A. V. Gerbessiotis  
CS 610-102  
March 2015  
Spring 2015  
Exam 2 (333 points)  

Name: .................................................................
ID Number: ......................... Exam Type: ...A... Exam Number: ...........
Grade: 1: ... 2: ... 3: ... 4: ... 5: ... 6: ... 7: ... 8: ... 9: ... 10: ... Total: .......

**Solve all the problems** in the space provided

**There are 6 (SIX) pages this page included**

Read and Sign the statement below at the end of the exam

Unsigned exams will be marked with grade 0 (zero).

**STATEMENT**

On my honor, I pledge that I have not violated the provision of the NJIT Student Honor Code.

Sign below at the end of the exam

Signature .................................................................

List of useful formulae

\[
\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}, \quad n! \approx \left(\frac{n}{e}\right)^n, \quad a^{\log_a n} = n^{\log_a a},
\]

For \(x \neq 1\), we have that

\[
\sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x - 1}, \quad \sum_{i=0}^{n-1} ix^i = \frac{(n-1)x^{n+1} - nx^n + x}{(1-x)^2}, \quad \sum_{i=0}^{\infty} i \cdot (1/2)^i = 2
\]

**NOTE:** Give an algorithm means, present the algorithm, show that it works as claimed, and analyze its running time by providing an asymptotically tight bound.

**Master Method.** \(T(n) = aT(n/b) + f(n)\), such that \(a > 0, b > 1, f(n) > 0\).

M1 If \(f(n) = O(n^{\log_b a - \epsilon})\) for some constant \(\epsilon > 0\), then \(T(n) = \Theta(n^{\log_b a})\).

M2 If \(f(n) = \Theta(n^{\log_b a} \lg^k n)\), then \(T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)\), where \(k \geq 0\) is a non-negative constant.

M3 If \(f(n) = \Omega(n^{\log_b a + \epsilon})\) for some constant \(\epsilon > 0\), and if \(af(n/b) \leq cf(n)\) for some constant \(c < 1\) and for large \(n\), then \(T(n) = \Theta(f(n))\).
Problem 1. (6 POINTS)
Answer with a TRUE or a FALSE or give a brief answer.

(a) Did you write your ID on page 1?
(b) Name 3 divide-and-conquer algorithms introduced in class. (3 points)
(c) Name 2 incremental algorithms introduced in class. (2 points)

Problem 2. (30 POINTS)
Give a TRUE or FALSE for each one of the statements below. Any answer other than a full TRUE or FALSE will be considered wrong. All algorithms are the book/notes implementations.

(1) MergeSort is stable.
(2) QuickSort is stable.
(3) On the sorted sequence ⟨1, 2, . . . , n⟩, InsertionSort is asymptotically faster than QuickSort.
(4) On the reverse-sorted sequence ⟨n, n − 1, . . . , 1⟩, HeapSort is asymptotically faster than QuickSort.
(5) The asymptotic solution of \( T(n) = T(n/2) + \log n \) is \( T(n) = \Theta(\log n) \).
(6) Heap-Sort sorts in-place.

Problem 3. (30 POINTS)
Solve the following recurrence relations by providing asymptotically tight bounds. You only need to provide the bound, intermediate derivations are not required. If no boundary case is given, the choice of the constants is yours. You may assume that \( T(n) \) is positive and monotonically increasing, if you need to do so.

\[
\begin{align*}
(1) \quad T(n) &= T(n/2) + n^2. \\
(2) \quad T(n) &= 16T(n/4) + n \\
(3) \quad T(n) &= T(n - 1) + \log n.
\end{align*}
\]
Problem 4. (35 points)
For this problem input size is specified separately for each case and thus Row-1, Column-1 indicates a problem size of \( n^2 \) keys. Graphs have \( n \) vertices and \( m \) edges and represented using an adjacency matrix. Find the worst-case running time of every (efficient) algorithm in Column-1 by using the time bounds in Column-2; use only class-introduced algorithms. If multiple answers are possible, use the asymptotically tightest bound. The same answer in Column-2 can be reused. Row 7 is filled for you; you should fill the remaining rows. If you see \( \lg^t n \), it means \((\lg n)^t\).

<table>
<thead>
<tr>
<th>Column-1</th>
<th>Answer</th>
<th>Column-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: BuildMaxHeap of ( n^2 ) keys</td>
<td>A. ( O(\lg n) )</td>
<td></td>
</tr>
<tr>
<td>2: Sorting ( n \lg n ) keys.</td>
<td>B. ( O(n^4) )</td>
<td></td>
</tr>
<tr>
<td>3: QuickSort of ( n \lg n ) keys</td>
<td>C. ( O(n^2) )</td>
<td></td>
</tr>
<tr>
<td>4: Binary Search on ( 2^n ) sorted keys</td>
<td>D. ( O(n) )</td>
<td></td>
</tr>
<tr>
<td>5: InsertionSort on ( n^2 ) keys</td>
<td>E. ( O(n \lg n) )</td>
<td></td>
</tr>
<tr>
<td>6: Find the outdegree of vertex ( u )</td>
<td>F. ( O(n \lg^2 n) )</td>
<td></td>
</tr>
<tr>
<td>7: MergeSort of ( n ) keys</td>
<td>E</td>
<td>G. ( O(n^2 \lg^2 n) )</td>
</tr>
<tr>
<td>8: MergeSort of ( n \lg n ) keys</td>
<td>H. ( O(n^3 \lg^2 n) )</td>
<td></td>
</tr>
</tbody>
</table>

Problem 5. (36 points)
Solve exactly the following recurrence. You may assume that \( n \) is a power of two.

\[
T(n) = 2T(n/2) + 12n, \quad T(2) = 4.
\]
Problem 6. (30 points)
A set \( X \) of \( n \) integers is given and also integer \( q \). Determine in worst case time \( o(n^2) \) (little oh) whether there exist two different integers \( a, b \) of \( X \) so that \( a - b + q = 0 \). (You may assume all integers are distinct.) 'Determine' means you print ONE pair \( a, b \) if it exists or 'NOTFOUND' if no such pair exists.

Problem 7. (60 points)
(a) Perform inorder, preorder and postorder traversals on the following tree.

```
    5
   / \
  3   8
 / \ / \
1  4 6  7
 \   
  2
```

(b) Compress the 20-character long text DATADATADATAMDATUMDU using Huffman coding. How many bits to represent the encoded string only?
Problem 8. (46 points)

Fill in the table below by providing the running time for every one of the indicated operations and every one indicated data structure of size $n$. Express the answer in terms of $n$. Fill entries by writing $n$ to denote running time of $O(n)$ or $\Theta(n)$, 1 to denote running time of $\Theta(1)$, and so on. All keys already in data structure are distinct.

We fill in the entry for Succ($z$) into an SDL. Since it is $\Theta(1)$ we write 1.

- (a) **Ins** is Insertion of a key $x$ into data structure,
- (b) **Del** is Deletion of a record of the data structure pointed by a pointer/reference (for linked structures) or index (for array-based structures) $z$, i.e. the location that will be deleted is known,
- (c) **Sea** is Search for key $x$.
- (d) **FindMin** is FindMinimum, i.e. determine and return the value of the key with the smallest value among the keys of the data structure; the key remains in it.
- (g) **ExMax** is ExtractMax, that is return and remove the key with largest value.
- (h) **Succ** is the successor of $z$, i.e. the smallest among all the elements greater than or equal to the value pointed by pointer/reference $z$ (it has nothing to do with pointers/references).

For removal operations the data structures must be maintained in likewise form after the operation (i.e. there are no gaps or locations with no keys).

<table>
<thead>
<tr>
<th></th>
<th>SA</th>
<th>RSL</th>
<th>SDL</th>
<th>MNH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ins</strong>($x$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Del</strong>($z$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sea</strong>($x$)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FindMin</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ExMax</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Succ</strong>($z$)</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Problem 9. (30 points)
You may assume that \( \lg n \) is integer in this problem. You are given two sorted sequences stored in arrays \( A \) and \( B \) of \( \lg n \) and \( n \) keys respectively. We are interested in forming the union \( C = A \cup B \), i.e. an array \( C \) containing all keys of \( A \) and \( B \) but duplicates are included only once. Give a time efficient algorithm for solving this problem. (Time efficient means its running time in terms of key comparisons cannot be asymptotically improved upon.) Justify your arguments.

Problem 10. (30 points)
(Time efficient means asymptotic time efficiency cannot be improved upon. Space efficiency is to be interpreted likewise.)
You are given \( n \) keys in array \( A \). The keys have no particular order. We would like to determine the \( \sqrt{n} \) smallest of the \( n \) keys and display them in the output in sorted order (smaller to larger). Give a space and time efficient algorithm, explain the algorithm, analyze its worst-case running time and space complexity. Justify your arguments.