University of Waterloo
CS240 Winter 2020
Assignment 2

Due Date: Wednesday, February 12 at 5:00pm

Please read `http://www.student.cs.uwaterloo.ca/~cs240/w20/guidelines.pdf` for guidelines on submission. This assignment contains both written problems and a programming problem. Submit your written solutions electronically as a PDF with file name a2Submit.pdf using MarkUs. We will also accept individual question files named a2q1.pdf, a2q2.pdf, ... , a2q6.pdf if you wish to submit questions as you complete them.

Note: you may assume all logarithms are base 2 logarithms: $\log = \log_2$.

Problem 1 [8 marks]

Generalize *quickSelect*1 to work on two input arrays. Let the resulting algorithm be called $quickSelect2Arrays(A, B, k)$. Arrays A and B are of size n and m, respectively, and $k \in \{0, 1, ..., n + m - 1\}$. Algorithm $quickSelect2Arrays(A, B, k)$ should return the item that would be in $C[k]$ if C was the array resulting from merging arrays A and B and C was sorted in non-decreasing order.

Your algorithm $quickSelect2Arrays(A, B, k)$ must be in-place, i.e. only $O(1)$ additional space is allowed. Briefly and informally (one or two sentences) argue that the time complexity of your algorithm is the same as of *quickSelect*1, i.e. $O(v)$ in the average case where v is the total number of elements in A and B, i.e. $v = n + m$. Hint: use the same pivot-value for partitioning both arrays.

Problem 2 [4+5(+6)=9(+6) marks]

For this question, assume *Quicksort* algorithm selects the last element of A as pivot, and $partition(A, p)$ is as below:

```
$partition(A, p)$
$A$: array of size $n$,   $p$: integer s.t. $0 \le p < n$
    Create empty lists $small$ and $large$.
    $v \leftarrow A[p]$
    for each element $x$ in $A[0, \dots, p-1]$ or $A[p+1 \dots n-1]$
        if $x < v$ append $x$ to $small$
        else append $x$ to $large$
    $i \leftarrow size(small)$
    Overwrite $A[0 \dots i-1]$ by elements in $small$
    Overwrite $A[i]$ by $v$
    Overwrite $A[i+1 \dots n-1]$ by elements in $large$
    return $i$
```

a) Give the recurrence equation for Quicksort when the input array A contains n copies of the same element, i.e. $A = [x, x, \dots, x]$. Briefly explain how you got your recurrence equation. You are not required to solve the recurrence equation.

b) Suppose we use the following approach to choosing the pivot in Quicksort: First, compute the mean $\bar{n}$ of the elements in the array. Then choose as the pivot the element x of the array, such that $|x - \bar{n}|$ is minimized, i.e., pick the element closest to the average value in the array. Let us call this modified Quicksort algorithm MeanQuicksort.

Assume that the elements of the array form an arithmetic sequence (i.e., have the form a, $a + k$, $a + 2k$, $a + 3k$, $\dots$, $a + (n-1)k$), scrambled in some order. Show that, under this distribution of array elements, MeanQuickSort always runs in $O(n \log n)$ time.

c) **Bonus**: Give an example for which MeanQuickSort in part (b) runs in $\Omega(n^2)$ time, and explain why this running time is achieved.

Problem 3 [5+4=9 marks]

You have found a treasure chest filled with $n \ge 3$ coins. Exactly 2 coins are counterfeit, the rest are genuine. All genuine coins weigh the same and the two counterfeit coins weigh the same, but a counterfeit coin weighs less than a genuine coin. Your task is to separate the genuine coins from the counterfeit coins. To accomplish this you will compare the weight of pairs of subsets of the coins using a balance scale. The outcome of one weighing will determine that each subset of coins weighs the same, or that one or the other subset of coins weighs more.

a) Give a precise (not big-Omega) lower bound for the number of weighings required in the worst case to determine which coins are genuine and which are counterfeit for $n \ge 3$. Briefly justify.

b) Describe an algorithm called `FindGenuine` to determine the genuine coins when n = 4. Use the names C_1, C_2, C_3, C_4 for the four coins, and the function

$$CompareWeight\left(\{first\_subset; second\_subset\}\right),$$

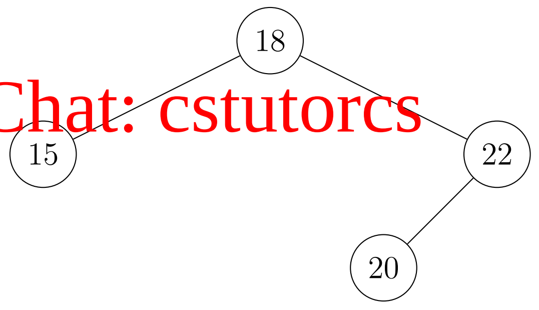
which returns 1 if *first_subset* weighs more, 0 if both subsets weigh the same, and -1 if *second_subset* weighs more. Your function should return the set of genuine coins. Give an exact worst-case analysis of the number of weightings required by your algorithm. For full marks, this should match exactly the lower bound from Part (a).

Problem 4 [6 marks]

Let $A[0...n-1]$ be an unsorted array of positive integers, in the range of $(0, ..., n^5)$. Entries in A are not necessarily unique. Design an algorithm that tests whether there are two numbers in A that are exactly ten apart, i.e. $|A[i] - A[j]| = 10$ for some indexes $i, j \in \{0, 1, ...n-1\}$. For example, the algorithm should return "yes" if the array is $A = [4, 1, 10, 6, 5, 11]$, because numbers 1 and 11 are 10 units apart. It should return "no" on $A = [10, 5, 13, 1, 14, 2]$. The worst-case run-time of your algorithm must be $O(n)$. You can use $O(n)$ additional space.

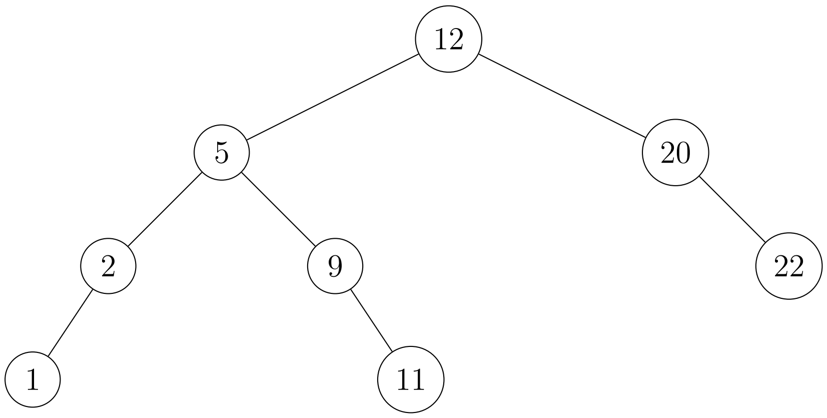
Problem 5 [4+4=8 marks]

a) Consider the following AVL-tree:



Insert key **23** into this AVL-tree. Then insert key **24** into the result. Show the resulting AVL-tree. For full credit it is enough to show the correct final tree, but in case of error, partial credit may be given for the intermediate correct trees. Use the algorithm and rotations as defined in class. If you draw intermediate trees, clearly indicate which tree is your final answer.

b) Consider the following AVL-tree:



Delete key **20** from this AVL-tree, using the algorithm from class. Show the resulting AVL-tree. For full credit it suffices to show the correct final tree, but in case of error, partial credit may be given for intermediate trees. If you draw intermediate trees, clearly indicate which tree is your final answer.

Problem 6 [4+3+4=11 marks]

Let an AVL*-tree be a binary search tree where for every node, the difference of heights of its left and right subtree is at most 2. Let $N(h)$ be the minimum number of nodes for AVL*-tree of height h.

a) What is $N(h)$ for $h \in \{0, 1, 2, 3\}$?. For justification, it is sufficient to draw the smallest AVL*-trees of heights $h \in \{0, 1, 2, 3\}$.

b) Derive a recursive formula for $N(h)$ for $h \geq 3$. Give a brief explanation of your derivation.

c) Show that in any AVL*-tree of height $h \geq 0$ with n nodes, we have $h \leq 3 \log_2(n)$.