Task 1.1 Evidence 1 Program Code
Loop to input up to 3 city names and temperatures
Determine absolute difference
Initialize suitable value for greatest difference
Compare subsequent values to determine greatest difference
Display city name and greatest temperature difference
Display #days elapsed since last greatest difference
Update most recent greatest difference and city name
Task 1.2 Evidence 2 Test Data Screenshot
Input showing the maximum 3 cities
Input showing between 1 or 2 cities
Attempt to input an invalid temperature
Daily differences have one greatest value
Daily differences have two or more equal greatest values
Daily greatest is less than stored greatest
Daily greatest is more than stored greatest
Daily greatest is equal to stored greatest
Task 2.1 Evidence 3 Program Code & Efficiency
Meaningful function name
with correct parameters (array, low, high)
Tail end terminating case
Condition for not found (low > high)
Output message for not found
Exit action for not found
Condition for recursive case
Recursive case for left subarray
Recursive case for right subarray
Task 2.2 Evidence 4 Quick sort
Function called from main program
Parameter list - array, left, right
Correct terminating case
Correct recursive case
Task 2.2 Evidence 5 Screenshot
All values shown
In order
Task 3.1 Evidence 6 Update Program Code
Meaningful function name
Files opened in correct modes, close files
File exception handling with error messsage
UPDATED.txt records sorted before processing [2]
Loop for master and transaction files

Comparison of country name
If master country > transaction country, write master
If master country = transaction country, write transaction
If end of master file, write remaining master
Task 3.2 Evidence 7 HashKey Program Code
HashKey function with input parameter string and output parameter integer
Address is hashed (call HashKey)
ASCII code calculated for each country characters
Total of all ASCII values calculated
Remainder calculated with modulo arithmetic
Address determined and returned
Task 3.3 Evidence 8 CreateCurrency Program Code
Comments for collision resolution strategy [2]
Files opened in correct modes, close files
File exception handling with error messsage
Loop for all countries
Calculate address using HashKey
Check if collision
If no write directly
If yes write to appropriate address [2]
Allow for wrap around
Task 3.4 Evidence 9 LookUpCurrency Program Code
Prompt + input country name
Call HashKey
File handling to get record
Formatted address, country name and exchange rate
Task 3.4 Evidence 10 Screenshot
Correct index
Singapore exchange rate data displayed
Task 3.5 Evidence 11 FindCollisions Program Code
Loop for all countries
Check if hashed address = current address
Add collided records to appropriate data structure
Display collided records
Task 3.5 Evidence 12 Two screenshots
Collided records screenshot 1
Collided records screenshot 2
Task 4.1 Evidence 13 Linked List Program Code
Open file in correct mode, close file
Initialize linked list data structure
Loop through all game records

Insert game record to linked list
Input and validate score
Traverse to player node and get old rank
Update player score
Update node position in linked list [2]
Get player node and new rank
Output player old and new ranks
Task 4.2 Evidence 14 Testing + Screenshots
Validation of erroneous score (data type, range)
Validation of boundary score (0, 20)
Validation of normal score (1-19)
Change in old and new ranks
No change in old and new ranks
Task 4.3 Evidence 15 Rank Range Program Code
Validation of two ranks (low <= high)
Validation of individual rank between 1 and #players
Validation of data type
Traverse linked list to get rank range
Correct terminating condition
Correct determination of rank
For each rank, loop to get player ids
Appropriate data structure to store results
Sum to get #players for each rank
Output rank, player ids and #players
Task 4.4 Evidence 16 Annotated Screenshots
Validation of erroneous range (low > high, negative, >#players)
Low rank = high rank (i.e. 1 rank) (at least 2 players)
Validation of normal range (low < high) (at least 2 players for 1 or more of the ranks)