Test #1

Name: ________________________________

Login: ________________

Honor code acknowledgment (signature) ______________________________

<table>
<thead>
<tr>
<th></th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>1 pt.</td>
<td></td>
</tr>
<tr>
<td>Problem 1</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>9 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>18 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 6</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Free!</td>
<td>1pt.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>100 pts.</td>
<td></td>
</tr>
</tbody>
</table>

This test has 6 questions on 10 pages. Be sure your test has them all. This test is worth 75 points. You have 75 minutes to complete it. That means you should spend no more than 1 minute per point.

This is an open-book test. You may consult any books, notes, or other inanimate objects (other than computers) available to you. Your answers may refer to any program text supplied in lectures, assignments, or solutions.

Please write your answers in the spaces provided in the test. Make sure to put your name, and login in the space provided below. Put your login clearly on each page of this test (worth 1 point) 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:  (N-List (12 points))
A N-list stores integers, the integers are stored in non-decreasing order, each integer k occurs exactly k times and N is the largest integer in the list. For example, a 4-list is (1,2,2,3,3,3,4,4,4,4) and a 5-list is a 4-list with (5,5,5,5,5) appended to the end.

a. Using the standard Node definition listed at the back of the test and replacing strings with ints, write MakeNList.

Node* MakeNList(int n)
// precondition: n > 0
// postcondition: returns the N-List

b. What is the Big-Oh of MakeNList?
PROBLEM 2:  (Analysis (9 points))

a. What is the big-Oh of the following algorithm? Briefly explain why.

```cpp
bool IsPrime(int n)
{
    int i = 3;
    if (n == 2 || n == 3)
        return true;
    if (n % 2 == 0)
        return false;
    while (i * i <= N)
    {
        if (n % i == 0)
            return false;
        i = i + 2;
    }
    return true;
}
```

b. Suppose $T_1(n) \in O(f(n))$ and $T_2(n) \in O(f(n))$. Which of the following are true:

A. $T_1(n) + T_2(n) \in O(f(n))$
B. $T_1(n) \cdot T_2(n) \in O(f(n))$
C. $T_1(n)/T_2(n) \in O(1)$
D. $T_1(n) \in O(T_2(n))$
E. none of the above

c. Which of the following functions grows fastest as $n$ grows large:

A. $\log n$
B. $n \log n$
C. $2^n$
D. $n^2$
E. $n^{20}$

d. Which of the following functions grows fastest as $n$ grows large:

A. $n + \log n$
B. $n \log n$
C. $n - \log n$
D. $n$
E. There is a tie among two or more functions for fastest growth rate
PROBLEM 3:  (*IsBalanced (12 points)*)

In the syntax of most programming languages, there are some characters that occur only in nested pairs, which are called *bracketing operators*. C++ has the following bracketing operators:

( ... )
[ ... ]
{ ... }

In a properly formed program, these characters will be properly nested and matched. To determine whether this condition holds for a particular program, you can ignore all the other characters and look simply at the pattern formed by the parentheses, brackets, and braces. In a legal configuration, all the operators match up correctly, as shown in the following example:

{ x = (s = v[1] + 2); y = 4*(v[v.size() - 1] + x ); }

The following configurations are illegal for the reasons stated:

- ( ( [ 4 ] )  The line is missing a close parenthesis.
- AB) (  The close parenthesis comes before the open parenthesis.
- { ( x} )  The parentheses and curly braces are improperly nested.

For this problem, your task is to write a function

```cpp
c bool isBalanced(string s)
```

that takes a string `s` with all characters except the bracketing operators removed. The method should return `true` if the bracketing operators in `s` are *balanced*, which means they are correctly nested and aligned, otherwise it should return `false`

You must either use a `tstack` or recursion in your solution.

Assume you have the following helper functions.

```cpp
c bool IsOpener(char ch);  // returns true if ch is ( { or [, else false
c bool IsCloser(char ch);  // returns true if ch is ) } or ], else false
c char MatchingChar(char ch);  // returns matching char.
    // In other words, returns { for },
    // } for {,
    // } for [, and so on
    // Returns ’ ’ if ch is not opener
    // or closer
```
static boolean isBalanced(String s) {

PROBLEM 4: (No Curve (18 points))
Consider a sorted Queue of elements - front is the minimum element, back is the maximum - implemented with a doubly linked list:

```cpp
class SortedQueue
{
  public:

  // constructors/destructor
  tqueue( ) // construct empty queue
              : front(0), back(0), mysize(0)
  { }
  // accessors
  const string & front( ) const;   // return front (no dequeue)
  bool isEmpty( ) const;          // return true if empty else false
  int size( ) const;              // return number of elements in queue
  // modifiers
  void enqueue( const string & item ); // insert item (at back)
  void dequeue( );                 // remove first element
  void dequeue( string & item );   // combine front and dequeue
  void makeEmpty( );               // make queue empty
  void clear( );                   // same as makeEmpty

  private:
  struct QNode
  {
    string info;
    QNode * next; // next node
    QNode * prev; // previous node

    QNode(const string & val, QNode* before, QNode* after)
    : info(val), next(before), prev(after)
    { }
  };
  QNode *front;
  QNode *back;
  int mySize;
};
```
a. Write `enqueue` so that it satisfies its postcondition. You can write helper functions for the insertion if you like.

```cpp
void SortedQueue::enqueue(const string & item)
// precondition: queue is [e1, e2, ..., en] with n >= 0
// ei <= ej if i < j
// postcondition: queue is [e1, e2, ..., en+1] with item now
// inserted in sorted order
{
```

b. Write a function that returns the median element of the queue. The median element is the \( n/2 \)th largest element. For a queue of 5 elements, the median should return the 3 largest element. For a queue of 6 element, the median should return either the 3rd or 4th largest element.

```cpp
string SortedQueue::median()
// precondition: queue has 0 or more elements
// postcondition: returns the n/2th largest element
// queue is not changed
{
```
PROBLEM 5 :  (Hashing (12 points))

Suppose you have a hash map where the keys are ints with 10 buckets and the hash function, \( h(k) = k \mod 10 \). Given the following order of input keys:

4371 1323 6173 4199 4344 9679 1989

A. Show the resulting hash map where collisions are resolved using linear probing.

B. Show the resulting hash map where collisions are resolved using quadratic probing.

C. Show the resulting hash map where collisions are resolved using chaining.
PROBLEM 6 :  *(Just like crunch (10 points))*

Write a function that removes all zeroes from a vector. Your function must operate in $O(n)$ time where $n$ is the size of the vector and cannot create a new vector.

```cpp
void RemoveZeroes(tvector<int> v)
// pre: vector is of size >= 0
// post: vector elements in same order but all zero elements have been
//       removed and vector has been resized
{   
```
Definitions

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

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

template <class Type>
class tstack
{
    public:

    tstack( ); // construct empty stack

    // accessors

    const Type & top( ) const; // return top element (NO pop)
    bool isEmpty( ) const; // return true if empty, else false
    int length( ) const; // return number of elements in stack
    int size( ) const; // return number of elements in stack

    // modifiers

    void push( const Type & item ); // push item onto top of stack
    void pop( ); // pop top element
    void pop( Type & item ); // combines pop and top
    void makeEmpty( ); // make stack empty (no elements)
    void clear(); // same as makeEmpty

    private:

    tvector<Type> myElements; // storage for stack
};
```