Name:		Index Number:		Class:	

DUNMAN HIGH SCHOOL
Preliminary Examination
Year 6

COMPUTING 9597

(Higher 2)

Paper 1

29 August 2016
3 hours 15 minutes

Additional Materials: Data files and `EVIDENCE.docx`

READ THESE INSTRUCTIONS FIRST

Type in the `EVIDENCE.docx` document the following:

- Candidate details
- Programming language used

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

All tasks and required evidence are numbered. The marks is given in brackets [ ] at the end of each task.

Note: You may not use the built-in `sort()`, `min()` and `max()` functions.

Copy and paste required evidence of program code and screen shots into `EVIDENCE.docx`.

At the end of the examination, print to pdf and submit your `EVIDENCE.pdf`.

Data files

Q1 – `DIVE1.TXT, DIVE2.TXT`
Q2 – `DATA.TXT`
Q3 – `POKEMONS.TXT, CANDIES.TXT`
Q4 – `Nil`

1. In Olympic diving, scoring is done by a panel comprising a minimum of 3 judges. The highest and lowest scores are dropped to eliminate the extremes. The raw score is computed by summing the middle scores. The raw score is then multiplied by the level of difficulty to give the total score.

Sample scoring for a 5-judges panel:

Scores: 6.5, 6, 6.5, 6, 5.5
Lowest (5.5) and highest (6.5) scores dropped
Raw Score = 18.5 (6.5 + 6 + 6)
Total Score = Score (18.5) x Difficulty Level (1.5) = 27.75

`DIVE1.TXT` contains the countries, difficulty levels and 5-judges scores for a diving competition.

Task 1.1
Write program code to determine the podium winners of the competition. Your program should output the medal (Gold, Silver, Bronze), country name and the total score (to 2 decimal places).

Sample output:

```
Gold:   China          45.90
Silver: Malaysia       39.20
Bronze: United States  36.00
```

Evidence 1:
Program code. [5]

Evidence 2:
Screenshot of output. [1]

In most international competitions with more than five judges, the 3/5 method is used. The middle 5 numbers are added and then multiplied by the difficulty of the dive and then multiplied again by 0.6.

`DIVE2.TXT` contains the countries, difficulty levels and 9-judges scores for a diving competition.

Task 1.2
Write program code to determine the podium winners of the competition. Your program should output the medal (Gold, Silver, Bronze), country name as well as the total score (to 2 decimal places).

Evidence 3:
Program code. [8]

Evidence 4:
Screenshot of output. [1]

2. Quicksort is an efficient sorting algorithm to order values in an array.

Task 2.1
Write a recursive quicksort method to sort the values in the text file `DATA.TXT` in descending order.

Evidence 5:
Program code for Task 2.1. [4]

Evidence 6:
Screenshot of output. [2]

Task 2.2
Using a suitable data structure, convert the recursive quicksort in Task 2.1 to one using iteration.

Confirm the correctness of your iterative quicksort on the data set used in Task 2.1

Evidence 7:
Program code for Task 2.2. [8]

Evidence 8:
Screenshot of output. [1]

3. A Pokemon Go fan who nicknames himself Pokeboy wishes to manage his Pokemon collection using a binary search tree. Each binary search tree node stores the name together with the numbers of that particular Pokemon and candies collected, as well as a reference to a linked list. Each linked list node stores the combat power (CP) of a Pokemon. The binary search tree is organised in ascending Pokemon name order. Each linked list is organised in descending order of CP. For ease of reference, we shall name this composite data structure Poketree.

Task 3.1
Using object-oriented programming, construct appropriate classes to initialise, insert and display the contents of the Poketree.

In your main driver program, write code to initialise a Poketree with the first Pokemon collected which is randomly generated from the contents in the file `POKEMONS.TXT` as well as a randomly generated CP in the range 10 to 200 inclusive. When a Pokemon is caught, 3 candies of that kind are also added to the candies count of the Poketree binary search tree node.

Evidence 9:
Program code for class definition, initialisation and display methods. [7]

Evidence 10:
Screenshot output to show contents of Poketree with first Pokemon inserted. [1]

Task 3.2
Write a class method insert() and the necessary main driver program code to insert another 23 randomly generated Pokemons and their corresponding CPs into the Poketree.

Evidence 11:
Program code for Task 3.2. [4]

Evidence 12:
Screenshot to show updated contents of Poketree. [1]

Apart from collecting Pokemons, one can also evolve Pokemons. When a new Pokemon is evolved, its previous incarnation is deleted from the Poketree and its new incarnation either inserted (if its kind is not previously collected) or updated (if its kind is previously collected) to the Poketree. Evolving also requires a set amount and type of Pokemon candy. If there is insufficient Pokemon candies, Pokeboy's preferred action is to exchange (i.e. delete) existing Pokemons of the same species for candies, starting from the Pokemon with the lowest CP. Each Pokemon can be exchanged for one candy. Pokeboy also wishes to keep at least 2 Pokemons of the same species in his collection (well he is a Pokemon fan).

The text file `CANDIES.TXT` contains the candies requirement for evolvement.

Task 3.3
Write a Boolean class method `can_evolve(pokemon)` to determine if a particular Pokemon can evolve by Pokeboy's preference. A Pokemon can evolve if there are sufficient candies or it is possible to exchange candies, leaving a minimum of 2 Pokemons of that species.

Write a class method `evolve(pokemon)` which will evolve a Pokemon using

`can_evolve(pokemon).`

Add necessary class method(s) and main driver program code to evolve all evolvable Pokemons.

Evidence 13:
Program code for Task 3.3. [7]

Evidence 14:
Screenshot(s) to show evolution. [1]

Task 3.4
Write a class method `most_poke()` to output the frequency and most collected Pokemon(s) collected by Pokeboy.

Add necessary program code to exercise `most_poke()`.

Evidence 15:
Program code for Task 3.4. [4]

Evidence 16:
Screenshot for Task 3.4. [1]

Task 3.5
Pokeboy aspires to join the Catch Them All club. The Catch Them All club is a group of elites who have managed to catch every species of Pokemon at least once. Write a class method `catch_them_all()` to either output the message "Welcome to the club! You have caught them all!" or output the number and remaining Pokemon names yet to be caught.

Add necessary program code to exercise `catch_them_all()`.

Evidence 17:
Program code for Task 3.5. [6]

Evidence 18:
Screenshot for annotated test cases. [2]

Task 3.6
After some time, it becomes necessary to reorganise the binary search tree to ensure optimal search performance. Write program code to rebalance the binary search tree and include as comments your strategy to do this. You should output the roots and heights of the old and new binary search trees.

Evidence 19:
Program code for Task 3.6. [5]

Evidence 20:
Screenshot. [1]

4. A magic index in an array `A` is defined to be an index such that `A[i] = i`.

Task 4.1
Given a sorted array of distinct integers, write a brute force iterative method `magic_index(A)` to find a magic index if one exists. If a magic index does not exist, return -1.

Evidence 21:
Function code for `magic_index(A)` and relevant driver code. [5]

Evidence 22:
Screenshot of output. [2]

Task 4.2
Write an efficient recursive method `magic_index_duplicates(A)` to find a magic index for an array containing non-distinct values. If a magic index does not exist, return -1.

Evidence 23:
Function code for `magic_index_duplicates(A)` and relevant driver code. [6]

Evidence 24:
Screenshot of output. [2]

A child is running up a staircase with n steps and can hop either 1, 2 or 3 steps at a time.

Task 4.3
Write a brute force recursive method to count how many possible ways the child can run up the stairs.

Evidence 25:
Function code and relevant driver code. [5]

Evidence 26:
Screenshot of output. [2]

Task 4.4
Optimise your brute force recursive solution in Task 4.3 by eliminating unnecessary recomputations.

Evidence 27:
Function code and relevant driver code. [6]

Evidence 28:
Screenshots showing annotated test cases. [2]

END OF PAPER 1