Practice final for EECS 380, 2001: Prof Markov

Available in Postscript and PDF

Total pages: 5
Exam duration: 1hr 50min.
Write your name and uniqname on every sheet, including the cover.

Maximum score: 100 points + 15 extra. Extra credit points do not affect the curve.
To be eligible for extra credit, you need to earn at least 70 regular points.

All complexity estimates are for runtime (not for memory), unless specified otherwise.

1. 30 points. **Algorithmic Complexity**

   Each line in the table corresponds to an algorithm or an algorithmic problem. Write \( P \) for problems and \( A \) for algorithms. A problem gives input and output, but an algorithm additionally entails a particular method of achieving this output. Fancy data structures (e.g., heaps, BSTs and hash-tables) often imply specific algorithms. Simple containers (e.g., arrays and linked lists) are typically used to store input or output and may restrict possible algorithms.

   For each algorithm, write its Theta-complexities.
   For each problem, write Theta-complexities of a best possible algorithm that solves the problem.
   There can be multiple correct answers, especially, if there is a trade-off between average-case and worst-case performance.
   **No explanation necessary.**

   You can assume that \( \text{operator<} \) and \( \text{operator==} \) for values stored in containers run in \( O(1) \) time.
   You cannot make any additional assumptions about algorithms/problems unless instructed by Prof. Brehob or Prof. Markov.

   Each line is worth 2 points. Each wrong or missing answer on a line costs -1 point.
   Minimum per line = 0 points.
<table>
<thead>
<tr>
<th>Algorithm or Problem:</th>
<th>Best-case Theta()</th>
<th>Avg-case Theta()</th>
<th>Worst-case Theta()</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Find a given value in an unsorted N-by-N matrix.</td>
<td><strong>P</strong></td>
<td><strong>N^2</strong></td>
<td><strong>N^2</strong></td>
</tr>
<tr>
<td>2. Binary search over N elements</td>
<td><strong>A</strong></td>
<td><strong>log N</strong></td>
<td><strong>log N</strong></td>
</tr>
<tr>
<td>3. Find the largest element in an unsorted array with N elements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Print all values appearing at least twice in a sorted stack of size N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. Insert a new element into a sorted singly-linked list with N elements so that the list remains sorted</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Given two unsorted arrays of N and N/10 elements, say whether they have at least one common element</td>
<td></td>
<td>don’t bother</td>
<td></td>
</tr>
<tr>
<td>7. Shaker sort of a doubly-linked list with N elements, using &quot;early termination&quot;.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Duplicate a queue of N elements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. One invocation of the <code>partition()</code> function used in the quicksort algorithm. Assume in-place partitioning of a complete array with N elements using a given pivot</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. Given a pointer to an element in a singly-linked list with N elements, remove that element from the list</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. Sort N 8-bit characters stored in an array. Only <strong>O(1)</strong> additional memory allowed (in-place sorting)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. Remove the middle element from an unsorted array of N elements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. Compute N! for a given N using a straightforward recursive algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14. Find the combination of N decimal digits that opens a bank safe. The safe opens when you enter the right combination, and you can try as many combinations as you wish. No other feedback is available</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. Print all diagonal values of a given N-by-N matrix</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. 10 points. **STL**

Fill in the blanks

a. "STL" stands for ____________________________

b. A *range* can be defined by two ________________________

c. **STL**’s `sort()` and `binary_search()` functions take an optional ___________ function-object

d. One can use class _________ from STL as an implementation of Abstract Symbol Table.

e. **Iterators** of linked list classes in STL **do not** allow _________ access.
3. **20 points. Fancy containers (heaps, generic trees, search trees, hash-tables, etc)**

   a. **10 points.** Follow instructions from Question 1.

<table>
<thead>
<tr>
<th>Algorithm or Problem:</th>
<th>Best-case ( \Theta() )</th>
<th>Avg-case ( \Theta() )</th>
<th>Worst-case ( \Theta() )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Print all values stored at nodes of a given tree with ( N ) nodes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Convert a binary heap of ( N ) elements into a sorted array</td>
<td>don’t bother</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Test whether a given array with ( N ) values is in a binary-heap order</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. One search in a BST of ( N ) elements. Assume that the tree is perfectly balanced and the search results in a miss</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. One successful look-up in a hash table with ( N ) elements and load ratio * 1.0. The hash-table uses separate chaining with singly-linked lists. Assume that hash-function can be computed in ( O(1) ) time. Note: elements contained in the hash-table may be poorly dispersed.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* The load ratio of a hash-table with \( N \) elements and \( M \) buckets is \( N/M \).

b. **5 points.** Consider `struct Key { char p1, p2, p3 };`
   and the following hash-functions (modulo hash-table size).
   1. `unsigned f1(struct Key& s) { return s.p1+5*s.p2; }
   2. `unsigned f2(struct Key& s) { return 10*s.p1+100*s.p2+1000*s.p3; }
   3. `unsigned f3(struct Key& s) { return 11*s.p1+101*s.p2+1001*s.p3; }

Assume a hash-table of size 1250 with linear probing. Mark each hash-function as **good** or **bad**. Use space below to explain.
c. 5 points. Fill in the blanks.

**Markov section only**

In BSTs, ______ and ______ rotations have time complexity Theta(____).

They are explicitly used in ______ insertion and ______ algorithms. Two
BSTs can be joined using a ______ algorithm, which applies ______
_______ to one of the trees. The worst-case complexity of such a join
algorithm is Theta(____), but the best case can be faster when
______________________________________________________.

**Brehob section only**

Each node in a 2-3-4 tree has ____, ____ or ____ keys in it. __________
trees are an implementation of 2-3-4 trees. Insertion into a 2-3-4 tree has
worst-case complexity Theta(____) and search has worst-case complexity
Theta(____).

4. 20 points. **Algorithm design: Recursion / Divide and Conquer / Dynamic Programming**

Implement the following C++ function

```cpp
void makeBalancedBST(unsigned *begin, unsigned numElem);
```

which takes an unsorted array and makes a balanced BST out of it, stored left
to right so that children of element k be 2*k and 2*k+1. You must achieve
worst-case complexity O(numElem log²(numElem)) and explain how you did it. 15
points for the case when numElem is a power of two minus one (say, 3, 7 or 15),
5 additional points for the general case. Use a separate page.

5. 20 points. **Questions related to HWKs and Projects**

a. 5 points. Provide a dictionary produced by the Huffman algorithm applied
to this input: AAABAABCCDCC. No explanation necessary.

b. 5 points. Heapify the digits of your student ID. Start with the digits in
the original order and show the process step by step.

c. 10 points. You are given a function that takes N planar points and returns
all points on the boundary listed clockwise. Provide an algorithm (in
pseudocode or valid C++) that sorts N doubles using that function and
spends O(N) time outside that function.
6. Extra credit: 15 points. ‘Comments not available’.

In this question you are given a printout of a C++ function, with coke spilled over the comments (=> you can’t read the comments). You need to explain what the function does, illustrate by several representative examples, give worst-case/best-case Theta() for runtime and substantiate these complexity estimates.

```cpp
int L2(const char * A, const char * B)
// COMMENTS NOT AVAILABLE
{
    int m=strlen(A), n=strlen(B), i, j;
    int L[m+1][n+1]; // g++ extension to C++
    for (i = m; i >= 0; i--)
        for (j = n; j >= 0; j--)
        {
            if (A[i] == '\0' || B[j] == '\0') { L[i][j] = 0; }
            else if (A[i] == B[j]) L[i][j] = 1 + L[i+1][j+1];
            else L[i][j] = max(L[i+1][j], L[i][j+1]);
        }
    j=L[0][0];
    return j;
}
```

Source code courtesy of Prof. David Epstein.