AVG: 80
H: 97
L: 58

CS302 Data Structures
Fall 2008 – Dr. George Bebis
Midterm Exam
1:00-2:15 PM

Name: — Solutions —

1. **[40 points – 4pts each]** True/False Questions – To get credit, **you must give brief reasons for each answer!**

T (F) Changes in the formal parameter values of a function would not affect the actual parameter values of the function.

If calling the function by reference

(T) F It would be more efficient to implement Binary Search using arrays than linked lists.

Const time to find the middle of the array while O(N) for Linked Lists

(T) F The value of the postfix expression 5 4 - 3 2 + * is 5

1 5

*

(=5)

T (F) A linked-list-based implementation of *QueueType* has always lower memory requirements than an array-based implementation, assuming that each queue contains N elements.

Suppose pointers take up more memory that data (e.g. short integers)

(T) F Implementing a dictionary of words (e.g., used by a spelling checker) using sorted list would be more efficient than using an unsorted list.

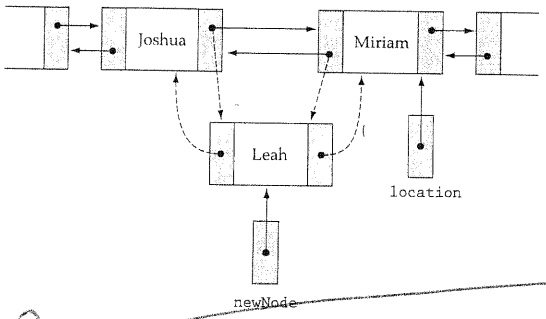
only if we use the array-based implementation (binary search — O(log N))

(if list size very small, linear better that binary)

T F The following code segment allocates memory dynamically for a 10 by 20 array (i.e., 10 rows by 20 columns).

```
arr2D = new int* [20];
for(i=0; i<10; i++)
  arr2D[i] = new int[10];
```

T F The statements shown below, executed in this order, insert a new element correctly in a doubly-linked list.



1. newNode->back = location->back;
2. newNode->next = location
3. location->back = newNode;
4. location->back->next=newNode;

T F The output of the piece of code shown below is:

```
1
4
3

UnsortedType<int> list;
int item;

list.InsertItem(1); list.InsertItem(2);
list.InsertItem(3); list.InsertItem(4);
list.DeleteItem(2);

list.ResetList();
while(!list.IsLastItem()) {
  list.GetNextItem(item);
  cout << item << endl;
}
```

T F When using templates, the implementation of the class cannot be compiled separately from the application of the class.

T F The running time of the code segment shown below is O(NlogN)

```
sum = 0;
for(i=1; i<=n; i=i+2)
 for(j=n; j>=1; j=j/2)
   sum++;
```

2. **[20 points – 5pts each]** Short answer questions

(a) In general, a copy of another object can be created using either the copy constructor or the assignment operator. What is the **main** difference between them?

Copy constructor: creates a new object

assignment: operates on an existing object

(b) Consider the array-based implementation of queues discussed in class; what are the conditions for an empty or full queue?

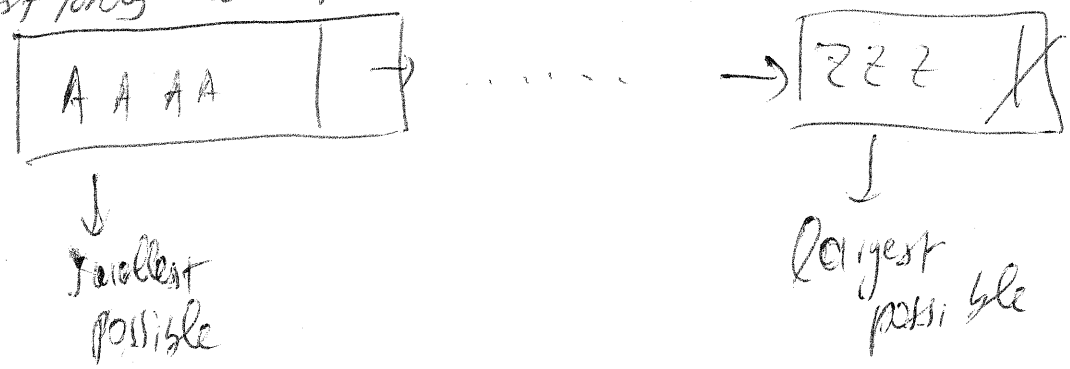
empty: front == rear

full: rear + 1 == front

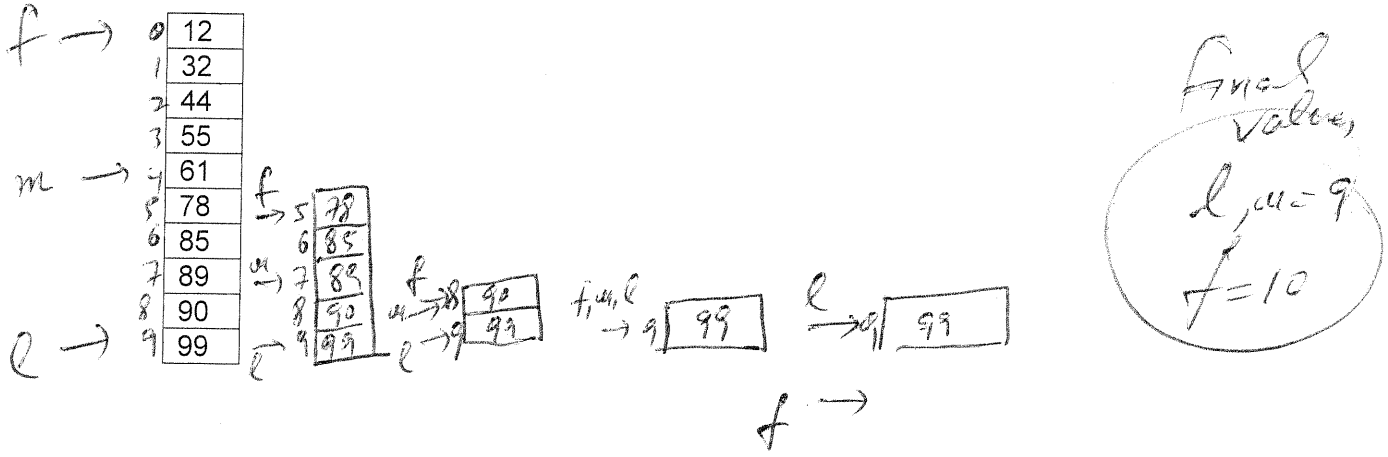
(c) We have discussed in class the idea of using "dummy" nodes (i.e., headers and trailers) to simplify list processing by eliminating some "special cases". What are these special cases and under what conditions could we use "dummy" nodes?

insert beginning/end/empty list

delete first/last/only element



(d) Demonstrate the binary search algorithm on the array shown below. The element to be retrieved is 101. Show clearly what would be the values of **first**, **last**, and **midpoint** at each iteration. What would be the values of **first**, **last**, and **midpoint** upon the execution of the while-loop?



3. [10 points] Consider *Algorithm 1* that takes $3N^2+2N+1$ time to solve a problem and *Algorithm 2* that takes $10N+1$ time to solve the same problem.

a. What are the big-O requirements of each algorithm? Which algorithm is more efficient by big-O standards?

A1: $O(N^2)$

A2: $O(N) \Rightarrow$ more efficient

b. Under what conditions, if any, would the "less" efficient algorithm execute faster than the "more" efficient algorithm?

$3N^2+2N+1 \overset{?}{<} 10N+1 \Rightarrow 3N^2-8N<0$

$\Rightarrow N(3N-8)<0 \Rightarrow 3N-8<0 \Rightarrow N<8/3 \approx 2.66$

$N \leq 2$

4. (a) [10 points] Write a client function **MoveSmallestItem(StackType<ItemType>& s)** that finds the smallest **item** on stack **s** and moves it to the top of the stack. You can assume that the stack contains no duplicates. You would need to maintain the ordering for all other elements.

```
void MoveSmallestItem (StackType<ItemType>& S)
{ ItemType item, smallest;
  StackType<ItemType> tmpS;
if (!S.IsEmpty()) {
    S.Pop(smallest); tmpS.push(smallest);} } O(1)
                                  else return;
while (!S.IsEmpty()) {
    S.Pop(item); tmpS.push(item);
    if (item < smallest)                    } O(N)
        smallest = item;
}
while(!tmpS.IsEmpty()) {                    } O(N)
    tmpS.pop(item);
    if (item != smallest)
}       S.Push(item);
S.Push(smallest);  } O(1)
}
```

(b) [5 points] Using big-O notation, carefully analyze the running time requirements of your solution.

See above — O(N)

5. **(a) [10 points]** Write a client function **DeleteDuplicates(UnsortedType<ItemType>& list)** that deletes all the duplicates in an unsorted list. For example, if L contains 1 5 2 5 2 3 1 2 4, then, after deleting all the duplicates, L should contain 1 5 2 3 4 (note: the final order of the elements in L is unimportant since the list is unsorted).

```
void DeleteDuplicates(UnsortedType<ItemType>& list)
{
  ItemType item;
  bool found;
  UnsortedType<ItemType> tmp;

  list.ResetList();
  while (!list.IsLastItem()) {                    N times
    list.GetNextItem(item);                       O(1)
    tmp.RetrieveItem(item, found);                O(N)
    if (!found)
      tmp.InsertItem(item)                        O(1)
  }                                                          } O(N^2)
  list = tmp;                                     O(N)
}
```

(or)

```
{
  list.MakeEmpty();
  tmp.ResetList();
  while (!tmp.IsLastItem()) {
    tmp.GetNextItem(item);
    list.InsertItem(item);
  }
}
```

(b) [5 points] Using big-O notation, carefully analyze the running time requirements of your solution.

Total: $O(N^2)$