CSCI 230 Spring 2017

Project #1

1. **Programming assignment** Do the "Empirical study of sorting algorithms" study in Chapter 7 section 8 and provide you own tables like figures 7.20 running the eight algorithms (notice duplicates-improved versions: Shell-O, Merge-O, Quick-O, Heap-O the Radix-4 and Radix-8 sort), just like Section 7.8 in the textbook. At a minimum run you comparison under both Windows 7-10, or older Win XP/Vista and optional UNIX (Linux or Solaris)in classroom, T-MARC or home.

Compile the files for the sort algorithms in Chapter 7, compare the running time and test the programs from your textbook on : **Exchange sort (insertion, selection, bubble), quicksort, quicksort improved, shellsort with powers of 2, shellsort with powers of 3, heapsort, mergesort, mergesort improved, radixsort/8** . I wand a short evaluation and calibration of your *"timing/profiling"* software as described in class, ask me if you need any details. I leave in your hands how deep you want to go with this. In addition you may use a profiler on both Unix and Windows.

You need to turn in the code and results for test runs and *very important* a report on comparisons based on a running time table and graphics. Your comparisons will look at at running time for at least thee different list sizes. Generate random lists (at lesat 5 sizes) or integers of size 100, 1,000, 10,000, 100,000 , 1,000,000 for input files, you may try further till the runtime becomes significantly expensive, say more than 15 min. You may also want to compare the running time for the same code and input but for different compilers and OS (a good job will earn you extra credit). The project is asking that you do a job **similar to section 7.8 p.259-261 and compare the running time of different algorithms**, see how this backs our theoretical results and if you could find the proportionality constant.

2. **Internet research individual report and class presentation** Do a Google search opf your choice on a topic related to this class Data Structure 2 (see the textbook table of content for keywords - or visit the link indicated in the Lab1 CSCI230 handout) In the past , for example I had the topic "137GB Hard Drive limit"- related to Chapter 8, to learm about HD capacities barriers, the HD CHS (cylinder-head-sector) geometry and more. This problem was addressed by a number of articles, as they were posted around Spring 2003 some may be unavailable, one that I recommended was:
"Hard Drive Size Limitations and Barriers" at:
www.dewassoc.com/kbase/hard_drives/hard_drive_size_barriers.htm
or the paper posted by Maxtor at:
www.maxtor.com/_files/maxtor/en_us/documentation/white_papers/big_drives_white_papers.pdf
Find some interesting papers on this or other topic that will also come in our textbook, Chapter 8.2 Explain in a few paragraphs (no more than 2 pages) what were the historical barrier size limitations, why did they appear what is involved ie hardware (motherboard), BIOS, Operating System,anything else? Are we likely to have a similar problem in ther next few years

Part 1 is a group project, each of you does part 2 and the presentation of it individualy. One paper folder should be submitted for each group for part 1, also a folder containing all the related source files, executables and output + report on drive S. Each member of the group will have to make a presentation. You may opt out of the group if necessary but you still need to complete the whole project.

You have to submit: *a folder on the S drive with source and executable program files, printouts of the source programs and any program outputs as well as your remarks and tables - graphs on benchmarking.* The project is due: **March 29/31, 2017** in class. When you submit the project be ready to answer questions and make a presentation on the project.

Project #2

Programming assignment:

1. a) Implement an external sort based on Replacement Selection an multiway merging as presented in Chapter 8 in the textbook. See question 8.4 p 308. Test your program on files with small records and on files with large records. For what size record do you find that key sorting would be worthwile? Find the theoretical size that is a limit to your algorithm assuming you want your program to run in minutes with a few runs read 8.5.1 p294-296 and use section 8.5.2 pag. 296-300 and multiway merging, section 8.5.3 pag. 300-303.

b) Use the previous external sort modified to use Quicksort for building initial runs, make a comparison of runtime. Make a comparison on the runtime between a) and b). Have in mind the textbook comparison table Fig 8.11 on p.303.

2. Implement the three self-organizing heuristic lists:
a) count
b) move to front
c) transpose.
Compare the cost for runing the three heuristics lists on various input data (this is Ch 9.2 p.320-322 and proj 9.2 p. 348).

You are encouraged to work together, this is a team project. You have to submit:
A report on behalf of all team including source code and printouts. Submit a folder on S-drive with source and executable program files, printouts of the source programs and any program output as well as your remarks.

The project is due: April 26/28 2017 in class. When you submit the project be ready to answer questions on the work you've done in the project.

Projects #3 and #4

Programming assignment:

1. Implement the dictionary ADT of Section 4.4 p.134, 137 by means of a hash table with linear probing for collision resolution, see proj 9.6 p.349.

2. Implement and test the $2-3^{+}$ tree. (Project 10.2 page 378.)

3. Use *sfold* function of Ex9.8 p329 and the *ELFhash* function below to solve a)-c) on problem 9.17 p348(use code given here with M=101).

```
int ELFhash(char* key) {
unsigned long h = 0;
while(*key) {
h = (h << 4) + *key++;
unsigned long g = h & 0xF0000000L;
if (g) h ^ = g >> 24;
h &= ~g; }
return h % M;}
```

4. Implement skip list of section 16.2.2 p.524-530 proj 16.1 p 541

5. Implement the sparse matrix representation of section 12.2 page 416-420 (proj 12.1 p. 435-436). Your implementation should do the following: insert, delete, search for an element, transpose, add and multiply two matrices.

6. Implement the Zip-Lempel like text compression system described in Section 9.2 p. 322-323

7. Extra credit Implement a Database on disk using bucket hashing, see details of project 9.5 p 349

8. Extra credit: Revise the BST class to use the AVL or Splay Tree rotations, your new implementation should not modify the original BST class. Compare the standard BST against the Splay tree over a wide variety of input data and comment on what conditions will make the splay tree save time? (see projects 13.3 and 13.4 on p.464)

9. Extra credit Implement the Strassen algorithm for an efficient matrix multiplication and test it. Look at p.532, 541

10. Extra credit Use the point-region PR quadtree AND the K-D tree to implement a city data base in two ways for the question in project 6.5 p. 226 (see problem description of Project 13.10, 13.11 on page 464) and test it.

You are encouraged to work together, this is a team project. You have to submit:
A report on behalf of all team including source code and printouts. Submit a folder on S drive with source and executable program files, and a paper softvover folder with printouts of the source programs and any program output as well as your remarks.

The project is due: May 24/26 2017 When you submit the project be ready to answer questions on the work you've done in the project.