

# **CSCI 2300 – Data Structures and Algorithms**

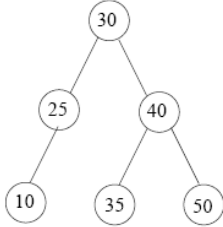
## **Lab 8 – Exam 2 Review**

### **Overview**

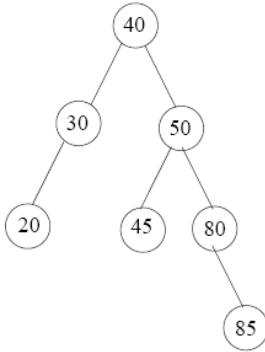
- Exam 2 will be held in class on Friday, April 6.
- There will be no makeup exam if you miss this exam.
- The exam is closed everything – closed book, closed notes, no crib sheets, no calculators, and no aids of any kind.
- Any formulas that you may need will be given to you.
- Coverage: Chapters 4 to 7.

This lab consists of the first seven problems taken from Exam 2 of Spring 2006.

1. (12 points) Starting from the AVL tree below, show the AVL tree that results after each of the following operations: (a) insert 20, (b) insert 5, (c) remove 25. Indicate the rotations you are performing at each stage.



2. (10 points) Starting from the splay tree below, show the splay tree that results after each of the following operations: (a) find 85, (b) remove 40. Indicate the splaying operations you are performing at each stage.



3. (a) (5 points) Consider the following hash function.

```

int
Hash(const string& key, int tbl_size)
{
    unsigned int value = 0;
    for ( int i=0; i<key.length(); i++ )
        value = value + 31*key[i];
    return value % tbl_size;
}
  
```

Is this a good hash function for `tbl_size = 100`? Why or why not?

- (b) (8 points) Suppose you are a Shakespeare scholar, and wish to print the  $k$  most frequent words, in decreasing order of frequency, in each of Shakespeare's books. Assume the number of distinct words in each book is  $n$  and that  $k$  is much smaller than  $n$ . Describe your algorithm for this task and the data structure(s) you would use. Justify your answer, using "O" notation, on the basis of efficiency.

4. (12 points) Consider a priority queue represented as a min binary heap, which is implemented as an array. Assume the heap elements are stored at array locations starting at 1, and that functions `Percolate_Down` and `Percolate_Up` exist with prototypes:

```
void Percolate_Down( ElementType heap[], int size, int i);
void Percolate_Up( ElementType heap[], int size, int i);
```

where `heap` is the array of elements, and `size` is the current number of elements. Assume that `i` is correctly given to you, i.e., assume  $1 \leq i \leq \text{size}$ .

Write a function to change the element stored at location `i` of the array by the value `delta`, while maintaining the binary heap. Assume `ElementType` represents a numeric type with the usual addition, comparison, and assignment operators. Note that `delta` may increase or decrease the value of the original element. Start from the following prototype

```
void ChangePriority( ElementType heap[], int size, int i, ElementType delta )
{
```

5. (10 points)

- (a) Assume you are given the following initial array, and you are to convert it to a min binary heap using the `BuildHeap` procedure.

0	1	2	3	4	5	6	7	8
	8	5	14	2	19	25	7	4

Write the heap that results (as an array) after the build heap procedure is run. Assume the heap array index begins at 1. (You may work out the problem by drawing the binary heap as a tree, but you will lose points if the final answer is not written as an array.)

- (b) Write the heap that results (as an array) after an `insert 1` operation is performed.

6. (a) (16 points) The following code is the original basic quick sort function given in class. It uses the value at location Left as the pivot. Modify this code so that the value stored at location Right is used as the pivot.

```
template <class T>
void
B_Q_Sort( T A[ ], int Left, int Right )
{
    if ( Left >= Right ) return;

    T Pivot = A[ Left ];
    unsigned int i = Left, j = Right+1;

    for( ; ; )    {
        while( A[ ++i ] < Pivot && i < Right );
        while( A[ --j ] > Pivot );
        if( i < j )
            Swap( A[ i ], A[ j ] );
        else
            break;
    }
    Swap( A[ j ], A[ Left ] ); // Place the pivot in its final position

    B_Q_Sort( A, Left, j - 1 );
    B_Q_Sort( A, j + 1, Right );
}
```

- (b) (4 points) The running time of Quicksort depends both on the input data and the partition rule for how the pivot element is selected. Suppose we always pick the pivot element to be from the last position of the (sub)array, as in the above question. What is the running time of this version of basic Quicksort on a **sorted array**? Write the running time in “O” notation in terms of  $n$ , where  $n$  is the value of  $\text{Right}-\text{Left}+1$  in the first call, and justify your answer.

7. (8 points) The *mode* of a set of numbers is the number that occurs most frequently in the set. For example, the set  $\{4, 6, 2, 4, 3, 1, 4, 2\}$  has a mode of 4.

Describe an efficient algorithm to compute the mode of a set of  $n$  numbers. Write the running time of your algorithm using  $O$  notation and justify it.