Sample

302

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.

linear search is faster for small sizes (e.g., $N \leq 20$)

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

~~call~~ 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.

hide the implementation details of the class from the user.

(1.4) (**T**) **F** The running time of *RetrieveItem* (sorted lists) is *O(N)*

if linear search is used (O(lgn) 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.

use big-O (asymptotic analysis)

machine dependant !!

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

3 times more (r, g, b)

(1.7) T (F) An $O(logN)$ algorithm is slower than an $O(N)$ algorithm.

$\lg N < N$

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

Stack is most appropriate

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

pass by value would lead to infinite recursion!

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

```
sum = 0;                          → O(1)
for(i=0; i<N; i++) {              → loop is executed N times:
  if(i > j)
    sum = sum + 1;                → O(1)
  else {
    for(k=0; k<N; k++)            → O(N)
      sum = sum - 1;
  }
}
```

$N \times O(N) = O(N^2)$

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.

```
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++;
}
```

best case O(1)
worst case O(N)

best case O(1)
worst case O(N)

O(N) overall

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

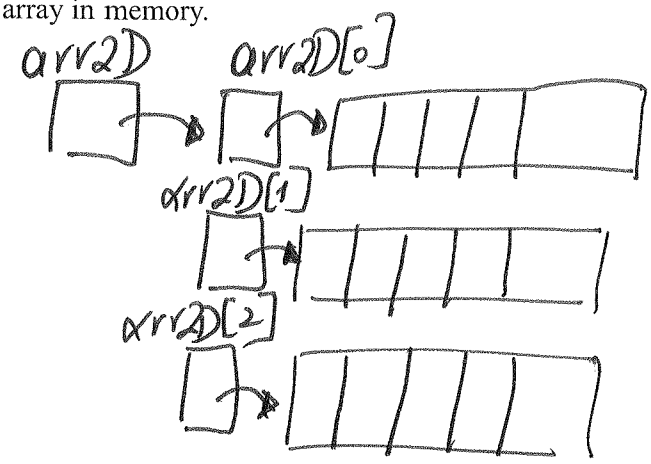
Static:
(i) fixed memory
(ii) stored as 1D array
(iii) stored in contiguous memory

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.

```
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 NxN 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

```
iterate through rows (N/s)
    iterate through columns (N/s)
        shrinked[i][j] = orig[i*s][j*s];
```

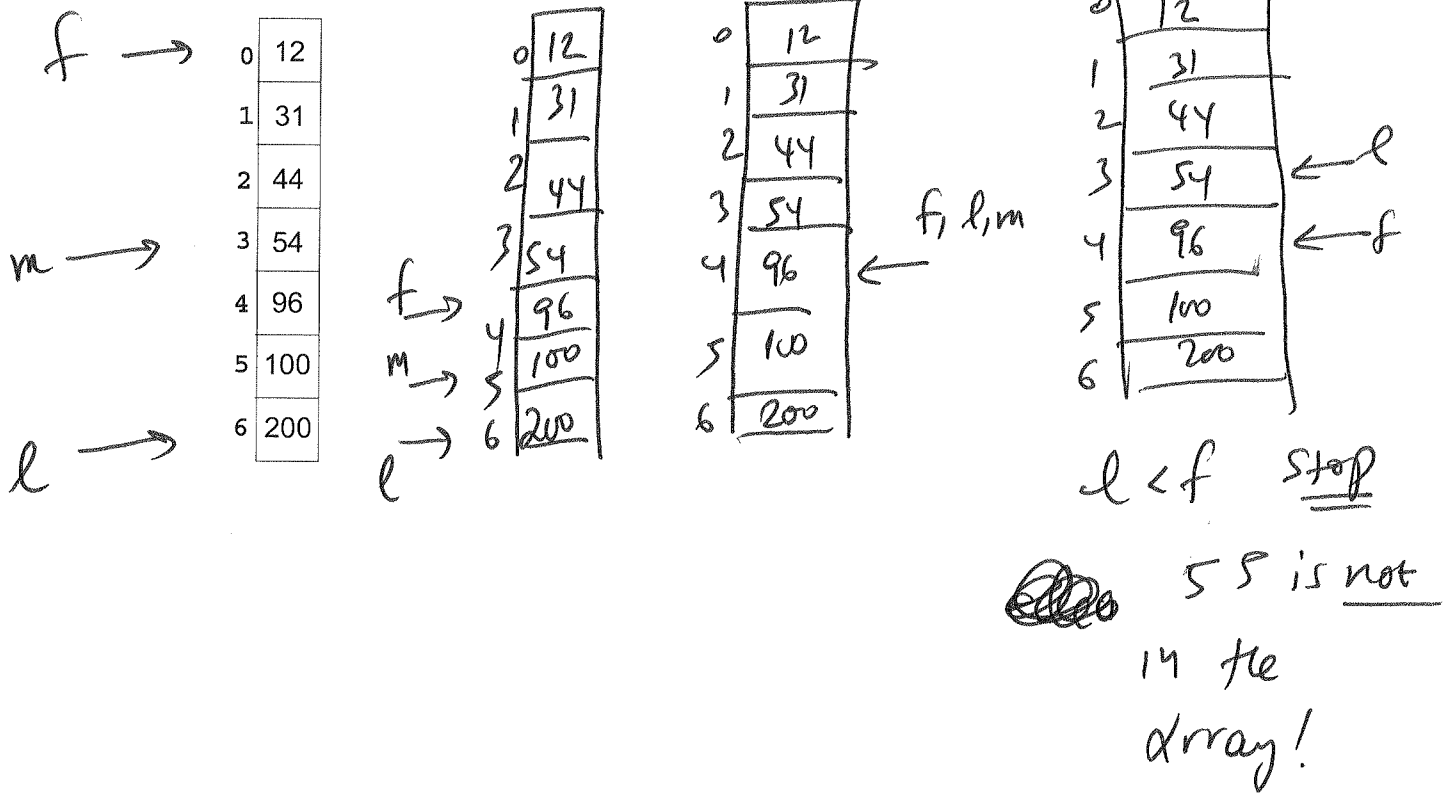
$N/s * N/s = O(N^2)$

(2.5) What are the differences between "call by value" and "call by reference" ?

function receives a copy of the actual parameter

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).



3. **Code** (20 pts) Overload the assignment operator for the class *SortedType* (i.e., ~~sorted linked list~~ array-based).

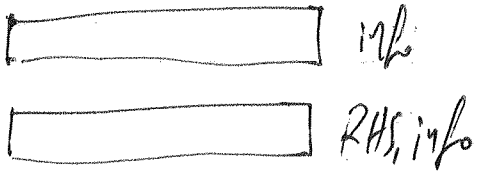
```
template<class ItemType>
class SortedType {
 public:
   SortedType();
   ~SortedType();
   void MakeEmpty();
   bool IsFull() const;
   int LengthIs() const;
   void RetrieveItem(ItemType&, bool&);
   void InsertItem(ItemType);
   void DeleteItem(ItemType);
   void ResetList();
   bool IsLastItem()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

(array-allocated statically)

ItemType info[MAX_ITEMS];
int currentPos;

```
template <class ItemType>
void SortedType<ItemType>::operator=(SortedType<ItemType>& RHS)
{
 ItemType item;
 int i;

 length = RHS.length;
 for(i=0; i<=RHS.length; i++)
   info[i] = RHS.info[i];
}
```



currentPos = RHS.currentPos;

OR

array-based

3. **Code** (20 pts) Overload the assignment operator for the class *SortedType* (i.e., ~~sorted linked~~ list).

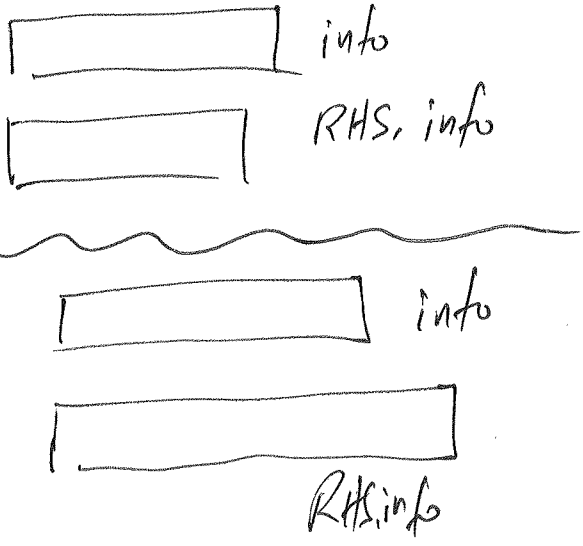
```
template<class ItemType>
class SortedType {
 public:
   SortedType();
   ~SortedType();
   void MakeEmpty();
   bool IsFull() const;
   int LengthIs() const;
   void RetrieveItem(ItemType&, bool&);
   void InsertItem(ItemType);
   void DeleteItem(ItemType);
   void ResetList();
   bool IsLastItem()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

int maxL;
ItemType *info;
int currentPos;
} array allocated dynamically

```
template <class ItemType>
void SortedType<ItemType>::operator=(SortedType<ItemType>& RHS)
{
 int i;

 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;
}
```



currentPos = RHS.currentPos;

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);
    }
}
```

have done in class!