# Homework Assignment #2

Due: Noon, Oct 22, 2018
Submit **printed** solutions via eClass
Submit **handwritten** solutions at class, by noon

CMPUT 204
Department of Computing Science
University of Alberta

---

**Note:** All log's are in base 2, unless specified otherwise.

You can use the fact $H(n) = \sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1)$.

This assignment is partitioned into Exercises and Problems. Exercises are optional and will not be graded. Problems are mandatory and will be graded. You are, however, strongly advised to work out a full solutions for both.

---

**Exercise I.** Solve the following recurrence relations. Unless specified otherwise, you may assume $T(n) = O(1)$ for $n = 0, 1, 2$ or any other small constant.

1. $T(1) = 0$ and $T(n) = 1 + T(\log(n))$.
2. $T(n) = 3T(n/3) + \sqrt{n^3}$.
3. $T(n) = T(n-3) + 3\log n$.
4. $T(n) = 4T(\frac{n}{7}) + n^{1.47474747\ldots}$.

---

**Exercise II.** Your boss has given you an assignment: to devise the fastest algorithm you can for solving a particular problem whose input is an array $A$ with $n$ elements. After mulling the problem over for a few days you come up with a few alternatives. Which alternative do you prefer and why? Explain.

- Algorithm 1:
  Iterate over each element in $A$ and do at most $100n$ operations per element.
- Algorithm 2:
  For inputs of size $n > 1$:
  Repeat a hundred times the procedure of: (i) recursing over an input of size $n/2$ and (ii) make a single operation.

---

**Problem 1.** (20 pts) Find asymptotic upper/lower bounds for $T(n)$. Show how you derived the solution and prove your bound.
Unless specified otherwise, assume that in each case $T(1) = 1$ (or any small constant).

**(a)** (4 pts) $T(n) = T(n-1) + \frac{7}{n}$ with $T(0) = 0$.

**(b)** (4 pts) $T(n) = 2T(\frac{n}{2}) + \frac{n}{\log(\log(n))} + n$.

**(c)** (4 pts) $T(n) = T(\lfloor\sqrt{n}\rfloor) + 4n$.

**(d)** (4 pts) $T(n) = 8T(\frac{n}{2}) + (n \log n)^3$.

**(e)** (4 pts) $T(n) = T(n-1) + a \cdot n^c$ for some constants $a, c > 0$.

---

**Problem 2.** (20 pts) Your boss has given you an assignment: to devise the fastest algorithm you can for solving a particular problem whose input is an array $A$ with $n$ elements. After mulling the problem over for a few days you come up with a few alternatives. Which alternative do you prefer and why? Explain.

**(a)** (8 pts)

- Algorithm 1:
  Iterate over each element in $A$ and do at most 333 operations per element.
- Algorithm 2:
  For inputs of size $n > 1$:
  1. Recurse on the first half of the input
  2. Recurse on the latter half of the input
  3. Make 333 arithmetic operations

**(b)** (12 pts)

- Algorithm 1:
  For inputs $A$ of size $n > 1$:
  1. Recurse on an instance of size $n - 1$
  2. Take 999 basic operations
  3. Recurse on a different instance of size $n - 1$
- Algorithm 2:
  For inputs $A$ of size $n > 1$:
  1. Recurse on an instance of size $n - 1$
  2. Iterate over each element in $A$ and do at most $O(2^{n^{0.999}})$ operations per each element.
- Algorithm 3:
  Iterate over all possible *subsets* of elements of $A$ and do 999 operations per subset.

---

**Problem 3.** (15 pts) The following are all examples of recurrence-relations for a function $T$ that takes on nonnegative values and is defined on the natural numbers.
Prove, using induction, that for each example $T(n) \in \Theta(n)$. You may ignore any rounding issues that might arise.

**(a)** (5 pts)

$$T(n) = \begin{cases} 0, & \text{if } n = 0 \\ 10n, & \text{if } 1 \leq n \leq 9 \\ T(\frac{7}{10}n) + T(\frac{1}{5}n) + 3n, & \text{if } n > 9 \end{cases}$$

**(b)** (5 pts)

$$T(n) = \begin{cases} 0, & \text{if } n = 0 \\ 9999n, & \text{if } 1 \leq n \leq 99 \\ T(0.9n) + T(0.09n) + 999n, & \text{if } n > 99 \end{cases}$$

**(c)** (5 pts) Let $(m_0, m_1, m_2, ..., m_n, ...)$ be an arbitrary series such that for every $n \geq 2$ we have that $m_n$ is some natural number $\leq n - 1$.

$$T(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ T(m_n) + T(n - m_n - 1) + 1, & \text{if } n \geq 2 \end{cases}$$

---

**Problem 4.** (15 pts) Consider the following elegant(?) sorting algorithm:

```
procedure SomeSort(A, b, e)      ** Sorts the subarray A[b, .., e]
if (e − b + 1 ≤ 3) then
    Sort A[b..e] using any sorting algorithm
        ** Even a brute-force comparison checking all 6 possible ways of sorting, seeing as it takes only O(1)-time
else
    p ⟵ ⌊(e−b+1)/4⌋
    SomeSort(A, b, e − p)
    SomeSort(A, b + p, e)
    SomeSort(A, b, e − p)
end if
```

**(a)** (2 pts) Illustrate the behavior of `SomeSort` on the input $A = [44, 33, 22, 11]$ with $b = 1$ and $e = 4$.

**(b)** (6 pts) Prove that for any natural $n \geq 1$, `SomeSort` correctly sorts any input array $A$ of size $n = e-b+1$.

**(c)** (6 pts) Find a recurrence relation for the worst-case running time of `SomeSort`. Give a tight (i.e. $\Theta$) asymptotic bound for the worse-case running time of `SomeSort`.
You may assume $n$ is a power of 4 if it helps.

**(d)** (1 pts) Compare the WC-runtime `SomeSort` to that of the other sorting algorithms we've learned and determine when (if) one would prefer `SomeSort` to those algorithms.

---

**Problem 5.** (15 pts) Here is another sorting algorithm, known as Bubble-Sort.

```
procedure BubbleSort(A, n)  ** precondition: A is an array containing n pairwise comparable elements
i ← 1
while (i ≤ n − 1)
    for (j from 1 to n − 1)
        if (A[j] > A[j + 1])
            exchange A[j] ↔ A[j + 1]
    end for
    i ← i + 1
end while
```

**(a)** (1 pts) Exhibit how `BubbleSort` modifies the array $[15, 5, 12, 3]$. Include all intermediate stages.

**(b)** (5 pts) Prove that `BubbleSort` is correct.
Your answer should include the formal loop-invariant of the while-loop and a proof of said LI.
Hint: Look at the keys 15 and 12 in the above-mentioned example. At what iteration does each of those keys take its correct position?

**(c)** (2 pts) What is, asymptotically, the number of key-comparisons `BubbleSort` makes on any input of $n$ elements?

**(d)** (3 pts) Based on the correctness proof, Haozhou has revised `BubbleSort` to the following version.

```
procedure Haozhou_BubbleSort(A, n)  ** A is an array that contains n pairwise comparable elements
i ← 1
while (i ≤ n − 1)
    for (j from 1 to n − i)        ** The revision is ONLY in this line
        if (A[j] > A[j + 1])
            exchange A[j] ↔ A[j + 1]
    end for
    i ← i + 1
end while
```

Does Haozhou's version make *significantly* fewer key-comparisons than the original `BubbleSort`?

**(e)** (4 pts) Sepehr also revised `BubbleSort`, by cleverly using a Boolean variable $flag$.

```
procedure Sepehr_BubbleSort(A, n)  ** A is an array containing n pairwise comparable elements
flag ← TRUE
i ← 1
while (i ≤ n − 1 and  flag =TRUE)
            ** Now, in order to make another iteration of the while-loop it must hold that j isn't too large
            ** and that flag is set to TRUE
    flag ← FALSE
    for (j from 1 to n − 1)
        if (A[j] > A[j + 1])
            exchange A[j] ↔ A[j + 1]
            flag ← TRUE
    end for
    i ← i + 1
end while
```

Does Sepehr's version make significantly fewer key-comparisons than the original `BubbleSort`?
**Discuss both the worst-case and the best-case.**

---

**Problem 6.** (15 pts) Both Alice and Beth were given the task of picking $m$ *random* items out of $n$ possible items. However, Alice picks the $m$ items *with* repetitions (so she repeats $m$ times the loop "pick a u.a.r chosen item from the $n$ possible items); whereas Beth picks the $m$ items *without* repetitions (so she initially sets $S$ as the set of all possible items, and repeats $m$ times the loop "pick a u.a.r chosen item from $S$ and remove it from $S$"). And so, in Alice's set of $m$ items *it could be* that some item appears multiple times; whereas in Beth's set no item can appear more than once.

In this question, your goal is to formalize the probability of each of the two of picking up a particular item $i$. So fix some item $i$ and denote $E$ as the event $E =$"$i$ is among the $m$ chosen items."

**(a)** (1 pts) If $m = 1$ (both pick a single item), what is the probability of $E$ holding under Alice's sampling technique, i.e. $\Pr_{\text{Alice}}[E]$; and what is the probability of $E$ holding under Beth's technique, i.e. $\Pr_{\text{Beth}}[E]$?

**(b)** (1 pts) In a sentence: what is the meaning of the complimentary event $\neg E$?
And so, if $m = 1$, what is $\Pr_{\text{Alice}}[\neg E]$ and what is $\Pr_{\text{Beth}}[\neg E]$?

**(c)** (10 pts) Assume $m > 1$. Whose technique, with or without repetitions, has a higher chance of picking a particular item $i$? Namely, who has the higher probability of $E$ holding — Alice or Beth?
Hint#1: Start with the special case of $m = n$, this should guide you as to the answer in the general case.
Hint#2: Leverage on the the previous article. Instead of arguing about $\Pr[E]$, reason about $\Pr[\neg E]$.

**(d)** (3 pts) Assume $n \geq 100$. Show that when both techniques are used for picking half of the items, namely when $m = \frac{n}{2}$, then:
–In the leading of the two techniques (the one where $E$ is more likely to hold) we have that $\Pr[E] = 0.5$
–And in the other technique (the one where $E$ is less likely to hold) we have that $\Pr[E] < 0.4$.
You are welcome to use the following inequality

$$\text{for any } x \geq 2 \text{ we have:} \quad 1 - \tfrac{1}{x} \geq e^{-\frac{1}{x} - \frac{1}{x^2}}$$

and the fact that $e^{-\left(\frac{1}{2} + \frac{1}{200}\right)} > 0.6$. (Instead of you using a calculator to check the value of $e^{-\left(\frac{1}{2} + \frac{1}{200}\right)}$, we have done this for you.)