NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

FINAL ASSESSMENT TEST FOR
Semester 2, AY2018/2019

CS3230 – DESIGN AND ANALYSIS OF ALGORITHMS

6 May 2019 Time Allowed: 2 hours

Instructions to Candidates:

1. This paper consists of **FOUR** questions and comprises **TEN (10)** printed pages, including this page.
2. Answer **ALL** questions.
3. Use the OCR form for multiple choice questions. For all other questions, write your answers in this examination book.
4. This is an **OPEN BOOK** examination.
5. Please write your Student Number only. Do not write your name.

Student Number: ____________________

QUESTION	POSSIBLE	SCORE
Q1	20	
Q2	20	
Q3	20	
Q4	20	
TOTAL	**80**	

IMPORTANT NOTE:

- *You can* ***freely quote*** *standard algorithms and data structures covered in the lectures and homeworks. Explain* ***any modifications*** *you make to them.*
- *Unless otherwise specified, you are expected to* ***prove (justify)*** *your results.*

Q1. Multiple Choice Questions

Use 2B pencil for this section. Shade and write your student number completely and correctly (please check!) on the OCR form. You do not need to fill in any other particulars on the OCR form.

For each multiple choice question, choose the best answer and shade the corresponding choice on the OCR form. Each multiple choice question is worth 2.5 marks. No mark is deducted for wrong answers.

1. Consider the following algorithm.

```
Algorithm Prime(n)
1. if (n == 1) and (n == 0), then return FALSE
2. if (n == 2) or (n == 3), then return TRUE
3. for i = 2 to $\lceil\sqrt{n}\rceil$,
4.     if n is divisible by i, return FALSE
5. Return TRUE
```

What is the invariant of the for loop in step 3?

(1) n is divisible by 2 to i
(2) n is divisible by 2 to $(i - 1)$
(3) n is not divisible by 2 to i
(4) n is not divisible by 2 to $(i - 1)$

ANSWER: (4)

2. What is the solution to the following recurrence?
   $T(n) = 2\,T\left(\frac{n}{3}\right) + n^{\log_3 2} \cdot \log\log n$

   (1) $\Theta\left(n^{\log_3 2} \log\log n\right)$
   (2) $\Theta\left(n^{\log_3 2} \log n\ \log\log n\right)$
   (3) $\Theta\left(n^{\log_3 2} \log n\right)$
   (4) $\Theta\left(n^{\log_3 2}\ (\log\log n)^2\right)$

ANSWER: (2)

3. Given an array of distinct numbers $A[1 \ldots n]$. Our problem is to extract all numbers of odd rank and sort them. For example, if $A[1 \ldots 6] = 4, 3, 8, 2, 5, 7$, we want to output $2 < 4 < 7$.

   Assuming the comparison model, what is a tight lower bound for this computational problem?

   (1) $\Omega(n \log n)$
   (2) $\Omega(n)$
   (3) $\Omega\left(n \sqrt{\log n}\right)$
   (4) $\Omega(n\ (\log n)^2)$

ANSWER: (1)

4. Consider the following recurrence.

$$V(a, a) = 1 \text{ if } 1 \le a \le n$$
$$V(a, b) = V\left(a, \frac{a+b}{2}\right) + V\left(\frac{a+b}{2} + 1, b\right) + (b-a)^2 \text{ if } 1 \le a < b \le n$$

   What is the running time to compute $V(1, n)$ if we use recursion or bottom-up dynamic programming?

   (1) Recursion: $\Theta(n^2)$, DP: $\Theta(n)$
   (2) Recursion: $\Theta(n)$, DP: $\Theta(n^2)$
   (3) Recursion: $\Theta(n)$, DP: $\Theta(n)$
   (4) Recursion: $\Theta(n^2)$, DP: $\Theta(n^2)$

ANSWER: (2)

5. You want to go to the hotel at the top of a mountain. Along the way, there are n hotels where the ith hotel is at height $H[i]$. Suppose $H[1] < H[2] < \cdots < H[n]$ and $H[n]$ is the hotel at the top of the mountain.

   Assume you are at height 0 initially and you can move up at most L meters every day. Also, assume $H[j+1] - H[j] \leq L$ for $1 \leq j < n$ and you need to stay in a hotel every night.

   Denote $D(i)$ be the minimum number of days to reach the ith hotel at height $H[i]$. Let $H[0] = 0$. Which of the following statement is correct?

   (1) $D(0) = 0$ and $D(i) = \min_{0 \leq j < i, H[i]-H[j] \leq L} D(j) + 1$
   (2) $D(1) = 1$ and $D(i) = \min_{1 \leq j < i, H[i]-H[j] \leq L} D(j) + 1$
   (3) $D(i) = 1$ if $H[i] \leq L$ and $D(i) = \min_{0 \leq j < i} D(j) + \delta(i,j)$, where $\delta(i,j) = 1$ if $H[j] - H[i] \leq L$; and $\delta(i,j) = 0$ otherwise.
   (4) $D(0) = 0$ and $D(i) = \min_{0 \leq j < i} D(j) + \delta(i,j)$, where $\delta(i,j) = 1$ if $H[j] - H[i] \leq L$; and $\delta(i,j) = 0$ otherwise.

ANSWER: (1)

6. Refer to Q5, please find an efficient algorithm that finds the minimum number of days required for you to go to the top of the mountain. What is its running time?

   (1) $\Theta(n^2)$
   (2) $\Theta(n \log n)$
   (3) $\Theta(n)$
   (4) $\Theta(\log n)$

ANSWER: (3)

7. There is a database $D[1 \ldots n]$ where each entry $D[i]$ is the information of a person. To access each entry in the database, it costs \$1.

   There are $\log n$ males in the database. Your aim to identify all these $\log n$ males. A randomized algorithm is proposed as follows.

   ```
   1. Initialize $S = \emptyset$.
   2. For $j = 1$ to $\log n$:
      2.1. Repeatedly random access an entry $D[i]$ where $i \notin S$. If
      the entry $i$ is a male, include $i$ in $S$.
   ```

   What is the expected cost of this algorithm?

   (1) $\Theta(n)$
   (2) $\Theta(n \log \log n)$
   (3) $\Theta(n \log n)$
   (4) $\Theta(n^2)$

ANSWER: (2)

8. In the MAX-3SAT problem, given a 3SAT instance, the goal is to find the maximum number of clauses that can be satisfied by an assignment to the variables. Consider the algorithm that simply outputs the number of clauses output by a random assignment. What is the expected approximation ratio of this algorithm?

   (1) 1/2
   (2) 1/8
   (3) 3/4
   (4) 7/8

ANSWER: (4)

Q2. (20 points) Distinct Subsequences

Given an n-length string of English letters, consider the problem of counting the number of distinct subsequences (not necessarily contiguous) appearing in it. For example, the string *aab* contains six distinct subsequences (empty string, *a*, *b*, *aa*, *ab*, *aab*), the string *aba* contains seven (empty string, *a*, *b*, *aa*, *ab*, *ba*, *aba*), while for the string *abc*, all the eight subsequences are distinct.

(a) Let $N[i]$ denote the number of distinct subsequences among the first i characters of a string x. Write a recurrence relation for $N[i]$ in terms of $N[j]$ with $j < i$, and justify.

Hint: Let $prev(i)$ be the largest j in $\{1,2,\dots,i-1\}$ such that $x_j = x_i$, if there exists one. Note that for every subsequence whose last element is $x_{prev(i)}$, there is another identical subsequence that ends at x_i.

ANSWER: Denote the string as $\boldsymbol{x}$. Let $\boldsymbol{prev(i)}$ be the largest $\boldsymbol{j < i}$ such that $\boldsymbol{x_j = x_i}$ and undefined otherwise. Then $\boldsymbol{N[0] = 1}$, and for $\boldsymbol{i > 0}$:

$$\boldsymbol{N[i] = \begin{cases} 2 \cdot N[i-1], & \text{if } prev(i) \text{ is undefined} \\ 2 \cdot N[i-1] - N[prev(i)-1], & \text{otherwise} \end{cases}}$$

Proof: Let $\boldsymbol{S_i}$ be the set of distinct subsequences among the first $\boldsymbol{i}$ characters of the string.

If $\boldsymbol{x_i}$ hasn't occurred before (i.e., $\boldsymbol{prev(i)}$ is undefined), then $\boldsymbol{S_{i-1}}$ and $\boldsymbol{\{s \circ x_i : s \in S_{i-1}\}}$ are disjoint, so $\boldsymbol{N[i] = 2 \cdot N[i-1]}$.

Otherwise, the items in the overlap $\boldsymbol{S_{i-1} \cap \{s \circ x_i : s \in S_{i-1}\}}$ are exactly the strings in $\boldsymbol{S_{prev(i)-1}}$ with $\boldsymbol{x_i}$ appended. So, $\boldsymbol{N[i] = 2 \cdot N[i-1] - N[prev(i)-1]}$.

(b) Using your answer to part (a), design an algorithm that given an n-length sequence of English letters, counts the number of distinct subsequences appearing in it. Your algorithm should run in time $O(n)$.

ANSWER: Below is the pseudocode:

```
CountSub(x):
    $n \leftarrow len(x)$
    $Counts \leftarrow$ empty array of length $n+1$
    $Counts[0] \leftarrow 1$
    $Prev \leftarrow$ empty array of length 26
    for $i \leftarrow 1$ to $n$:
        if $Prev[x[i]]$==null:
            $Counts[i] \leftarrow 2 \cdot Counts[i-1]$
        else:
            $Counts[i] \leftarrow 2 \cdot Counts[i-1] - Counts[Prev[x[i]] - 1]$
        $Prev[x[i]] \leftarrow i$
    return $Counts[n]$
```

Q3. (20 points) Amortized Analysis

Recall that a standard queue maintains a sequence of items subject to the following operations: PUSH(x), PULL() and SIZE(). Each of these operations takes $O(1)$ time in the worst case.

Suppose that we want to support another operation, HALVE(), that removes every alternate element in the queue. We implement it as follows:

```
HALVE()
    $n \leftarrow$ SIZE()
    for $i \leftarrow 0$ to $n-1$
        if $i \bmod 2 == 0$
            PULL()
        else
            PUSH(PULL())
```

For any sequence of PUSH, PULL, and HALVE operations, bound the amortized cost of each operation using the accounting method.

ANSWER:
We can charge each PUSH an amortized cost of 4, and each PULL and HALF an amortized cost of 0.

Each PUSH has an actual cost of 1, so there is a credit of 3 stored with each inserted element.

A PULL can use the stored credit to pay for the actual cost of 1.

For HALVE, in the for loop, when i is even, 1 unit of the stored credit is used to pay for the PULL and the other 2 units are used to pay for the PUSH and PULL for i+1.

Q4. (20 points) Reductions

(a) Consider the following two problems:

- FACTOR: Given a positive integer n, the product of two distinct primes p and q, find p and q.
- EULERTOTIENT: Given a positive integer n, the product of two distinct primes p and q, find $\phi(n) = (p-1)(q-1)$.

Reduce FACTOR to EULERTOTIENT.

ANSWER:
Suppose we have an algorithm to compute $\phi(n) = (p-1)(q-1)$. Then, we can compute $b = n - \phi(n) = p + q - 1$. So, $q = b + 1 - p$. Therefore, if we solve the quadratic equation $p(b + 1 - p) = n$ using the quadratic formula, we can recover p and then q, solving FACTOR.

(b) Consider the following two problems:

- PARTITION: Given a set of n non-negative integers $\{a_1, \dots, a_n\}$, decide if there is a subset $P \subseteq \{1, \dots, n\}$ such that $\sum_{i \in P} a_i = \sum_{i \notin P} a_i$..
- SUBSETSUM: Given a set of n non-negative integers $\{a_1, \dots, a_n\}$ and an integer T, decide if there is a subset $P \subseteq \{1, \dots, n\}$ such that $\sum_{i \in P} a_i = T$.

Reduce SUBSETSUM to PARTITION.

ANSWER: Given an instance of SUBSETSUM $\{a_1, \dots, a_n\}$ and T, let $S = a_1 + a_2 + \cdots + a_n$. Assume S is not 0 or T, and T is not zero, as otherwise the problem is trivial.

Consider the solution to PARTITION on the input $\{a_1, \dots, a_n, S + T, 2S - T\}$. The total sum of all the elements in the input is $4S$.

If the SUBSETSUM instance is a YES-instance, i.e., there is a subset P such that $\sum_{i \in P} a_i = T$, then the PARTITION instance is also a YES-instance because $\sum_{i \in P} a_i + 2S - T = 2S$.

Now, suppose the PARTITION instance is a YES-instance. Note that by assumption, $S + T, 2S - T \neq 2S$ and $(S + T) + (2S - T) > 2S$. So, it must be that $2S - T$ is in one partition and $S + T$ is in the other. Then, the rest of the elements in the partition containing $2S - T$ must sum up to T, forming a solution to the SUBSETSUM instance.

END OF PAPER