Missouri University of Science & Technology — Department of Computer Science

Spring 2023 — **CS 2500: Algorithms (Sec: 102)**

Homework 2: Sorting

Instructor: *Sid Nadendla* — **Due:** *March 3, 2023*

Problem 1. Workflow of Heapsort and Quicksort *20 points*

Demonstrate HEAP-SORT and QUICK-SORT iterations for both the following arrays:
(i) $A_1 = \{2, 6, 4, 3, 1, 5\}$, and (ii) $A_2 = \{1, 5, 2, 3, 0, 2, 2, 1, 4, 5\}$.

Problem 2. Empirical Analysis of Heapsort and Quicksort *20 points*

Implement HEAP-SORT (Page 170 with supporting functions in Pages 165, 167, all in *CLRS*) and QUICK-SORT (Page 183, *CLRS*) in Python, and validate its average run-time performance (similar to Problem 2 in Homework 1).

Problem 3. Modified Quicksort *20 points*

Traditional quicksort routine chooses a pivot q such that $A[p : q-1] \leq A[q] \leq A[q+1, r]$. Instead, present an analysis when the quicksort algorithm partitions the array $A[p : r]$ into three parts using two pivots q_1 and q_2 such that $A[p : q_1 - 1] \leq A[q_1] = \cdots = A[q_2] \leq A[q_2 + 1 : r]$.
(Hint: Assume that the entries in A are picked from $\{1, \cdots, m\}$, where $m < n$.)

Problem 4. Sort by Frequency *20 points*

Write a program in Python that sorts all the integer entries in an input array A of size n according to the decreasing frequency of occurrence. If the frequency of two numbers is the same, then sort them in the increasing order of value. Assume that $A[j] \in \{0, 1, \cdots, k\}$ for all $j = 1, \cdots, n$, and let $k \ll n$ to allow enough number of repetitions.
(Hint: You can find frequencies using COUNTING-SORT).

Example: Let A = $\{$3, 5, 2, 1, 0, 1, 2, 3, 4, 2, 0, 3, 4, 2, 1$\}$. Note that $n = 15$ and $k = 5$. Let $f(i)$ denote the frequency of occurrence of a number i in A. Then, we have

$$\begin{aligned} f(0) &= 2, & f(3) &= 3, \\ f(1) &= 3, & f(4) &= 2, \\ f(2) &= 4, & f(5) &= 1. \end{aligned}$$

Then, the output should look like: $B = \{2, 2, 2, 2, 1, 1, 1, 3, 3, 3, 0, 0, 4, 4, 5\}$.

Problem 5. Extra credit (5 points)

You are strongly encouraged to solve this problem.

SELECTION-SORT(A) sorts the input array A by first finding the j^{th} smallest element in A and swapping it with the element in $A[j]$, in the order $j = 1, j = 2, \cdots, j = n - 1$. Write pseudocode for SELECTION-SORT, and find the best-case and worst-case running times of SELECTION-SORT in Θ-notation.