CS 308 Data Structures
Spring 2002 - Dr. George Bebis

Exam 1  Midterm

Duration: 1:00 - 2:15 pm

Name:

1. True/False (3 pts each) To get credit, you must give brief reasons for your answers!!

(1.1) T F Binary search is always faster than linear search.

\[ \text{linear search is faster for small sizes} \quad \text{(e.g., } N \leq 20 \text{)} \]

(1.2) T F When an array is passed to a function, the function receives a copy of the array (call by value).

\[ \text{call by reference} \]

(1.3) T F Changes in the implementation of a class should not require changes in an application that uses the class.

\[ \text{hide the implementation details of the class from the user.} \]

(1.4) T F The running time of \text{RetrieveItem} (sorted lists) is \( O(N) \)

\[ \text{if linear search is used} \quad (O(N \log N) \text{ if binary search is used}) \]

(1.5) T F An objective way to compare two algorithms is by comparing their execution (i.e., machine) times.

\[ \text{use big-O (asymptotic analysis)} \]

machine dependent!!
(1.6) T F Color images take up twice as much memory compared to gray-level images.

\[ 3 \text{ times more} \quad (r, g, b) \]

(1.7) T F An \( O(\log N) \) algorithm is slower than an \( O(N) \) algorithm.

\[ \log N < N \]

(1.8) T F The most appropriate structure to print a list of elements in reverse order is the Queue.

\[ \text{Stack is most appropriate} \]

(1.9) T F The parameter to a copy constructor must be passed by reference.

[Diagram: pan by value would lead to infinite recursion!]

(1.10) T F The running time of the program fragment shown below is \( O(N) \)

```c
sum = 0; -> O(1)
for(i=0; i<N; i++) {
    if(i > j)
        sum = sum + 1; -> O(1)
    else {
        for(k=0; k<N; k++) \{ -> O(N)
            sum = sum - 1;
        \}
    }
}
```

\[ \text{loop is executed \( N \) times:} \quad N \times O(N) = O(N^2) \]
2. Questions (5 pts each)

(2.1) Analyze the running time of the function `InsertItem` shown below (sorted list). To get credit, you need to be as specific as possible.

```cpp
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
    int location = 0;
    bool found;

    found = false;
    while( (location < length) && !found ) {
        if(item > info[location])
            location++;
        else
            found = true;
    }

    for(int index = length; index > location; index--)
        info[index] = info[index - 1];
    info[location] = item;
    length++;  
}
```

(2.2) What are the main differences between static and dynamic array allocation?

**Static:**
(i) fixed memory
(ii) stored as 1D array
(iii) stored in contiguous memory

**Dynamic:**
(i) memory is not fixed (can change from run to run)
(ii) non-contiguous memory

(See C++ review slides on static/dynamic arrays)
(2.3) Give the C++ statements for the dynamic allocation of an array with 3 rows and 5 columns. Draw a diagram that shows the structure of the dynamic array in memory.

```cpp
int **arr2D;
arr2D = new int * [3];
for (i = 0; i < 3; i++)
    arr2D[i] = new int [5];
```

(2.4) In programming assignment 1, you implemented a function that takes an image and shrinks it by a given factor. Describe in simple words how the shrink function works (no code). Assuming $N \times N$ images, give the running time of the function in terms of $N$, using big-O notation. Justify your answer.

- Running time would be $O(N^2)$ for an $N \times N$ image.

\[
\begin{align*}
N/5 + N/5 &= O(N^{1/2}) \\
\begin{align*}
\text{Iterate through rows} &\quad (N/5) \\
\text{Iterate through columns} &\quad (N/5)
\end{align*}
\end{align*}
\]

\[
\text{shrink} \ [i] \ [j] = \text{orig} \ [i * 5] \ [j * 5];
\]

\[
= \text{orig} \ [i * 5] \ [j * 5];
\]
(2.5) What are the differences between "call by value" and "call by reference"?

\[ \text{function receives a copy of the actual parameter} \]
\[ \downarrow \]
\[ \text{formal parameter becomes an "alias" of the actual parameter.} \]

(2.6) Demonstrate the binary search algorithm on the list (array-based) shown below. The element we want to retrieve is 55 (note that I am not asking you to write down the code; just include some figures that show the values of first, last and mid indices at each iteration).

```
\[
\begin{array}{c}
  f \rightarrow \begin{array}{c}
    0 \rightarrow 12 \\
    1 \rightarrow 31 \\
    2 \rightarrow 44 \\
    3 \rightarrow 54 \\
    4 \rightarrow 96 \\
    5 \rightarrow 100 \\
    6 \rightarrow 200 \\
  \end{array} \\
\end{array}
\]
```
3. **Code** (20 pts) Overload the assignment operator for the class `SortedType` (i.e., sorted linked list):

```cpp
template<class ItemType>
class SortedType {
public:
    SortedType();
    ~SortedType();
    void MakeEmpty();
    bool IsFull() const;
    int Lengths() const;
    void RetrieveItem(ItemType&, bool&);
    void InsertItem(ItemType);
    void DeleteItem(ItemType);
    void ResetList();
    bool IsLastItem() const;
    bool GetNextItem(ItemType&);
private:
    int length;
    NodeType<ItemType> *ListData;
};
```

```cpp
template <class ItemType>
void SortedType<ItemType>::operator=(SortedType<ItemType>& RHS)
{
    ItemType temp;
    int i;
    length = RHS.length;
    for (i = 0; i <= RHS.length; i++)
        info[i] = RHS.info[i];
}
```

```cpp
current Pos = RHS.current Pos;
```
3. **Code** (20 pts) Overload the assignment operator for the class *SortedType* (i.e., sorted linked list).

```cpp
template<class ItemType>
class SortedType {
public:
    SortedType();
    ~SortedType();
    void MakeEmpty();
    bool IsFull() const;
    int Lengths() const;
    void RetrieveItem(ItemType&, bool&);
    void InsertItem(ItemType);
    void DeleteItem(ItemType);
    void ResetList();
    bool IsLastItem() const;
    bool GetNextItem(ItemType&);
private:
    int length;
    ItemType* first;
};
```

```cpp
template <class ItemType>
void SortedType<ItemType>::operator=(SortedType<ItemType>& RHS)
{
    int i;
    int mL = maxL = RHS.maxL;
    if (maxL == RHS.maxL)
    for (i=0; i<=RHS.length; i++)
        info[i] = RHS.info[i];
    else {
        delete [] info;
        info = new ItemType[RHS.maxL];
        for (i=0; i<=RHS.length; i++)
            info[i] = RHS.info[i];
    }
    length = RHS.length;
}
```
4. **Code** (20 pts) Write a client function that merges two sorted lists using the following specification:

```
MergeLists(SortedType list1, SortedType list2, SortedType& result)
```

*Function*: Merges two sorted lists into one sorted list.
*Precondition*: list1 and list2 have been initialized.
*Postconditions*: result is a sorted list that contains all of the items from list1 and list2 (no duplicates)

```
MergeLists(SortedType l1, SortedType l2, SortedType& r)
{
    ItemType item;
    bool found;

    l1.ResetList();
    l2.ResetList();
    r.MakeEmpty();

    while(!l1.IsLastItem()) {
        l1.GetNextItem(item);
        r.InsertItem(item);
    }

    while(!l2.IsLastItem()) {
        l2.GetNextItem(item);
        l1.RetrieveItem(item, found);
        if(!found)
            r.InsertItem(item);
    }
}
```