CSCI 3030U
Assignment #1

---

(1) *Asymptotic complexity*

Consider the following algorithm.

```
function f(x: List[Integer]) {
  var n: Integer = Length(x)
  if(n == 1) {
    return x[0] == 0
  }
  else if(n == 0) {
    return false
  } else {
    var y1 = n/3;
    var y2 = 2 * n/3;
    return (f(Sublist(x, 0, y1)) or
            f(Sublist(x, y1+1, y2) or
            f(Sublist(x, y2+1, n-1)))
  }
}
```

(1.1) Describe the functionality of the function f in one sentence.

(1.2) Formulate a recursive equation of the complexity of the function f in terms of the length of the input list x.

(1.3) Solve the recursive equation, and determined the runtime complexity of f in terms of its asymptotic complexity.

(1.4) Rewrite f using a non-recursive formulation.

(1.5) Compare the run-time complexity of $f_{\text{recursive}}$ and $f_{\text{iterative}}$ by their asymptotic run-time complexity. Do they differ in efficiency asymptotically? If so, which is more efficient?

For the following questions. You are allowed to use either Java or Python or C/C++. You must accompany your submission with a *Makefile* which compiles and executes your submission in a Linux environment. Your makefile must support at least these two targets.

`make clean`
: This cleans the project by removing the compiled objects or generated data files (if any).

`make prepare`
: This compiles the project and generates the necessary data files (if any).

`make run`
: This runs the compiled binaries.

---

(2) *Sorting 1 million items*

(2.1) Use either Java or Python to generate 1 million random strings of length 100, and store them in an array. What is the estimated memory consumption of the array?

(2.2) Implement the three sorting algorithms *mergesort*, *quicksort* and *heapsort* on the array.

(2.3) For each sorting algorithm, run the implementation on the 1,000,000 strings multiple times (~100 times). Record the runtime of *each* execution. Plot and report the histogram of the runtime. This step should be executed by *make run*.

---

(3) *Sorting 100 million items*

(3.1) Calculate the approximate space required to store 100 million strings of length 100 in GB. Each GB $= 2^{30}$ B.

(3.2) Write a program to generate 100 million random strings of length 100 each, and store them in a file. This should be executed by *make prepare*.

(3.3) Design a sorting algorithm that works with sorting the file. Your sorting algorithm does not have to be in-place. Clearly describe your algorithm in the style found in the text book. You must be very explicit in terms of the disk I/O functions used in your algorithm in order to deal with the data volume.

Hint: For disk I/O, data transfer during sequential scan is about 100 times after than random access. So, your only hope is to perform multiple passes of sequential scan of the data file. Ask yourself the question: which sorting algorithm uses sequential scan the most? Feel free to utilize auxiliary disk space as needed.

(3.4) Provide the complexity analysis of your disk based sorting algorithm. Using the assumption that the data is uniformly random, estimate the run-time of your algorithm in terms of the number of passes of data you need to perform.

(3.5) Implement your disk based sorting algorithm, and apply it to the data file generated in (3.2). Your program **must** generate an output file **sorted.txt**. Report the run-time. This step should be executed by *make run*.

Hint: If you are using *Java*, make sure all disk I/O is done by *BufferedReader* and *PrintWriter* to ensure that buffered disk I/O is used. *Python's* **`file.readline()`** performs buffered I/O by default.

Marking Scheme & Submission

(1.1) 5	(2.1) 10	(3.1) 5
(1.2) 5	(2.2) 10	(3.2) 5
(1.3) 5	(2.3) 10	(3.3) 10
(1.4) 5		(3.4) 10
(1.5) 10		(3.5) 10
(1) Total 30	(2) Total 30	(3) Total 40

You must include a report with write-up on *each* question.

You must submit all source code and two separate Makefile for (2) and (3).

Your submission folder must conform to the following structure:

```
.
├── Q2
│   └── Makefile
├── Q3
│   └── Makefile
└── report.pdf
```

Marks will be deducted if a non-standard directory structure is used.