4. d_1 d_2 d_3 d_4 must all be used as input by the hash function to generate the address.
- The address is fully determined by the data being hashed or given two identical input data the hash function must generate the same address.
- Produce minimal collision (2 inputs generating the same address)

Additional Notes: **Dictionary (Map)**

Storing (key: immutable object, value) in a Hash Table : Find a value, given the key in O(1)/O(n)

Hash(key) → address

address	**(key,value)**

(b)[2]

When the hash function generates a address in the hash table that is already occupied.

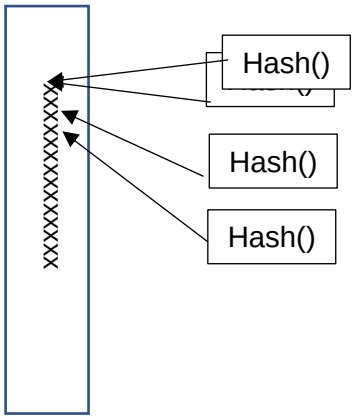
In separate chaining:

```
IF the address maps to a memory location that is not occupied THEN
//1m
 Create a empty linked list in the memory location
 Append the data value to the linked list
ELSE //1m
 Append the data to the end of the linked list
ENDIF
```

(c)[1]

- In linear probing, the data values that are map to the same address in the hash table will have the data stored in consecutive locations, input data that maps to these

locations that are already occuied further increase the length of the consecutive locations, resulting in clustering of data in the hash table.



(d)

```
IF table[i] = NONE
   THEN //1m (i)
     table[i] ← name
     RETURN True
   ELSE IF i = key
   THEN //1m (ii)
     RETURN FALSE
   ELSE //1m (iii)
     step ← step + 1
 ENDIF
```

(iv) [2]

Instead of using the next free memory location, this algorithm selects a new location that is twice the interval from the previous location .(1m)

This solves the clustering problem , as the data are more distributed in the hash table.(1m)

(v)[2]

The probing does not search for all memory locations in the table.

Results in a lot of empty slots that are not used.

Q4

(a) [1] A set of rules that determine how the sender and receiver exchange data
(b) [1] Resolve domain names to ip addresses
(c) [2]

- **The logical name space (data) is centralised, there is only 1 global domain name space on the Internet. ie domain names are unique.**
- **The database storing the name space is decentralised via a hierarchical structure. Different partitions of the namespace are maintained by different authoritative DNS servers.**

(d) [2] any 2

- **http: protocol**
- **nationaljc.moe.edu.sg: domain name**
- **8088: port number of listening socket**
- **/20SH07: resource**

(e) [2]

- **Domain name resolved to ip address**
- **Browser connects to the remote server port 8088**
- **Sends a http request for /20SH07**

(f) [2] Any 2

- **Native email client uses the SMTP protocol to communicate with a email server**
- **Web browser client uses the HTTP protocol to send and receive html contents, which is converted from the raw email content by the email web application server.(Gmail)**
- **Offline vs online access**
- **Native device user interface vs web interface**

(g) [3]

- **Create a socket to connect to the web server ip address and port number (8088)**
- **Generate a HTTP GET request message for the hidden page resource (/20SH07)**
- **Receive and decode the return HTML message content**
- **Parse the HTML content to extract the parts that contains data**

(h) [2] NOTE: This question is not related to PDPA, it is a code of conduct of the programmer

- **The contents of the school's web site is protected by copyrights laws. (you aren't copyrighting a "website," but you can copyright the contents of that website)**
- **By extracting data from the web site, you are re-using "original content" without permission and therefore is in violation of copyrights laws.**
- **The privacy of the people whose personal data you extract has been compromised. This goes against this code of conduct: *Computing professionals should only use personal information for legitimate ends and without violating the rights of individuals and groups.***

(i) [2]

- **Consent is not obtained from the staff / students**
- **Purpose for the use of the data is different from the original intent.**
- **Data Protection is not inplace**
- **Removal of contents once its purpose is no longer in need.**

(j) [1]

- **A central authority should kept the information, access to information is granted and control by the central authority. Authentication and access control should be enforced.**

(k) [2]

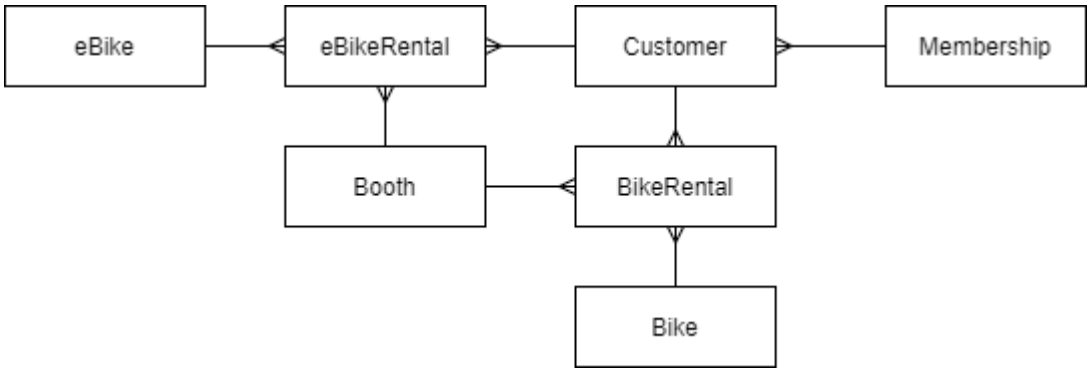
- **Encoding is in utf-16**
- **Web scraping program is using another encoding scheme by default**

- **Terminal console used by the web scraping program does not have the fonts for the character encoding.**

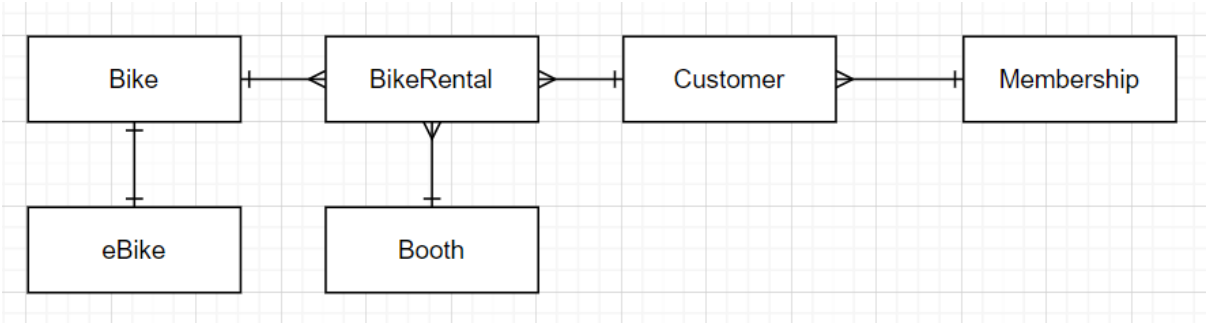
Q5

(a)[6]

- **Customer,Bike/eBike 1m**
- **Booth 1m**
- **Membership 1m**
- **Bike/eBike Rental 1m**
- **Relationship Cardinality 2m**



OR



(b)[4]

- **Customer, Bike/eBike all attributes captured**
- **Bike/eBike Rental all attributes captured**
- **All FK/PK**
- **Booth, Membership**

Customer (CustID, Name,Contact,CreditCardNumber, Tier*)
OR
Customer (Name,Contact,CreditCardNumber, Tier*)

Bike (SerialNumber, ModelNumber, YOM, ServiceDate,Mileage)

eBike (SerialNumber, ModelNumber, YOM, ServiceDate, Mileage, Power, BatteryCapacity,Charge)

OR

Bike (SerialNumber, ModelNumber, YOM, ServiceDate,Mileage, eBike: BOOLEAN)

eBike (SerialNumber*, Power,BatteryCapacity,Charge)

OR

Bike (SerialNumber, ModelNumber, YOM, ServiceDate,Mileage, eBike*: INTEGER)

eBike (SerialNumber, Power,BatteryCapacity,Charge)

eBike/BikeRental (TransactionID, CustID*, SerialNumber*, CheckoutDateTime, CheckoutBooth*, CheckinDateTime, CheckinBooth*, RentalFee)
OR
eBike/BikeRental (CustID*, SerialNumber*, CheckoutDateTime, CheckoutBooth*, CheckinDateTime, CheckinBooth*, RentalFee)

Booth (BoothID, Address)

Membership (Tier, MonthlySubscription, RentalRate)

(c) [2] Any 2

- **Range Eg date and time**
- **Format, Eg Credit Card Number**
- **Presence, Eg already a Customer, BoothID**
- **Length, Eg Serial Number**

(d)

(i)[4]

- **Identify 3 conditions**
- **Identify3 actions**
- **All outcomes presented**
- **All condition/action correct**

		Rules							
		1	2	3	4	5	6	7	8
Conditions	Last Serviced Date > 12 months	Y	Y	Y	Y	N	N	N	N
	Mileage > 1000km	Y	Y	N	N	Y	Y	N	N
	e-bike battery < 50% charge	Y	N	Y	N	Y	N	Y	N
Actions	Cannot checkout bike	✓	✓	✓	✓	✓		✓	
	Can check out with Warning message						✓		
	Can check out bike								✓

(ii)[2]

Conditions	Last Serviced Date > 12 months	Y	-	N	N
	Mileage > 1000km	-	-	Y	N
	e-bike battery < 50% charge	-	Y	N	N
Actions	Cannot checkout bike	✓	✓		
	Can check out bike with Warning message			✓	
	Can check out bike				✓

(iii)[3]

- **Correct condition**
- **Correct action**
- **All symbols/arrows correct**