DUKE UNIVERSITY
Department of Computer Science

CPS 100 **J. Forbes**
Fall 2001

Test #1

Name: _____

Login: _____

Honor code acknowledgment (signature) _____

	value	grade
Problem 1	12 pts.	
Problem 2	12 pts.	
Problem 3	18 pts.	
Problem 4	26 pts.	
Problem 5	8 pts.	
Problem 6		
Problem 7	24 pts.	
Free!		6pts.
TOTAL:	100 pts.	

This test has 7 questions on 8 pages. Be sure your test has them all.

This is an open-book test. You have at least 50 minutes to complete it. That means you should spend no more than *30 seconds per point.* If the number You may consult any books, notes, or other inanimate objects (other than computers) available to you. You may use any program text supplied in lectures, assignments, or solutions. In writing code you do not need to worry about specifying the proper `#include` header files. Assume that all the header files we've discussed are included in any code you write.

Please write your answers in the spaces provided in the test. Make sure to put your name, login, and lab section in the space provided below. Put your login and initials *clearly* on each page of this test and on any additional sheets of paper you use for your answers.

Don't panic. Just read all the questions carefully to begin with, and first try to answer those parts about which you feel most confident. Do not be alarmed if some of the answers are obvious.

PROBLEM 1 : (*The Sorted Link (12 points)*)
Using the following definition of a linked list node, write a function that inserts a string into a sorted linked list.

```
struct Node
{
  string info;
  Node * next;

  Node(const string& s, Node * ptr)
    : info(s), next(ptr)
  { }
};

Node* InsertSorted(Node *list, const string & word)
// pre: list (may be empty) is in sorted order
// post: list still sorted, new node with word added
{









}
```

PROBLEM 2 : (*Piling it on (12 points)*)
Given the following interface to `tstack`:

```
template <class Type>
class tstack
{
  public:
    tstack( );                          // construct empty stack

    const Type & top( )          const;   // return top element (NO pop)
    bool             isEmpty( )  const;   // return true if empty, else false
    void push( const Type & item );   // push item onto top of stack
    void pop( Type & item );          // pop top element into item

};
```

Draw a box and pointer diagram of what `list` points to after executing the following code fragment:

```
string *name = 0;
tstack<string*> s;
Node *list = 0;

s.push(new string("Nelson"));
s.push(new string("Malcolm"));
s.push(new string("Martin"));
s.push(new string("Marcus"));

*(s.top()) = "Patrice";
name = new string("");
while (!s.empty())
  {
    s.pop(name);
    list = new Node(name, list);
  }
```

PROBLEM 3 : (*Pledge Oh Theta Omega! (18 points)*)
For the following, pick the correct relation (O, Ω, or Θ) and give justification in the form of correct constants for the definition of $O(\cdot)$, $\Omega(\cdot)$ or $\Theta(\cdot)$.

For example:

$$n \in \qquad n^n$$

Answer: $\mathbf{O}(n^n)$ for $\mathbf{c = 1}$ and $\mathbf{n_0 = 1}$. That is, $n \leq 1 \cdot n^n$ for all $n > 1$.

a. (6 points) $n^3 + 100n^2 \in \qquad (n^4)$

b. (6 points) $n + n \lg n \in \qquad (n \log n)$

c. (6 points) $2^n \in \qquad (3^n)$

PROBLEM 4 : (*The All Powerful Registrar (26 points)*)

You are in charge of scheduling classes for the Love Auditorium. Your job is to schedule as many classes as possible. You are given a `tvector` of `Intervals`. An `Interval` is a start and end time (in integer time units) described below:

```
struct Interval
{
  int start; // begin time (in minutes after midnight)
  int end;   // end time (in minutes after midnight)
  Interval(int a, int b)
    : start(a), end(n)
  { }
};
```

Luckily, your friend has taken CPS 100 and can describe an algorithm that yields the maximum number of intervals without conflicts in pseudo-code:

(1) Sort all of the intervals in increasing order by *finish* time, breaking ties by length of the interval.
(2) Loop starting from the first element until you reach the end of the vector
  (3) Let x be the current item in the vector.
  (4) Add x to your result vector
  (5) Skip over all intervals that overlap with x

As an example, if the proposed class intervals were:

$$(2,5),(1,3),(1,2),(1,4),(3,5),(4,5),(6,7),(3,4).$$

The sorting would yield:

$$(1,2),(1,3),(3,4),(1,4),(4,5),(3,5),(2,5),(6,7)$$

and the best set of intervals would then be:

$$(1,2)(3,4)(4,5),(6,7).$$

a. (10 points) In thinking about step (1), you reflect back on your vast experience with *Anagram*. You then realize that you can just use `QuickSort` to sort and define the less than operator ($<$). Define the $<$ for the Interval struct for the requirements of the sort described in step (1).

```
 bool operator < (const Interval& lhs, const Interval& rhs)
// post: return true if lhs < rhs
{
  // FILL IN




}
```

b. (12 points) Now, it is time to sit down and code up the algorithm. Your friend was also kind enough to give you a function that determines if two intervals overlap.

```
bool IntervalsOverlap(const Interval & a, const Interval & b)
// pre: a and b are both proper nonzero intervals where a.end > a.start
// post: returns whether a and b overlap
{
    if (if a.begin >= b.end || b.begin >= a.end)
        return false;
    else
        return true;
}
```

She also started the scheduling function for you. Finish it so that it implements her pseudo-code.

```
void ScheduleClasses(tvector<Interval> classList,
                     tvector<Interval>& result)
// pre: classList contains legal intervals
        result is empty
// pos: result contains elements
{
  int i=0;
  Interval x;


  // Sort the list
  QuickSort(classList, classList.size());

  while (i < classList.size())
    {
      x = classList[i];
      // add x to the result list
      // FILL IN









      // skip over all overlapping intervals and increment i
      // FILL IN
```

```
    }


}
```

c. (4 points) What is the Big-Oh of `ScheduleClasses` in terms of n the number of intervals in `classList`?

PROBLEM 5 : (***Thoughts on vectors and lists (8 points)***)
Describe an application where using a vector of linked lists would be more efficient than either using a vector or a linked list alone.

PROBLEM 6 : (***Cool Math (2 points EXTRA CREDIT)***)

What is

$$4 * \frac{2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \cdots 2n}{3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \cdots 2n-1}$$

as $n \to \infty$?

PROBLEM 7 : (*Summing Up (24 points)*)
In this problem, you will analyze the big-Oh for 3 solutions to the Maximum Contiguous Subsequence Problem. Given possibly negative integers $A_1, A_2, \ldots, A_n$, find the maximum value of $\sum_{k=i}^{j} A_k$. The maximum contiguous subsequence would be zero (the empty sequence) if all integers all negative. For example, if the input is $\{-2, \mathbf{11}, \mathbf{-4}, \mathbf{-13}, -5, 2\}$, then the answer is 20 for the subsequence in bold (elements 2-4). Your job is to analyze the three algorithms below, mark the line(s) of code that is executed the greatest number of times, and give the tight big-Oh bounds with justification. n is the number of elements in the vector `a`.

a. (7 points) Analyze `maxSubSum1`

```
int maxSubSum1(const tvector<int> & a )
{
    int maxSum = 0;
     for( int i = 0; i < a.size(); i++ )
      for( int j = i; j < a.size(); j++ )
        {
          int thisSum = 0;
           for( int k = i; k <= j; k++ )
            thisSum += a[ k ];
           if( thisSum > maxSum )
            {
              maxSum   = thisSum;
              seqStart = i;
              seqEnd   = j;
            }
        }
     return maxSum;
}
```

b. (7 points) Analyze `maxSubSum2`

```
int maxSubSum2(const tvector<int> & a )
{
  int maxSum = 0;
  int thisSum = 0;

  for( int i = 0, j = 0; j < a.size(); j++ )
    {
      thisSum += a[ j ];
      if( thisSum > maxSum )
        {
          maxSum = thisSum;
          seqStart = i;
          seqEnd   = j;
        }
      else if( thisSum < 0 )
        {
          i = j + 1;
          thisSum = 0;
        }
    }
  return maxSum;
}
```

c. (10 points) Give the recurrence relation and big-Oh for `maxSubSum3`.

```
/* Return maximum of three integers. */
int max3( int a, int b, int c )
{
  if (a > b)
    if (a > c)
      return a;
    else
      return c;
  else if (b > c)
    return b;
  else
    return c;
}
/* Finds maximum sum in subvector spanning a[left..right]. (*/
int maxSumRec(const tvector<int> & a, int left, int right )
{
  int maxLeftBorderSum = 0, maxRightBorderSum = 0;
  int leftBorderSum = 0, rightBorderSum = 0;
  int center = ( left + right ) / 2;

  if( left == right )  // Base case
    if (a[ left] > 0 )
      return a[left];
    else
      return 0;

  int maxLeftSum  = maxSumRec( a, left, center );
  int maxRightSum = maxSumRec( a, center + 1, right );
  for( int i = center; i >= left; i-- )
    {
      leftBorderSum += a[ i ];
            if( leftBorderSum > maxLeftBorderSum )
              maxLeftBorderSum = leftBorderSum;
    }

  for( int i = center + 1; i <= right; i++ )
    {
      rightBorderSum += a[ i ];
      if( rightBorderSum > maxRightBorderSum )
        maxRightBorderSum = rightBorderSum;
    }

  return max3( maxLeftSum, maxRightSum,
               maxLeftBorderSum + maxRightBorderSum );
}

int maxSubSum3(const tvector<int> & a )
{
  return maxSumRec( a, 0, a.size() - 1 );
}
```