CPSC 200 Fall 2002
Midterm I— 7 October 2002

Name(Printed) : ______________________

Signature : ______________________

StudentNumber : [ ][ ] [ ][ ][ ] [ ][ ][ ][ ]

BETA	BIRD	BLOT	BOOK	BREW	CAMP
INCH	IRIS	ISLE	KERN	KILN	KITE
MANX	MESH	MINK	MOTH	MOVE	MUSK
PARK	PINE	POET	RAFT	REED	RING
RUBY	RUFF	SEAM	SEED	SHOP	SILK
SINE	SNIP	SOAP	STUB	TASK	TAXI
WICK	WOLF	WRIT	YARD		

Question	Score
1	/5
2	/4
3	/3
4	/3
5	/6
6	/8
7	/12
8	/4
9	/3
10	/2
Total	/50

- Write the word circled above on each page of your exam. Do not put any other identifying marks on any page of your exam. Failure to put the circled word on a page of your exam may result in no marks being awarded for that page.
- *Read each question carefully. Ask yourself what the point of the question is. Check to make sure that you have answered the question asked.*
- This is a **50** minute exam. This exam contains **7** pages of questions not including this cover page. Make sure that you have all of them.
- Answer all questions on the exam sheet. If you do some of your work on the back of a page, clearly indicate to the marker what work corresponds with which question.
- Partial marks shall be awarded for clearly identified work.
- Non-programmable calculators are allowed.
- This exam counts as **20%** of your total grade. There are **50** points total on the exam.

True False

(1each) **1.** Indicate whether or not the following statements are true or false by circling the appropriate word. No marks shall be awarded if the indication is in any way ambiguous.

(a) $\log_2(n) = \Theta(\log_{10}(n))$. **TRUE** **FALSE**

(b) In order to find the maximum subsequence of an array of size n, the maximum subsequence sum algorithm 3 (the recursive algorithm) uses $\Theta(n \log n)$ calls. **TRUE** **FALSE**

(c) Tail call optimization is primarily useful because it makes tail-recursive algorithms faster. **TRUE** **FALSE**

(d) The fastest subsequence sum algorithm is $o(n \log n)$. **TRUE** **FALSE**

(e) $\log(n!) = o(\log(n^n))$. **TRUE** **FALSE**

Asymptotic Analysis

(4) **2.** Suppose that $\lim_{n\to\infty} f(n)/g(n)$ either exists or goes to $+\infty$. Match up the following statements:

A. $f(n) = O(g(n))$	(a) $\lim_{n\to\infty} f(n)/g(n) = 0$
B. $f(n) = \Omega(g(n))$	(b) $0 \leq \lim_{n\to\infty} f(n)/g(n) < +\infty$
C. $f(n) = \Theta(g(n))$	(c) $0 < \lim_{n\to\infty} f(n)/g(n) < +\infty$
D. $f(n) = o(g(n))$	(d) $0 < \lim_{n\to\infty} f(n)/g(n) \leq +\infty$

A. matches ____. **B.** matches ____. **C.** matches ____.

D. matches ____.

(3) **3.** Consider the following functions of n: • $16\,n+5$ • $(1.01)^n$ • $n(\log n)^2$ • $300+\log_2 n$ • n^n and • n^2 . List these in order of growth, from slowest growing to fastest growing.

(3) **4.** Consider the following functions of n: • $n \log n$ • $n(\log n)^2$ • $n(\log n^2)$ • $\log(n!)$ • n^2 and • $n^2/(\log n)$. State which, if any, of these functions are Θ-equivalent to other functions in the list and what the equivalences are.

(2) **5.** (a) Write down the formal definition (*not* the limit definition) for $f(n) = \mathrm{O}(g(n))$.

(4) (b) Prove, using the formal definitions, that if $f(n) = \mathrm{O}(g(n))$ then $f(n) = \mathrm{O}((1 - \frac{1}{n})g(n))$.

Variants and Invariants

6. This question refers in part to the code in Figure 1.

Figure 1: Another `isSorted` function (for Question 6).

```
// isSorted -- return true if  v is sorted.
bool isSorted(const vector<string>& v)
{
    bool sorted = true ;
    for(int i=0,e=v.size()-1;sorted&&i<e;++i)
        {
        sorted = sorted && (v[i] <= v[i+1]) ;
        }
    return sorted ;
}
```

(2) (a) What is a *loop variant*?

(2) (b) What is a loop variant for the loop in the code shown in Figure 1?

(2) (c) What is a *loop invariant*?

(2) (d) What is a non-trivial loop invariant for the code shown in Figure 1?

Figure 2: The `isSorted` function.

```
// isSorted -- return true if  v is sorted.
#include "isSorted.h"

bool isSorted(const vector<string>& v)
{   return isSorted(v, 0, v.size()) ; }

bool isSorted(const vector<string>& v,
              int left, int size)
{
    if (size<2)
        return true ;

    int newsize = size/2 ;
    int middle = left + newsize ;
    return (v[middle-1] <= v[middle])
        && isSorted(v,left,newsize)
        && isSorted(v,middle,size-newsize) ;
}
```

Recursion

7. This question refers to the code in Figure 2.

(2) (a) What are the base cases of the 3-argument version of `isSorted`? Are the base cases coded correctly?

(3) (b) State what else you must check in order to check the correctness of a

recursive algorithm. Is this algorithm correct as written?

(1) (c) Let $T(n)$ be the *worst-case* time for the 4-argument `find` function, where $n =$ `size`. Explain why $T(n)$ is given by the equations

$$T(0) = c_1 \qquad T(1) = c_1$$
$$T(n) = c_2 + 2T(n/2) \qquad \text{for } n \geq 2.$$

(2) (d) Use telescoping sums to get a Θ-estimate for $T(n)$.

(2) (e) Estimate (to Θ) the worst-case space usage of `isSorted`.

(2) (f) Is the best-case time for `isSorted` $\Omega(n)$? Why or why not?

Sorting

(4) **8.**

Number of items	*time* (s)
5 000	0.44s
10 000	1.7 s
15 000	3.8 s
20 000	6.7 s

A user implementing insertion sort and observes the average run times shown to the left. Do the times, and in particular the ratio of the times, seem reasonable to you?

(3) **9.** As a function of n, how many inversions are there in the list?

$$[3, 2, 1, 6, 5, 4, 9, 8, 7, \ldots, 3n, 3n-1, 3n-2]$$

As a function of n what kind of running-time would you expect from insertion sort on the above array?

(1) **10.** (a) Define what it means for a sort to be stable.

(1) (b) Why is stability a useful property?