Data Structures and Algorithms (EECE 330)

Quiz #2 Solutions

1. Indicate the performance of the *bubble sort* algorithm as a function of *n* (size of input data) in the worst case scenario. (3 points)
$O(n^2)$

2. List three sorting algorithms that are based on the *divide-and-conquer* principle and briefly (one sentence or two) explain how they work so as to indicate the differences between them. (9 points)
Shell sort: it divides the array of size *n* into *n*/2 sub-arrays, sorts each one, and then combines them. It then divides the array into *n*/4 sub-arrays, sorts each one, and then combines them. It repeats this pattern until it divides the array into 2 sub-arrays and afterwards, it sorts the entire array.

Quick sort: It is based on finding a pivot (middle element of the array) and placing it in its final. Repeat this for the arrays to the left and right of the pivot until all elements are chosen as pivots.

Merge sort: Divide the array into two halves and then each half into two halves and son on until we end up with sub-arrays of two elements. Sort each sub-array and then merge into sub-arrays of double the size. Repeat until we end up with the entire array.

3. Out of all the *divide-and-conquer* sorting algorithms, which one requires twice the amount of space? (3 points)
Merge Sort

4. Write the pseudo-code for the *Quicksort* algorithm. (10 points)

```
Define LA as an array of pointers and initialize to NULLs
Add a pointer to the input array A[] to LA
Define l, r, sl, sr as integers
Do
    Pivot = middle element
    Swap pivot with last element
    Assign l to the leftmost element and r to the rightmost element.
    While l is less than r
        While A[l] is less than pivot
            Move l right
        End while
        While A[r] is greater than pivot
            Move r left
        End while
        Swap A[l] and A[r]
    End while
    Move pivot to position l
    sl= size of array to the left of pivot
    sr= size of array to the right of pivot
    If sl > 1
                Add a pointer to the left array to LA
    End IF
    If sr > 1
        Add a pointer to the right array to LA
    End IF
    If sl < 2 and sr < 2 //the whole array was processed
        Remove the pointer of this array from LA
    End If
While there are array pointers in LA
```

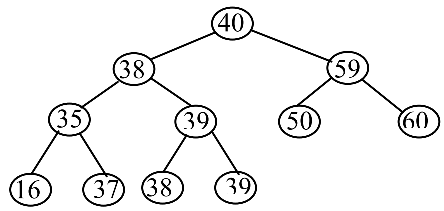
5. Write the pseudo-code for the algorithm that deletes a node from a BST (6 points)

define ND as the pointer to the node to be deleted
define RC as the right child of ND
call *deletemin* and pass it RC
define temp as the node pointer that was returned by *deletemin*
replace the value in ND by the value in temp
delete temp

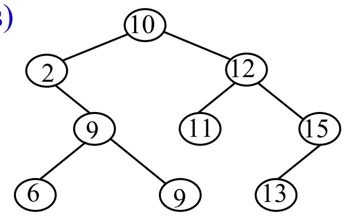
deletemin:

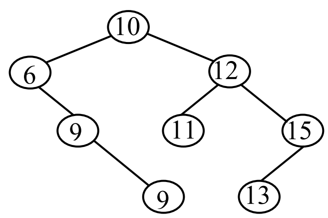
```
while the current node has a left child
    current node = left child
End while
define temp as the pointer to current node
reassign the link from the parent of temp to temp's right child
return temp
```

6. Draw the BST that results from inserting the following values in the shown order 40, 38, 35, 16, 37, 59, 50, 39, 60, 39, 38. (8 points)



7. Draw the BST tree that results after deleting the record whose key is equal to 2 in the BST below. (7 points)





8. Given a regular Hash table of size `L` (every slot represents a possible home position: no buckets and no linked lists), and given `N` records to be placed in the table (`N` ≤ `L`), write the C++ code for a *hash function* that would work (we don't care if it is not optimal). (5 points)

```
h(k)=k%L
```

9. Suppose we have a hash table (`HT`) in which we have inserted many records and from which we have deleted many records. Now, we want to insert record R_i whose key is K_i and the probe sequence is p_0, p_1, p_2, p_3, p_4, …, where p_0, p_1, p_2, … correspond to positions in `HT`. What is the condition that tells us where R_i will be inserted in `HT`? (4 points)

When we encounter an `EMPTY` record or a `TOMBSTONE` record

10. Suppose `k` is the record's key, `M` is the size of the Hash table, `h` is the hash function, `p` is the probe function, and `i` is the probe sequence index. Write the C++ expression that is used to determine the Hash Table slot (`pos`). (8 points)

```
pos=(h(k)+p(k,i))%M
```

11. What do we mean by primary clustering in hashing? (4 points)

When the slots in the hash table do not have the same probability for being filled

12. Suppose we use quadratic probing to search for records. We have two records with keys K_1 and K_2. The output of the hash function for K_1 is 15 and for K_2 is 19. Suppose we have a simple Hash table (no buckets and no linked lists) of size 50 and it takes 8 searches for each record to be found, write the two probe sequences for K_1 and K_2. (10 points)

Two possible solutions:

K_1: [15,16,20,29,45,20,6,5]
K_2: [19,20,24,33,49,24,10,9]

or

K_1: [15,16,19,24,31,40,1,14]
K_2: [19,20,23,28,35,44,5,18]

13. What 3 factors distinguish indexing from hashing? (6 points)

Ability to process large files
Ability to handle multiple keys
Ability to handle range Queries

14. In a graduate class at AUB, there are 16 students whose *secondary keys* (last names) and *primary keys* (AUB IDs) are shown in the table below. The values in parentheses inside the table cells indicate the order in which the corresponding student records were inserted in the database.

Last Name ↓	AUB IDs					
Balkourkian	200132141 (4)	200099812 (15)	200188881 (16)			
Haidar	200122001 (7)	200167112 (14)	200056112 (2)	200055566 (10)	200166220 (11)	200144411 (6)
Nasreddine	200132114 (13)	200111103 (8)				
Haddad	200178112 (9)	200189110 (1)	200056661 (3)	200160001 (12)		
Safa	200167101 (5)					

Design an *inverted List* for storing the secondary and primary keys. (12 points)

Secondary Keys

Haddad	0
Haidar	1
Balkourkian	3
Safa	4
Nasreddine	7

Primary Keys

200189110	2
200056112	5
200056661	8
200132141	14
200167101	
200144411	6
200122001	9
200111103	12
200178112	11
200055566	10
200166220	13
200160001	
200132114	
200167112	
200099812	15
200188881	

15. What is a 2-3 tree and what are two of its properties? (5 points)

It is a tree where a node stores up to two data values (records) and has up to three children

Properties:

1. It is always balanced (leaves are at the same level)
2. It grows upward.