This assignment is worth 50 marks representing 8.33% of your total course grade.

Model answers with brief instructions for marking

• 25% of a total mark for each step of solution should be deducted if detailed comments are absent.

1. (2 marks) Work out the time complexity $T(n)$ of the following piece of code in terms of the number of operations:

   ```
   for ( int i = 1; i < n; i *= 3 ) {
     for ( int j = 0; j < n; j += 2 ) {
       for ( int k = 1; k < n; k *= n ) {
         // constant number $C$ of operations
       }
     }
   }
   ```

   (0.25 marks) The inner loop is executed only once (for $k = 1$), so it contributes $C$ operations.
   (0.5 marks) The middle loop is executed $\nu = n/2$ times where $2(\nu - 1) < n \leq 2\nu$.
   (1 mark) The outer loop is executed $m$ times where $3^{m-1} < n \leq 3^m$, or $m = \log_3 n$.
   (0.25 marks) Thus $T(n) = \frac{1}{2}Cn \log_3 n$ operations.

More accurate evaluations such as $\nu = \lfloor n/2 \rfloor$ where $\lfloor z \rfloor$ is the closest integer smaller than or equal to $z$, or $\lfloor \log_3 n \rfloor \leq m < 1 + \lfloor \log_3 n \rfloor$, or $C[n/2]\lfloor \log_3 n \rfloor \leq T(n) < C[n/2] (\lfloor \log_3 \rfloor + 1)$, are not expected and should receive just the same marks.
It is quite legal to solve in terms of Big-Oh provided that all steps of solution are commented so that the processing time was evaluated in terms of the number of operations.

2. (2 marks) Work out the time complexity $T(n)$ of the following piece of code in terms of the number of operations:

   ```
   for ( int i = 0; i < n; i += 3 ) {
     for ( int j = 2; j < n; j = ( j * j ) ) {
       // constant number $C$ of operations
     }
     for ( int k = n; k > 0; k /= 2 ) {
       // constant number $C$ of operations
     }
   }
   ```

   (1 mark) The loop variable $j$ in the first inner loop is changing as $2, 2^2, 2^4, \ldots, 2^{m-1} < n \leq 2^m$ where $m = \log_3 \log_2 n$. Thus, this loop contributes $C \log_2 \log_2 n$ operations.

   (0.75 marks) The second inner loop is executed $\mu$ times where $\frac{\mu}{\log_2 n} > 1 > \frac{\mu}{\log_2 n + 1}$ so that $\log_2 n \leq \mu < \log_2 n + 1$. Thus, this loop contributes about $C \log_2 n$ operations.

   (0.25 marks) The outer loop is executed $n/3$ times, so that the time complexity $T(n) = \frac{1}{3}Cn (\log_2 \log_2 n + \log_2 n)$.

More accurate evaluations are not expected and should receive just the same marks.
It is quite legal to solve in terms of Big-Oh provided that all steps of solution are commented so that the processing time was evaluated in terms of the number of operations.
3. (8 marks) Define formally and prove basic arithmetic relationships for the “Big Omega” notation, namely, scaling, transitivity, rule of sums, and rule of products.

**Hint:** Reformulate Lemmas 1.16–1.19 from the course textbook, p.18.

Solutions need not follow exactly the model answers below: the markers should check logical rather than textual correctness.

**Proof.** By the definition of \( \Omega \), there exist the nonnegative integers \( n_0 \) and \( n_0' \) and positive real constants \( c \) and \( c' \) such that \( h(n) \geq c g(n) \) for all \( n \geq n_0 \) and \( g(n) \geq c' f(n) \) for all \( n \geq n_0' \). Therefore, \( h(n) \geq C f(n) \) for \( n \geq N_0 \) where \( C = c \cdot c' \) and \( N = \max\{n_0, n_0', \} \), what is to be demonstrated.

**Rule of sums.** If \( g_1 \) is \( \Omega(f_1) \) and \( g_2 \) is \( \Omega(f_2) \), then \( g_1 + g_2 \) is \( \Omega(\max\{f_1, f_2\}) \).

**Rule of products.** If \( g_1 \) is \( \Omega(f_1) \) and \( g_2 \) is \( \Omega(f_2) \), then \( g_1 g_2 \) is \( \Omega(f_1 f_2) \).

4. (3 marks) Processing time \( T(n) \) for algorithms below depends on the problem size \( n \). Find the dominant terms having the steepest increase in \( n \) and determine the “Big-Theta” complexity of each algorithm:

(a) \( T(n) = 100n^2 + n^3 + 0.003n^5 \)

(b) \( T(n) = 100n^2 \log_{128} n + n^2 \log_2 n \)

**Hint:** \( \log_a x = \log_n b \cdot \log_b x \) for \( a > 0, b > 0, \) and \( x > 0 \).
5. (5 marks) Prove that $T(n) = 0.1n^2 \log_2 n + 500n$ is $\Omega(n^2)$ and $O(n^{2+\varepsilon})$ where $\varepsilon > 0$ is an arbitrary small positive constant.

(a) (1 mark) $T(n)$ is $\Omega(n^2)$ because $T(n) = 0.1n^2 \log_2 n + 500n \geq 0.1n^2$ for all $n \geq 2$.

(b) (4 marks) $T(n)$ is $O(n^{2+\varepsilon})$ because

- (2 marks out of 4) by the Limit Rule $\lim_{n \to \infty} \frac{T(n)}{n^{2+\varepsilon}} = 0$, i.e.
  
  $$
  \lim_{n \to \infty} \frac{0.1n^2 \log_2 n + 500n}{n^{2+\varepsilon}} = \lim_{n \to \infty} \left( \frac{0.1 \log_2 n}{n^{\varepsilon}} \right) + \lim_{n \to \infty} \left( 500 \frac{1}{n^{1+\varepsilon}} \right) = 0
  $$

- (0.5 marks out of 4) For $n \to \infty$, the last term $\frac{1}{n^{1+\varepsilon}}$ tends to zero.

- (1.5 mark) By the L’Hopital’s rule of calculus for $n \to \infty$, the first term $0.1 \frac{\log_2 n}{n^{\varepsilon}}$ also tends to zero:
  
  $$
  \lim_{x \to \infty} \frac{\log_2 x}{x^{\varepsilon}} = \lim_{x \to \infty} \frac{\log_2 e}{e x^{1+\varepsilon}} = \lim_{x \to \infty} \frac{\log_2 e}{x^{\varepsilon}} = 0
  $$

  where $e = 2.71828 \ldots$ is the base of the natural logarithms.

6. (5 marks) Suggest which of the two software packages, A and B, of the same price should be bought to maintain large data bases having each up to $10^9 \approx 2^{30}$ records. As was found empirically, the average time to process $n$ records with A and B is $T_A(n) = 0.1n \log_2 n$ milliseconds and $T_B(n) = 2.5n$ milliseconds, respectively. Decide which package is better in “Big-Oh” sense, work out exact conditions, in terms of the database size $n$, when this package outperforms the other, and recommend the best choice in your case.

- (1 mark) In “Big-Oh” sense, the package B is better (linear vs. $n \log n$ time complexity).

- (3 marks) The package B begins to outperform the package A when $T_B(n) \leq T_A(n)$, that is, when $2.5n \leq 0.1n \log_2 n$, or $\log_2 n \geq 25$, or $n \geq 2^{25}$.

- (1 mark) Therefore, to maintain the databases of size $2^{30}$, the package B is the best choice.

7. (5 marks) Derive a closed-form formula for $T(n)$ by solving the recurrence relation

$$
nt(n) = (n + 1)T(n - 1) + c
$$

with the base condition $T(0) = 0$. 

(a) (0.5 marks) $T(n) = n^5 \left( 0.003 + \frac{1}{n^2} + 100 \frac{1}{n^5} \right)$ so that the dominant term is $0.003n^5$.

(1 mark) Thus, $0.003n^5 < T(n) \leq 101.003n^5$ for $n \geq 1$, so that $T(n)$ is $\Theta(n^5)$.

(b) (0.5 marks) Both terms are dominant (because $\log_2 n = \log_2 128 \cdot \log_{128} n = 7 \log_{128} n$ and $T(n) = 107 \log_{128} n -$ but this proof is not expected).

(1 mark) Thus, $106 \log_{128} n \leq T(n) \leq 108 \log_{128} n$ for $n \geq 1$, so that $T(n)$ is $\Theta(\log n)$
Hint: you may need to prove, e.g. by the math induction, that \( \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \ldots + \frac{1}{n(n+1)} = \frac{n}{n+1} \).

- **(1.5 marks)** By dividing both sides of the given recurrence by \( n(n+1) \), the given recurrence is represented in the following form:
  \[
  \frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{c}{n(n+1)}
  \]
  which is more convenient for telescoping.

- **(2 marks)** The telescoping (note that an explicit system of its equations is not expected and should receive the same marks) yields the closed form formula
  \[
  \frac{T(n)}{n+1} = \frac{T(0)}{1} + c \left( \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \ldots + \frac{1}{n(n+1)} \right) = c \frac{n}{n+1}
  \]
  so that \( T(n) = cn \).

- **(1.5 marks)** The math induction can be used to prove that the sum in parentheses is equal to \( \frac{n}{n+1} \).
  - (0.5 marks out of 1.5) The base case holds for \( n = 1 \): \( \frac{1}{1 \cdot 2} = \frac{1}{2} \).
  - (1 mark out of 1.5) By the induction hypothesis, let the relationship \( S_n = \frac{1}{1 \cdot 2} + \ldots + \frac{1}{n(n+1)} = \frac{n}{n+1} \) hold for \( n = k - 1 \), i.e. \( S_{k-1} = \frac{k-1}{k} \). Then
    \[
    S_k = S_{k-1} + \frac{1}{k(k+1)} = \frac{k-1}{k} + \frac{1}{k(k+1)} = \frac{(k-1)(k+1) + 1}{k(k+1)} = \frac{k^2}{k(k+1)} = \frac{k}{k+1}
    \]
    what is to be demonstrated.

8. **(5 marks)** Assuming \( n = 10^m \) with the integer \( m = \log_{10} n \), derive a closed-form formula for \( T(n) \) by solving the recurrence relation \( T(n) = 10T(n/10) + 5 \) with the base condition \( T(1) = 0 \).

- **(4 marks)** Telescoping the recurrence \( T(10^m) = 10T(10^{m-1}) + 5 \) yields the closed-form formula \( T(10^m) = 10^m T(1) + 5(1 + 10 + 10^2 + \ldots + 10^{m-1}) \).
  - **(1 mark)** Therefore, \( T(10^m) = 5 \frac{10^m - 1}{10-1} = \frac{5}{9}(10^m - 1) \), or \( T(n) \approx 0.55n \).

  *If the sum \( 1 + x + \ldots + x^{p-1} \) in parentheses is not reduced to \( \frac{x^p - 1}{x - 1} \), but the solution indicates clearly that \( T(n) \) is linear by \( n \), the last step should receive its mark.*

9. **(5 marks)** You need to select \( k = 100 \) most successful students from an unordered array of academic records of \( n = 1,000,000 \) students all over the world. Each record contains a non-negative GPA (Grade Point Average) score specifying how successful the student is. You know (possibly, from your course COMPSCI.220.SIT) two options for selecting the \( k \) higher-rank students:

   - **(a)** to run \( k \) times the quickselect algorithm with linear processing time, \( T_{\text{select}}(n) = cn \), in order to select each time the next rank (i.e. the ranks \( n, n-1, \ldots, n-k+1 \)), or
(b) to run once the quicksort algorithm with \( n \log n \) processing time, \( T_{\text{qsort}} = cn \log_2(n) \),
to sort the array in ascending order of GPAs and fetch the \( k \) higher-rank entries.

Providing the factors \( c \) are the same for both the algorithms and the fetching time is
negligibly small comparing to the sorting time, which option does result in the fastest
selection?

- \((3.5 \text{ marks})\) The option (a) is better if \( kT_{\text{qselect}} \leq T_{\text{qsort}} \), that is, \( ckn \leq cn \log_2 n \), or \( k < \log_2 n \).
- \((1.5 \text{ marks})\) Because in this case \( k = 100 > \log_2 10^6 \approx 20 \), the option (b) ensures the
fastest selection.

10. \((5 \text{ marks})\) Convert the array \([30, 5, 25, 10, 15, 0, 20, 40, 35, 45]\) of the size 10 into the maximum heap, delete the maximum key, and restore the heap order for the remaining nine items.

<table>
<thead>
<tr>
<th>Array position:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Steps: pos.5</th>
<th>45</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos.4</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>pos.3</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>pos.2</td>
<td>45</td>
<td>5</td>
</tr>
<tr>
<td>pos.1</td>
<td>45</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heap:</th>
<th>45</th>
<th>40</th>
<th>25</th>
<th>35</th>
<th>15</th>
<th>0</th>
<th>20</th>
<th>10</th>
<th>30</th>
<th>5</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Delete max:</th>
<th>5</th>
<th>40</th>
<th>25</th>
<th>35</th>
<th>15</th>
<th>0</th>
<th>20</th>
<th>10</th>
<th>30</th>
<th>–</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restore heap:</td>
<td>40</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heap:</th>
<th>40</th>
<th>35</th>
<th>25</th>
<th>30</th>
<th>15</th>
<th>0</th>
<th>20</th>
<th>10</th>
<th>5</th>
<th>–</th>
</tr>
</thead>
</table>

Comments need not be in tabular form, but otherwise they should explain in detail
how the initial array is converted into the heap, how the maximum value is deleted,
and how the reduced heap is restored. Every erroneous position of key values is penalised
by \(-0.25\) marks.

11. \((5 \text{ marks})\) Place the key \( k = 42 \) into the hash table of size 11 using the modulo-based hash
address \( i = k \) modulo 11 and double hashing with backward step \( \Delta = \max\{1, \lfloor k/11 \rfloor \} \)
where \( \lfloor z \rfloor \) is the largest integer smaller than or equal to \( z \). Assume that just before you
place the key 42, the array is already filled as follows (“–” indicates free locations):

<table>
<thead>
<tr>
<th>address</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>55</td>
<td>45</td>
<td>–</td>
<td>25</td>
<td>–</td>
<td>–</td>
<td>17</td>
<td>–</td>
<td>–</td>
<td>31</td>
<td>–</td>
</tr>
</tbody>
</table>
• (0.5 mark) For the key \( k = 42 \), the hash address \( i = 42 \mod 11 = 9 \) and the backward step \( \Delta = \max\{1, \lfloor 42/11 \rfloor \} = 3 \).

• (3.5 marks) Successive steps of placing the key to this hash table are as follows (every erroneous step is penalised by \(-0.25\) marks):
  - the initial address \( i = 9 \) – collision;
  - the backward step of double hashing to the address \( (9 - 3 = 6) \) – collision;
  - the next backward step to the address \( (6 - 3 = 3) \) – collision;
  - the next backward step to the address \( (3 - 3 = 0) \) – collision;
  - the next wrap-around address \( (0 - 3) \mod 11 = 8 \) – the empty place is found to insert the key.

• (1 mark) The resulting hash table is as follows:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>55</td>
<td>45</td>
<td>–</td>
<td>25</td>
<td>–</td>
<td>–</td>
<td>17</td>
<td>–</td>
<td>42</td>
<td>31</td>
<td>–</td>
</tr>
</tbody>
</table>

Submission

*The due date is Thursday, 29th March 2007, 8:30 p.m. (ADB time)* [then penalty linearly grows in time from 0% to 50% on 31st March 2007, 8:30 p.m.; no submission afterwards].

The markers should evaluate only solutions in each submission; the course administrator will account for the above penalty.