ECE 264 Final Exam

8-10AM, May 5, 2011

I certify that I will not receive nor provide aid to any other student for this exam.

Signature:

You must sign here. Otherwise you will receive a **1-point** *penalty.*

Read the questions carefully.

Please write legibly. Your exam is not graded if your writing is hard to read.

This exam is printed **double sided**. Please read the questions carefully. Two common mistakes are answering the wrong question and failing to answer all questions.

This is an *open-book, open-note* exam. You can use any book or note or program printouts.
Please turn off your cellular phone and iPod. No electronic device is allowed.

Please report the points that are currently not counted in Blackboard. You may receive 0.5 bonus points for filling the on-line course evaluation (already closed) and returning a survey (sent to you by email).
You may receive 1 point for each visit to the instructor's office; this is part of the attendance points. You may receive up to 10 attendance points.

Contents

This page is blank. You can write answers here.

1 Binary Search Tree (7 points)

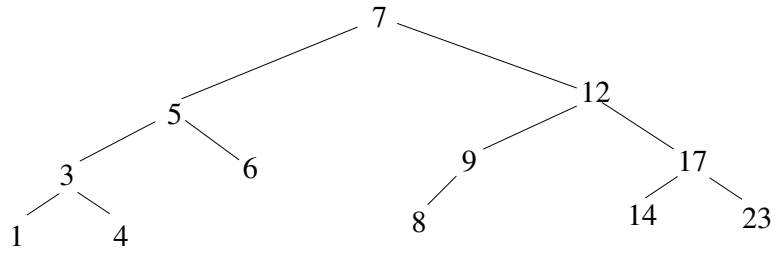
1.1 Construction (1 point)

Consider a binary search tree with one integer value stored at each node. **Draw** the tree after the following numbers are inserted in the following order.

```
35, 58, 47, 36, 21, 32, 89, 34, 23, 10
```

1.2 Traversal (2 points)

Tree traversals visit the nodes of a tree in varying order: pre-order, post-order, or in-order. The code for these traversals are given in the code listing at the end of this question.



For the above tree, what is the sequence of **pre-order** (visiting parent before children) traversal? (1 point)

What is the sequence of **post-order** (visiting children before parent) traversal? (1 point)

1.3 Insertion (2 points)

Write `Tree_insert` using recursion.
You must use **recursion**. You will lose at least one point if you do not use recursion.

1.4 Search (2 points)

Write `Tree_search` using recursion.
You must use **recursion**. You will lose at least one point if you do not use recursion.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct treenode {
  int value;
  struct treenode * left;
  struct treenode * right;
} TNode;

TNode * TNode_construct(int v)
{
  TNode * tn;
  tn = malloc(sizeof(TNode));
  tn -> value = v;
  tn -> left = NULL;
  tn -> right = NULL;
  return tn;
}

void TNode_destruct(TNode * tn)
{
  free(tn);
}

void Tree_delete(TNode * top)
{
  if (top == NULL) { return; }
  Tree_delete(top -> left);
  Tree_delete(top -> right);
  TNode_destruct(top);
}

/*
   Insert value v into the tree.  If this value is already in the
   tree, do nothing.
   Return the top node of the new tree.  This function must be recursive. */
```

```
TNode * Tree_insert(TNode * top, int v)
{
  if (top == NULL) {



  }
  if ((top -> value) == v) {



  }
  if ((top -> value) > v) {
```

```
  }
  else {
```

```
  }
```

```
  return
}

void Tree_inOrder(TNode * top)
{
  if (top == NULL) { return; }
  Tree_inOrder(top -> left);
  printf("%d\n", top -> value);
  Tree_inOrder(top -> right);
}

void Tree_preOrder(TNode * top)
{
  if (top == NULL) { return; }
  printf("%d\n", top -> value);
  Tree_preOrder(top -> left);
  Tree_preOrder(top -> right);
}

void Tree_postOrder(TNode * top)
{
  if (top == NULL) { return; }
  Tree_postOrder(top -> left);
  Tree_postOrder(top -> right);
  printf("%d\n", top -> value);
}

/* return 1 if any node in the tree contains value v */
/* return 0 if no node in the tree contains value v */
/* use recursion */
int Tree_search(TNode * top, int v)
{









}
```

```
int main(int argc, char * argv[])
{
  const int NUMELEM = 10;
  int data[] = {35, 58, 47, 36, 21, 32, 89, 34, 23, 10};
  TNode * root = NULL;
  int index;
  for (index = 0; index < NUMELEM; index ++)
    {
      root = Tree_insert(root, data[index]);
    }
  Tree_inOrder(root); printf("\n\n");
  Tree_preOrder(root); printf("\n\n");
  Tree_postOrder(root); printf("\n\n");
  printf("search(13) = %d\n", Tree_search(root, 13));
  printf("search(58) = %d\n", Tree_search(root, 58));
  Tree_delete(root);
  return 0;
}
```

2 Sorting (8 points)

Sorting is an important algorithms in programming and is a basic building block for many advanced algorithms. Below we discuss two sorting algorithms: insertion and selection sort.

Note: For both of the below questions, fill in the missing code in the skeleton source listing below.

2.1 Insertion Sort (4 points)

Insertion sort arranges items (such as integer numbers) by inserting them, one by one, into a new and sorted list. Inserting a value proceeds by iterating through a list until the appropriate position is found; a position where the element just before is *smaller*, and the one just after is *larger*.

Sorting an array using insertion sort can be done by iterating through the array and treating everything after the current position as unsorted, and everything before as sorted.

Fill in the missing code in the function `insertion_sort` below.

2.2 Selection Sort (4 points)

Selection sort turns an unsorted array into a sorted array by repeatedly finding the smallest value in the unsorted part of the array and putting it at the end of the sorted part. In the beginning, the unsorted part is the whole array and there is no sorted part; in the end, the algorithm has sorted the whole array and no unsorted part remains.

Fill in the missing code in the function `selection_sort` below.

2.3 Skeleton Code

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void randomize() {
    srand(time(NULL));
}

int getRandom(int min, int max)  {
    return (max - min) * ((double) rand() / (double) RAND_MAX) + min;
}

void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```

```
/* Insertion sort for an integer array.
 *
 * HINT: In the for loop below, everything *before* i (0 to i-1) is
 * sorted, everything *after* (i to num - 1) is unsorted. For every
 * index i, move left (until index 0) until the preceding value in the
 * array is larger.
 *
 * NOTE: You are NOT allowed to allocate new memory (even if you
 * deallocate it).  Sorting should be done in-place.
 */
void insertion_sort(int *array, int num)
{
    // ==== FILL IN CODE BELOW ====
    int i;
    for (i = 0; i < num; i++) {









    }
}
```

```
/* Selection sort for an integer array.
 *
 * HINT: In the for loop below, you will want to find the element in
 * the rest of the list (i to num - 1) that is the smallest.  That is
 * the one that should be swapped into index i.
 *
 * NOTE: You are NOT allowed to allocate new memory (even if you
 * deallocate it).  Sorting should be done in-place.
 */
void selection_sort(int *array, int num)
{
    // ==== FILL IN CODE BELOW ====
    int i;
    for (i = 0; i < num; i++) {









    }
}
```

```
void print_array(int *array, int num)
{
    int i;
    for (i = 0; i < num; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");
}

int main(int argc, char *argv[])
{
    // Check for arguments
    if (argc < 2) {
        printf("need an argument!\n");
        return -1;
    }

    // Retrieve arguments and allocate memory
    int num = strtol(argv[1], NULL, 10);
    int *array = malloc(sizeof(int) * num);

    // Fill in the array with random values
    int i;
    randomize();
    for (i = 0; i < num; i++) {
        array[i] = getRandom(0, 1000);
    }

    // Sort and print
    selection_sort(array, num);
    insertion_sort(array, num);
    print_array(array, num);

    // Free memory
    free(array);

    return 0;
}
```

3 Permutation (6 points)

Write a function that prints all possible and distinct (i.e., unique) permutations of an array.

Hint: The algorithm for creating integer partitions may be useful as inspiration. You may create additional functions called by `permute`.

```
#include <stdio.h>
```

```
void permute(char * array, int size)
{
  /*
    print all possible and distinct permutations of the array
    size is the number of elements in the array
    You may create additional functions called by permute.
    Assume the elements in the array are distinct.
   */
```

```
}

int main(int argc, char * argv[])
{
  int numElem = 3;
  char array[] = {'E', 'C', 'P'};
  permute(array, numElem);
  return 0;
}
/*
  sample output: your program has to print these lines, but the order may be
  different.
  E C P
  E P C
  C E P
  C P E
  P C E
  P E C
*/
```