COMP2119A Introduction to Data Structures and Algorithms
Homework 3 **Due Date:** 7pm, Nov 20, 2018

Rules: discussion of the problems is permitted, but writing the assignment together is not (i.e. you are not allowed to see the actual pages of another student). Please make your answers precise and concise.

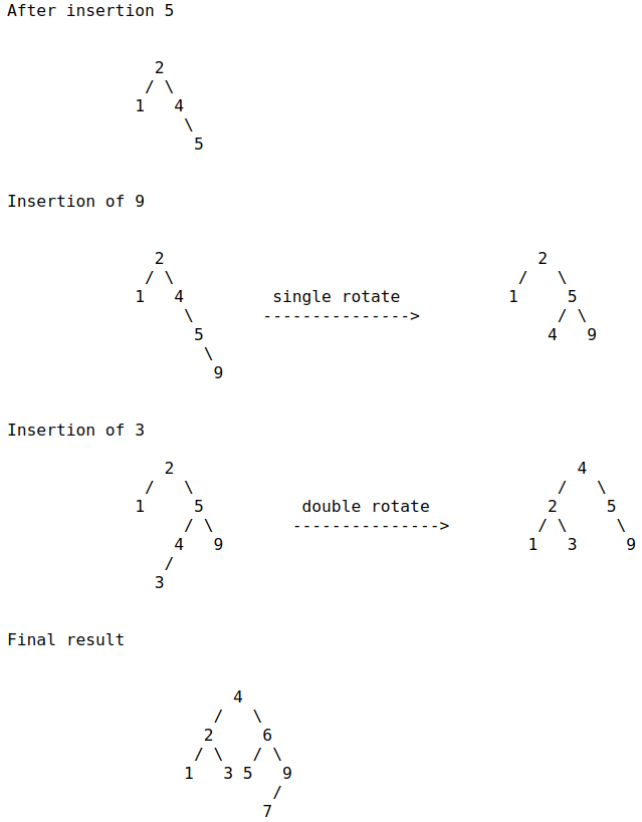
Course Outcomes

- **[O1]. Mathematics foundation**
- **[O2]. Basic data structures**
- **[O3]. Problem solving**
- **[O4]. Implementation**

1. **[O4]**(20 pts)

   Show the result and intermediate steps of inserting 2,1,4,5,9,3,6,7 into an initially empty AVL tree.

   Solution: .



2. **[O1]** (20 pts) Assume the array size n is large enough.

   (a) What is the worst-case asymptotic running time of heap-sort?

(b) What is the worst-case asymptotic running time of merge-sort?

(c) What is the worst-case asymptotic running time of quick-sort?

(d) Can heap-sort be done in-place?

(e) Can merge-sort be done in-place?

(f) Can quick-sort be done in-place?

(g) Consider one item in an array that is sorted with mergesort. In asymptotic (big-O) terms, how many times can that one item move to a different location?

(h) What is the asymptotic running time of quick-sort if the array is already sorted (or almost sorted) and the pivot-selection strategy picks the leftmost element in the range-to-be-sorted?

(i) What is the asymptotic running time of quick-sort if the array is already sorted (or almost sorted) and the pivot-selection strategy picks the rightmost element in the range-to-be-sorted?

(j) What is the asymptotic running time of quick-sort if the array is already sorted (or almost sorted) and the pivot-selection strategy picks the middle element in the range-to-be-sorted?

In-place: Basically it means using $O(1)$ extra space. Check `https://en.wikipedia.org/wiki/In-place_algorithm`.
Choose answer for each question from {yes, no, $O(n \log n)$, $O(n^2)$, $O(\log n)$, $O(n)$, $O(\sqrt{n})$, $O(\log \log n)$}. You do not need to prove your answer.

Solution:

a $O(n \log n)$

b $O(n \log n)$

c $O(n^2)$

d yes

e no

f yes / no

g $O(\log n)$

h $O(n^2)$

i $O(n^2)$

j $O(n \log n)$

Notice: for question (f), both yes and no are considered as correct answer. The "In-place" definition above, using $O(1)$ extra space, is too narrow. More broadly, in-place means that the algorithm does not use extra space for manipulating the input but may require a small though nonconstant extra space for its operation. In quick sort, this space is $O(logn)$, the size of stack space.

3. **[O3]** (25 pts) Modify randomized quick-sort to find the k-th smallest integer in an unsorted array $A[1..n]$ of distinct integers. Your algorithm should run in expected $O(n)$ time.

Solution:

```
function K-SMALLEST(A[i..j], k)
    if i == j then
        return A[i].
    end if
    v ← Random Pivot in A[i..j].
    m ← PARTITION(A[i..j], v)                    ▷ [i, m] ≤ v and [m + 1, j] > v
    if i + k − 1 <= m then
        K-SMALLEST(A[i..m], k)
    else
        K-SMALLEST(A[m + 1..j], k − (m − i + 1))
    end if
end function
```

4. **[O2]** (35 pts) You are given an integer b and an *almost sorted* array $A[1..n]$ of distinct integers, where each element can be misplaced by at most b positions. (i.e. the sorted position of the i-th element $A[i]$ is at least $i - b$ and at most $i + b$.) You are asked to design algorithms to sort the array A.

   (a) (15 pts)Design an algorithm to sort A in $O(n \cdot b)$ time.
   [Hint: Modify selection sort.]

   (b) (20 pts)Design an algorithm to sort A in $O(n \cdot \log b)$ time.
   [Hint: Use a heap.]

   Solution:

   (a)
   ```
   function SORT(A[1..n], b)
       for i from 1 to n do
           m ← argmin_{i≤j≤i+b} A[j].          ▷ O(b) time.
           SWAP(A[i], A[m])
       end for
   end function
   ```

   (b)
   ```
   function SORT(A[1..n], b)
       idx ← 1.
       H ← Priority Queue.                    ▷ Implemented using a heap.
       for i from 1 to n do
           while idx ≤ min(i + b, n) do
               INSERT(H, A[idx]).
               idx ← idx + 1.
           end while
           A[i] ←MIN(H).
           DELETE-MIN(H).
       end for
   end function
   ```