UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Science
Computer Science Division

CS61B Fall 96
J.Canny

CS 61B: Midterm Exam II

This is an open-book exam worth 100 points. There are SEVEN questions and TWELVE pages in the booklet. Write all your answers in this booklet. You have 120 minutes, but to be sure to finish on time, we suggest you budget N minutes for an N-point question. Do not let yourself get stuck on one question. Make sure you start every one. The easiest points are at the beginning of the questions. Wait for permission to start before opening the booklet. Good Luck!

Very important: Fill in this information now!

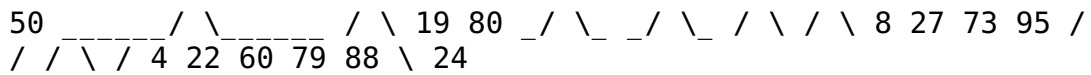
Your name: _____ Login: _____

Section number: _____ Lab TA: _____

For grading purposes only.

Problem #	Possible	Score
1	10	
2	15	
3	25	
4	10	
5	5	
6	15	
7	20	
Total	100	

1. (10 points) The three below is a binary search tree (but not an AVL tree). Show the result of doing Insert(20), followed by Delete(50).



Draw the tree after Insert(20) here:

Draw it after Delete(50) here:

2. (15 points) The following are the contents of an array A, going left-to-right from A[0] to A[12]:

A = {98, 60, 20, 45, 3, 16, 18, 23, 17, 1, 2, 8, 9}

This array represents a heap under the usual arrangement. Draw the heap as a binary tree below.

Now draw the heap after an insert(35) operation:

Now draw the heap after a Getmax() operation:

3. (25 points) (Please read parts (a)-(f) of this question carefully and answer all of them)

Its your first week working at "Don Corleons's Data Structures with Class". An important customer needs a reliable implementation of stacks and queues. You are eager to please in your first assignment, and also aware that two other employees who got behind deadline have mysteriously disappeared recently.

Rather than writing a stack and queue implementation from scratch, you decide to use a priority queue that you implemented for an old course project. You realize that by assigning the right priorities to data items when they are inserted, you can make data come out in either a LIFO or FIFO order.

Your priority queue class has the following prototype:

```
class Pqueue {
```

```
public:
void insert(int priority, dataType D); // Insert D with priority
dataType getmax();       // Get the max element and remove it
int empty();             // Return 1 if Pqueue is empty, 0 otherwise
Pqueue();                // Create an empty priority queue
};
```

and the insert definition says:

```
void Pqueue::insert(int priority, dataType D) {
// pre: priority is any integer, positive or negative...
....
```

Both your stack and your queue should have member functions

```
void insert(dataType D);  // Insert data into the stack/queue
dataType getnext();   // Get & remove the next element (in LIFO/FIFO order
int empty();        // Return 1 if stack/queue is empty, 0
otherwise
```

(a) (5 points) Write the class prototype for the STACK. Include private data fields, and use a comment to say what they are.

(b) (2 points) Write the class prototype for the QUEUE. Include private data fields, and use a comment to say what they are.

(c) (5 points) Write the getnext() member function for the STACK. Briefly describe how you would change it if you were to implement a queue.

(d) (5 points) Write the insert(dataType D) member function for the STACK. Briefly describe how you would change it if you were to implement a queue.

(e) (3 points) Assume that the Pqueue class is implemented with a HEAP. Give the tightest big-O bounds you can for getnext() for your QUEUE class, assuming N elements in the queue. Briefly explain your answer.

(f) (5 points) Assume that the Pqueue class is implemented with a HEAP. Give the tightest big-O bounds you can for insert() for your QUEUE class, assuming N elements in the queue. Briefly explain your answer.

4. (10 points) The Quirk are a feisty, subatomic race of people who run the Universe below one Angstrom unit. Some people argue that Quirks don't exist, because they have no mass, and have extremely short lives. But inestimable numbers of Quirks live, work, create art, make war, love and die in a typical sand grain in a single pico-second, and they resent

that humans (especially Werner Heisenberg) ignore them.

Quirks come in three types, A, B, and C. Quirks react instantly when they encounter a Quirk of another type:

An A Quirk and a B Quirk react to produce a C
a B Quirk and a C Quirk react to produce an A
a C Quirk and an A Quirk react to produce a B

In an effort to enhance public understanding of Quirks, you are commissioned to write a simulator for Quirk behavior. You decide to use three different classes, one of each Quirk, and to overload the + operator to represent a Quirk-Quirk reaction.

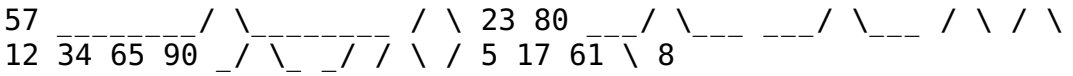
(a) (3 points) Write a class definition for *one* type of Quirk.

(b) (5 points) Write code for *one* type of Quirk-Quirk reaction, e.g. overload the + operator so that B + C evaluates to an A Quirk. Your code for this one reaction should allow both orders of the arguments, i.e. B + C and C + B.

(c) (2 points) Are the results of each Quirk addition known at compile-time? Explain briefly.

5. (5 points) BRIEFLY describe how to use a Data Display Debugger like DDD to determine whether a list structure is circular.

6. (15 points) The tree below is almost an AVL tree. Find the out-of-balance node N, and circle it. Choose an edge in the tree to rotate, to restore the tree's balance. The edge will include N and one of its children. Highlight this edge by shading it.



Now draw the tree after the rotation:

7. (20 points) Listed below are three sorting functions, name Alice, Bob and Chuck. Each function has a helper function just below it. For this question you have to understand each of these three functions and answer the questions on the next page. You should find it useful to read those questions before studying the code. Try simulating each program!! It's probably easier than trying to understand them by reading alone. If they look strange, try to remember the recursive approach to problem solving, or how you programmed in CS61A...

All sorting functions satisfy the specification:
// pre: A is an array of integers. left is the index of the first element
// in the subarray to be sorted, and right is index of the

```
last element.
// post: A contains the same elements and
// A[left] <= A[left+1] <= A[left+2]... <= A[right]


void Alice(int * A, int left, int right) {
  if (right <= left) return;
  Alice(A, left, right - 1);
  Alice_helper(A, left, right - 1, A[right];}

void Alice_helper(int * A, int left, int right, int elt) {
  if (right >= left)
    if (elt <A[right]) {
      A[right + 1] = A[right];
      Alice_helper(A, left, right - 1, elt);
      return;}
  A[right + 1] = elt;
  return;}

void Bob(int * A, int left, int right) {
  if (left >= right) return;
  int elt = A[right];
  int index = Bob_helper(A, left, right, elt);
  Bob(A, left, index-1);
  Bob(A, index, right);
  return;}

int Bob_helper(int * A, int left, int right, int elt) {
  if (left >= right) return left;
  if (A[left]  <elt) return Bob_helper(A, left + 1, right, elt);
  if (A[right] > elt) return Bob_helper(A, left, right - 1, elt);
  swap(A[left], A[right]);    // This swaps A[left] and A[right]
  return Bob_helper(A, left + 1, right - 1, elt);}

void Chuck(int * A, int left, int right) {
  of (left >= right) return;
  int index = Chuck_helper(A, left, right);
  swap(A[left], A[index]);  // This swaps A[left] and A[index]
  Chuck(A, left + 1, right);
  return;}

int Chuck_helper(int * A, int left, int right) {
  if (left >= right) return left;
  int index = Chuck_helper(A, left + 1, right);
  if (A[left]  <A[index])
    return left;
  else
    return index;}
```

(a) (6 points) Each of these sorts is a familiar sort, i.e. its one of: Insertion Sort, Selection Sort, Quicksort, Heapsort, Bubblesort, Mergesort or Radix Sort.
Write down the familiar sort name for each program:

Alice_____

Bob_____

Chuck_____

(b) (6 points) What is its BEST-CASE runnning time for each sort?

Alice_____

Bob_____

Chuck_____

(c) (6 points) Which of the programsa above are STABLE? Say YES if it is stable, NO otherwise:

Alice_____

Bob_____

Chuck_____

(d) (2 points) One of the programs above that is NOT stable can be made stable by changing a single line of the program. Give the name of the program, and give the changed program line:

Program Name_____

```
Changed Line_____________________________________
```
