# University of Waterloo
# CS240 Fall 2017
# Assignment 2

**Due Date: Wednesday, October 4, at 5pm**

Please read `http://www.student.cs.uwaterloo.ca/~cs240/f17/guidelines.pdf` for guidelines on submission. This assignment contains written questions and a programming question. Submit your written solutions electronically as a PDF with file name a02wp.pdf using MarkUs. We will also accept individual question files named a02q1w.pdf, a02q2w.pdf, a02q3w.pdf, a02q4w.pdf and a02q5w if you wish to submit questions as you complete them. Submit the programming question in a file named `teampq.cpp`.

## Problem 1 [3+4+3 = 10]

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 $\bar{n}$ 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. 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 $a$, $a+k$, $a+2k$, $a+3k$, $\ldots$, $a+(n-1)k$), 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 for which SSort runs in $\Theta(n^2)$ time, and explain why the worst-case running time is achieved.

## Problem 2 [10 marks]

Given an array $A[0 \ldots n-1]$ of numbers, show that if $A[i] \geq A[i-j]$ for all $j \geq \log n$ that the array can be sorted 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 3 [3+3+5=11 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 4 [6+7+7=20 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)** [6 marks] 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.

**b)** [7 marks] 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)** [7 marks] Describe an algorithm to determine the genuine loonies, for any $n$. Use the names L1, L2, ... , L$n$ 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.

## Problem 5 [3+5+10 marks]

**a)** Show that an *arbitrary* item can be removed from a heap in $O(\log n)$ time, if the item's index into the heap is known. Specifically, give an algorithm *heapRemove(h, i)* that removes the element at index $i$ from heap $h$. Justify your algorithm's running time.

**b)** Consider the following C++ class:

```
class Team {
   int wins;
   int losses;
   string name;
   // You may add fields/methods/constructors/destructor as necessary
};
```

We wish to build a data structure that allows for efficient retrieval of the team with the most wins and also for efficient retrieval of the team with the fewest losses. Essentally we want a priority queue that supports two different priority measures. Provide an implemenation of the following class:

```
class TeamPQ {
   // add fields/methods/constructors/destructor as necessary
 public:
   void insert(const Team &t);   // O(log n) time

   const Team &findMaxWins() const;    // O(1) time
   const Team &findMinLosses() const;  // O(1) time

   void removeMaxWins();     // O(log n) time
   void removeMinLosses();   // O(log n) time
};
```

Give pseudocode implementations of these routines, justify the runtimes, and submit with your written submission. Then implement this class in C++ and submit electronically (the remainder of these instructions pertain to the programming portion of this assignment).

Provide a main function that accepts the following commands from stdin (you may assume that all inputs are valid):

- `i wins losses name` - inserts a team with the given wins, losses, and name into the priority queue. You may assume that `name` contains no whitespace.
- `pw` - prints the name of the team with the most wins, without removing it from the priority queue. Prints nothing if the priority queue is empty.
- `pl` - prints the name of the team with the fewest losses, without removing it from the priority queue. Prints nothing if the priority queu is empty.
- `rw` - removes the team with the most wins from the priority queue and prints nothing. Does nothing if the priority queue is empty.
- `rl` - removes the team with the fewest losses from the priority queue and prints nothing. Does nothing if the priority queue is empty.

All output is printed to stdout. The program terminates when eof is encountered.

For example, the following input:

```
i 15 5 jays
i 13 3 yankees
i 12 7 orioles
i 10 10 rays
i 0 14 sox
pw
pl
rw
pw
rl
pl
i 20 1 jays
pw
pl
```

should produce the following output:

```
jays
yankees
yankees
orioles
jays
jays
```

Place your entire program in the file `teampq.cpp`