University of Waterloo
CS240 Fall 2020
Assignment 2

Due Date: Wednesday, Oct 7 at 5pm

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every asessment in this course you must read and sign an Academic Integrity Declaration before you start working on the assessment and submit it **before the deadline of October 7th** along with your answers to the assignment; i.e. **read, sign and submit A02-AID.txt now or as soon as possible**. The agreement will indicate what you must do to ensure the integrity of your grade. If you are having difficulties with the assignment, course staff are there to help (provided it isn't last minute).

The Academic Integrity Declaration must be signed and submitted on time or the assessment will not be marked.

Please read `http://www.student.cs.uwaterloo.ca/~cs240/f20/guidelines/guidelines.pdf` for guidelines on submission. **Each question must be submitted individually to MarkUs as a PDF** with the corresponding file names: a2q1.pdf, a2q2.pdf, ... , a2q6.pdf .

It is a good idea to submit questions as you go so you aren't trying to create several PDF files at the last minute. **Remember, late assignments will not be marked.**

Problem 1 [3+3=6 marks]

a) Given a heap H, suppose we want to search for an item with key k and minimize (as best we can) the number of entries we check for key k.

i) Describe an efficient algorithm *heapSearch(H, k)* that searches H for k and returns the index where k is found or -1 if not found. If the heap contains multiple entries with key k, any index where the item has key k is okay.

ii) For your algorithm, give a lower bound on the best-case runtime and a lower bound on the worse-case runtime. Briefly justify the runtimes.

iii) Are the lower bounds from part ii) actually tight (i.e. the same as the corresponding upper bounds)? Briefly justify your answer.

b) Given a heap H, suppose we want to remove an *arbitrary* item at index i. Describe an efficient algorithm *heapRemove(H, i)* that removes the item at index i from heap H. State and justify the algorithm's runtime.

Problem 2 [6 marks]

Generalize *quickSelect*1 to work on two input arrays. Let the resulting algorithm be called *quickSelect*2*Arrays*(A, B, k). Arrays A and B are of size n and m, respectively, and $k \in \{0, 1, ..., n + m - 1\}$. Algorithm *quickSelect*2*Arrays*(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 *quickSelect*2*Arrays*(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 3 [2+3+3=8]

A clever student (let's call him Sajed) thinks he can avoid the worst-case behaviour of QuickSort by employing the following pivot-selection procedure. First, compute the mean $\bar{M}$ of the elements in the array. Then choose as the pivot the element x of the array, such that $|x - \bar{M}|$ is minimized, i.e., pick the element closest to the average value in the array. Everything else is the same as QuickSort. He calls the modified QuickSort algorithm SSort.

a) Write down the recurrence for running time $T(n)$ of SSort. In doing so, assume x is placed at index i of the partitioned array. The recurrence relation may be expressed in terms of n and i.

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

c) Unfortunately, Sajed's scheme is not as clever as it looks. Give an example of an array where SSort achieves its worst case runtime of $\Theta(n^2)$ and briefly explain why this example requires this time.

Problem 4 [8 marks]

Given an array $A[0 \ldots n - 1]$ of numbers, such that $A[i] \geq A[i - j]$ for all $0 \leq i \leq n - 1$ and $\log n \leq j \leq i$, design an algorithm to sort A in $O(n \log \log n)$ time.

Hint: Partition A into contiguous blocks of size $(\log n)$; i.e. the first $(\log n)$ elements are in the first block, the next $(\log n)$ elements are in the second block, and so on. Then, establish a connection between the elements within two blocks, which are separated by another block.

Problem 5 [2+2+4=8 marks]

Consider the problem of finding the location of a given item k in an array of n distinct integers. The following randomized algorithm selects a random index and checks whether its entry is the desired value. If it is, it returns the index; otherwise, it recursively calls itself.

Recall that *random(n)* returns an integer from the set of $\{0, 1, 2, \ldots, n-1\}$ uniformly and at random.

```
find-index(A, n, k)
1: i ← random(n)
2: if A[i] == k then
3:     return i
4: else
5:     return find-index(A, n, k).
6: end if
```

In your answers below, be as precise as possible. You may use order notation when appropriate. Briefly justify your answers.

a) What is the **best-case** running time of `find-index`?

b) What is the **worst-case** running time of `find-index`?

c) Let $T(n)$ be the expected running time of `find-index`.
Write a recurrence relation for $T(n)$ and then solve it.

Problem 6 [3+2+3=8 marks]

In a game of loonie poker, all bids are placed using the Canadian loonie (one dollar coin). At the end of the night there is one winner who walks away with all the loonies. Unfortunately, some of the players were using counterfeit loonies. Suppose there are n loonies, some of which are genuine and others that are counterfeit (there is at least one of each). All genuine loonies weigh the same and all counterfeit loonies weigh the same, but the counterfeit coins weigh less than the genuine coins. The goal is to separate the genuine coins from the counterfeit coins by comparing the weight of pairs of subsets of the coins using a balance scale. The balance scale gives one of the following outcomes:

- the two subsets of coins weigh the same
- the subset on the left weighs more than the subset on the right
- the subset on the right weighs more than the subset on the left

a) Give a formula for the 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; i.e. your formula should produce an integer value that is the minimum number of weighings necessary in the worst case.

b) Describe an algorithm called `FindGenuine` to determine the genuine coins when $n = 4$. Use the names L1, L2, L3, L4 for the four loonies, and the function

$$BalanceResult\left(\{left\_subset; right\_subset\}\right),$$

which returns either "left weighs more", "right weighs more", or "both weigh the same". Your function should return the set of genuine loonies.

Give an exact worst-case analysis of the number of weighings required by your algorithm. For full marks, this should match exactly the lower bound from Part (a) when $n = 4$.

c) Describe an algorithm to determine the genuine loonies, for any n. Use the names L1, L2, ... , Ln and the *BalanceResult* subroutine from Part (b). Analyze your algorithm by counting the number of weighings. Your answer, for this part, should be $O(a(n))$ where $a(n)$ is the precise cost you found in part (a) of this problem.