Exam 2 (Sample)

MIDTERM EXAM 2 (SAMPLE)

Note: The actual Midterm 2 will contain exactly 5 questions. In this sample exam the number of questions may differ. (*You must justify all your answers to recieve full credit. The questions that are written on paper should be photographed and uploaded as JPEG or PDF. The C++ programming question should be submitted as the C++ source file in a separate folder.*)

Question 1:

(*1.A. Given a list/stack/queue algorithm pseudocode, find its time complexity.*)

Consider the following List ADT (Abstract Data Type):

void L.add(E item)	// add *item* to the end of the list *L.*
void L.add(int pos, E item)	// add *item* at position *pos*; shift other items forward.
boolean L.contains(E item)	// return `true` iff the item is in the list *L*
int L.size()	// return the number of items in the list *L*
boolean L.isEmpty()	// return `true` iff the list is empty
E L.get(int pos)	// return the item at position pos in the List
E L.remove(int pos)	// remove and return the item at position pos in the List

Consider the following pseudocode – it erases all the odd elements and reverses the remaining list:

```
function ModifyList(L: List[int]):
pos := 0
while pos < L.size():
    if L.get(pos) % 2 == 1:
        L.remove(pos)
    else:
        pos := pos+1
n := L.size()
for i from n − 1 to 0 by -1:
    item := L.get(i)
    L.add(item)
repeat n times:
    L.remove(0)
```

What is the worst-case time complexity of the three loops (**while**, **for** and **repeat**) in the function *ModifyList(L)*. Express the complexity in terms of n (the size of the original list L)? Consider two cases:

(A) Assume that the List ADT is implemented as a doubly linked list with the pointers to the first and the last element and an additional number `size` telling the current number of items – the number of nodes in this linked list.

(B) Assume that the List ADT is implemented as an array (assume that the array has enough space for any possible input) and an additional number `size` telling the current number of items (namely, only the array elements from 0 to `size - 1` contain actual values, everything else in the array should be ignored).

Question 2:

(*2.A. Given some tree properties and element counts, calculate or estimate other counts.*)

Consider a tree T with the following properties:

- T has exactly 12 internal nodes,
- Every internal node of T has one or two children.
- The average number of children for an internal node is exactly 1.5.
- Tree T is built from `TNode` structures:

```
struct TNode { int info; TNode* left; TNode* right; };
```

(A) What is the number of leaves in tree T?

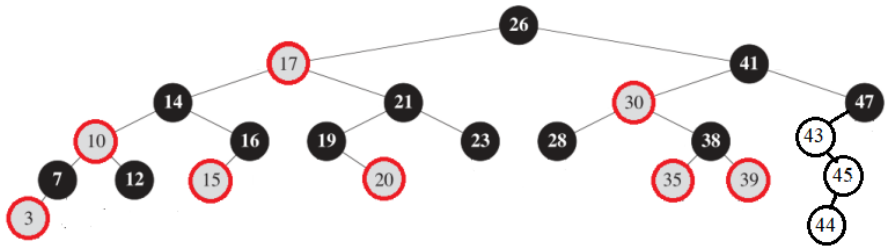
(B) What is the number of `NULL` values among all the `TNode.left` and `TNode.right` pointers used to build the tree T?

(C) What is the largest and the smallest value of the height of the tree T. (Write your estimates for minimum height and maximum height, and explain why these estimates cannot be improved.)

Note: We define *height* of a tree as the number of edges to traverse on the path that connects its root with the deepmost leaf. (In particular, a tree with just the root node has height 0, but a tree with the root and a single child leaf has height 1.)

Question 3:

(*3.A. Perform insert and delete operations in an arbitrary binary search tree.*)



(A) Draw the Binary Search tree obtained when the tree shown in figure has its node 47 deleted.

(B) Draw the Binary Search tree obtained when the tree from (A) has its node 26 deleted.

note To avoid ambiguity, try to replace the node to be deleted with its inorder successor whenever it is possible. (You do not need to balance the resulting tree as an AVL or as a red-black tree – as long as it stays a binary search tree and preserves the tree ordering property.)

Question 4:

(*4.D. Use and analyze Heapsort.*)

An array of 10 elements is used to initialize a minimum heap (as the first stage of the Heap sort algorithm):

$$\{5, 3, 7, 10, 1, 2, 9, 8, 6, 4\}$$

Assume that the minimum heap is initialized in the most efficient way (inserting elements level by level – starting from the bottom levels). All slots are filled in with the elements of the 10-element array in the order they arrive.

(A) How many levels will the heap tree have? (The root of the heap is considered L_0 – level zero. the last level is denoted by L_{k-1}. Just find the number k for this array.)

(B) Draw the intermediate states of the heap after each level is filled in. Represent the heap as a binary tree. (If some level L_k is only partially filled and contains less than 2^k nodes, please draw all the nodes as little circles, but leave the unused nodes empty.)

(C) What is the total count of comparisons ($a < b$) that is necessary to build the final minimum heap? (In this part you can assume the worst case time complexity – it is not necessarily achieved for the array given above.)

Question 5:

(*5.A. Use and analyze Selection sort, Insertion sort, Bubble sort algorithms.*)

```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat
        swapped := false
        for i := 1 to n-1 inclusive do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
                /* swap them and remember something changed */
                swap(A[i-1], A[i])
                swapped := true
            end if
        end for
    until not swapped
end procedure
```

The image shows Bubble sort pseudocode for a 0-based array $A[0] \dots A[n-1]$ of n elements.

(A) How many comparisons (`A[i-1] > A[i]`) in this algorithm are used to sort the given array. Show the state of the array after each `for` loop in the pseudocode is finished.

$$A[0] = 9,\ 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ A[9] = 8.$$

(B) How many comparisons (`A[i-1] > A[i]`) in this algorithm are used to sort the following array:

$$A[0] = 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ A[9] = 0.$$

Question 6:

(*6.C. (C++ code) Use STL classes for priority queue operations.*)

Create a C++ program that reads from the standard input a positive integer n, and then exactly n space-separated integers. It should output the median of these integers to the standard output as a real number. Use `std::priority_queue` to find the median. (You can assume that the program will receive data that matches this description and you do not need to handle erroneous input.)

Note: Recall that the *median* of a collection of n numbers $a_1, a_2, \ldots, a_n$ is the middle number in the sorted order of a_i (if n is odd), or the arithmetic mean of the two middle numbers in the sorted order of a_i (if n is even).