CS 6301.502. Implementation of advanced data structures and algorithms

Fall 2017

Short Project 5: Partition, Quick sort, Selection

Fri, Sep 22, 2017

Version 1.0: Initial description (Fri, Sep 22).

Due: 11:59 PM, Sun, Oct 1.

Solve as many problems as you wish. Maximum score: 50

For experiments, choose a few values of n between 8M and 512M.

1. [20 points]

Compare the performance of the two versions of partition discussed in class

on the running time of Quick sort, on arrays with distinct elements.

Try arrays that are randomly ordered (by shuffle) and arrays in

descending order.

2. [30 points]

Implement dual pivot partition and its version of quick sort.

Compare its performance with regular quick sort. Try inputs that

are distinct, and inputs that have many duplicates.

3. [30 points]

Implement 3 versions of the Select algorithm (finding k largest elements)

and empirically evaluate their performance:

(a) Create a priority queue (max heap) of the n elements, and use remove() k times.

(b) Use a priority queue (min heap) of size k to keep track of the

k largest elements seen so far, as you iterate over the array.

(c) Implement the O(n) algorithm for Select discussed in class.

4. [20 points]

Compare the performance of your best implementation of Merge sort with

quick sort that uses dual-pivot (or multi-pivot) partition. Try inputs

that are distinct, and inputs that have many duplicates.

5. [50 points]

Implement and compare external versions of the best implementations of

Quick sort and Merge sort. In this version, the numbers to be sorted

reside in a file on the disk. To simulate limited memory, your program

can keep in memory only 1M elements at a time. Intermediate results

have to be written to the disk, and re-read into memory as needed.

The final output will also be stored in a file. Try large values

of n, such as 1G.