This question paper consists
of 5 printed pages each
of which is identified by the Code COMP271101

School of Computing

January 2018

COMP271101: Algorithms and Data Structures I

Answer all THREE questions

Time allowed: 2 hours

TURN OVER

Question 1

Suppose n points lying on a horizontal axis are given by their coordinates `X[0]`, `X[1]`, ... , `X[n-1]`. The distance between points i and j is `|X[i] - X[j]|`. The task is to find the distance between two nearest points. For example, given 5 points with coordinates 8, 1, 15, 3, 12, the two nearest points have coordinates 1 and 3 and the distance is |1 - 3|=2.
The two algorithms `A1` and `A2` are presented below.

```
ALGORITHM A1(X[0..n-1])
// Input:  array X[0..n-1] of X-coordinates of n points on X-axis
// Output: the distance between two nearest points
minDist ← |X[0] - X[1]|
for i←0 to n-2 do
  for j←i+1 to n-1 do
     Distance ← |X[i] - X[j]|
     if Distance < minDist
         minDist ← Distance
output minDist
```

```
ALGORITHM A2(X[0..n-1])
// Input:  array X[0..n-1] of X-coordinates of n points on X-axis
// Output: the distance between two nearest points
Sort array X using quick-sort
minDist ← |X[0] - X[1]|
for i←1 to n-2 do
  Distance ← |X[i] - X[i+1]|
  if Distance < minDist
     minDist ← Distance
output minDist
```

(a) Provide a brief explanation on how algorithms `A1` and `A2` operate and why they succeed in solving the problem. **[2 marks]**

(b) Trace algorithm `A1` on the list 6, 15, 2, 13, 3, 10. Show how the values of variables `Distance` and `minDist` change for different values i and j. What does the algorithm output? **[2 marks]**

(c) What is the time complexity of algorithm `A1` in terms of big-O? Explain your answer by using the formal definition of big-O. **[3 marks]**

(d) Algorithm A2 starts with quick-sort shown below. Analyse the worst-case time complexity of quick-sort. **[4 marks]**

```
ALGORITHM quickSort(X[0..n-1])
//Input:  array X[0..n-1] of n numbers
//Output: array X sorted in ascending order
if n > 1
   pivot ← X[0]
   copy the elements of X smaller than pivot to S1
   copy the elements of X equal to pivot to S2
   copy the elements of X larger than pivot to S3
   quickSort(S1)
   quickSort(S3)
   copy the elements back into X in order:
   -  first copy the elements of S1,
   -  then those of S2,
   -  and finally those of S3.
```

(e) Based on the result of part **(d)**, analyse the time complexity of Algorithm A2. **[2 marks]**

(f) Actual running times of algorithms in computational experiments may differ from their theoretical estimates derived for worst-case instances. Which algorithm A1 or A2 would be faster in computational experiments on random data? Explain. **[2 marks]**

(g) Describe another approach that outperforms both algorithms A1 and A2 for worst-case instances and uses one of the classical sorting algorithms as a subroutine. Give the name of the appropriate sorting algorithm and make a conclusion about the overall time complexity of the new approach. **[2 marks]**

(h) Suppose the coordinates of the points are stored in a sorted array $X[0..n-1]$). Our task is to identify a sub-array of X representing all points which fall between two given values L and U: $L \le X[i] \le U$. Explain how this can be done in $O(\log_2 n)$ time.

Will your algorithm work if there are several equal elements in the array X (they represent points having the same coordinate)? **[3 marks]**

[20 marks total]

Question 2

(a) What is the definition of **recursive** as used in algorithms?
What are the main two parts of a recursive algorithm?
What are the overheads associated with recursive algorithms? **[2 marks]**

(b) What are the overheads associated with recursive algorithms? **[2 marks]**

(c) Describe the main differences between the iterative and recursive implementations of the binary search algorithm. What is the time complexity of each version? **[3 marks]**

(d) The two recursive algorithms presented below calculate the same expression 2^n for a given non-negative integer n. Which algorithm is faster? Provide a brief explanation. **[2 marks]**

```
ALGORITHM power1(n)
if n == 0 return 1
else return power1(n-1)+ power1(n-1)
```

```
ALGORITHM power2(n)
if n == 0 return 1
else return 2×power2(n-1)
```

(e) A recursive algorithm $h(n, k)$ is given below.

```
ALGORITHM h(n, k)
//Input:  two positive integers n and k.
if k == 1 return n
else
     if n == k return 1
     else
          return h(n-1,k-1) + h(n-1,k)
```

(i) Perform box-tracing of the algorithm and demonstrate how it computes $h(5,3)$. Specify the value found. **[6 marks]**

(ii) Explain whether the base case is appropriate or not. If you think that the algorithm will not terminate on some input values, give an example. Observe that only positive integers n and k can be considered as input parameters. **[2 marks]**

(f) Suppose the number of elementary operations performed by an algorithm is $n^5 + 1000n^2 \log_2 n - 5n + 5$. Indicate the complexity class $O(?)$. Use the formal definition of Big-O to prove your statement. **[3 marks]**

Question 3

(a) Describe the main operations of the following data structures:
(i) stack, **[2 marks]**
(ii) queue, **[2 marks]**
(iii) priority queue **[2 marks]**

(b) There are two common implementations of a stack. Draw the schemes that illustrate these implementations for a stack consisting of elements *A*, *B*, *C*, *D*, *E*, *F*, *G* under an assumption that the elements have been inserted in the stack in this order. **[6 marks]**

(c) What are the drawbacks of each implementation of a stack stated in (b)? **[2 marks]**

(d) For each implementation, specify the time complexity required for the main stack operations. Explain. **[2 marks]**

(e) The algorithm presented below uses a stack to convert a decimal integer into a binary equivalent. Consider an input number 20 and show the content of the stack after the first loop is completed. Specify the number printed after the algorithm terminates. **[4 marks]**

```
ALGORITHM decimalToBinary(Number)
//    Input:  a decimal integer Number
//    Output: a binary equivalent is printed
create an empty stack
while Number > 0
  digit ← Number % 2          (Number % 2 is the remainder of dividing Number by 2)
  push digit to stack
  Number ← ⌊Number/2⌋        (⌊Number/2⌋ is the integer part of Number/2 rounded down)

// Binary number created in stack
while stack is not empty
  pop digit from stack
  print digit
```

[20 marks total]

END