COSC 3320 Assignment 1

Due: Monday, June 14, 2022, in class

1. Solve the Towers of Hanoi game for the graph G = (V,E) with V={S, A1, A2, A3, A4, D} and E = {(S,A1), (A1,A2), (A2,A3), (A3,A4), (A4,A1), (A4,D)}.
(a) Design an algorithm and determine the time and space complexities of moving n disks from S to D.
(b) Implement this algorithm whereby your program prints out each of the moves of every disk. Show the output for n=1, 2, 3, 4, 5, 6, 7, 8, 9, and 10. (If the output is too long, print out only the first 100 and the last 100 moves.)

2. Determine for the following code how many pages are transferred between disk and main memory (you must count reads and writes separately!), assuming each page has 2000 words, the active memory set size is 1000 (i. e., at any time no more than 1000 pages may be in main memory), and the replacement strategy is LRU (the Least Recently Used page is always replaced); also assume that all two-dimensional arrays are of size (1:4000, 1:4000), with each array element occupying one word, N=4000

```
for I := 1 to 4000 do
   for J :=1 to 4000 do
      { A[I,J]:=A[I,J]*B[I,J]; B[I,J]:=C[J,N-I+1]*A[J,I] }
```

provided the arrays are mapped into the main memory space
(a) in row-major order, (b) in column-major order.

3. Consider QuickSort on the array A[1:n] and assume that the pivot element x (used to split the array A[lo:hi] into two portions such that all elements in the left portion A[lo:m] are ≤x and all elements in the right portion A[m:hi] are ≥x) is the **last** element of the array to be split (i. e., A[hi]).
Construct an infinite sequence of numbers for n and construct an assignment of the numbers 1...n to the n array elements that causes QuickSort, with the stated choice of pivot, to
(a) execute optimally (that is A[lo:m] and A[m:hi] are always of equal size)
(b) execute in the slowest possible way.

Programming

Remember: For each of the next three programs, you are expected to write a report that uses that program as a research tool. The fourth is a standard programming exercise, with standard documentation required.

4. Write a program, using your favorite computer (under some operating system that must support VMM) and your favorite programming language, which demonstrates that the timings of matrix addition differ substantially for large enough matrices, depending whether you use Version 1 or Version 2:

```
for i:=1 to n do                    for j:=1 to n do
    for j:=1 to n do                    for i:=1 to n do
      C[i,j]:=A[i,j]+B[i,j]               C[i,j]:=A[i,j]+B[i,j]
```

Version 1 **Version 2**

Specifically, use this sequence of values for n, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, and 65536, and study the timings of both versions. (Be aware that some runs may take longer than you are willing, or able, to wait!) Keep in mind that theoretically the two versions should have the same timings and the doubling the value of n should result in a quadrupling of time spent to do the addition, assuming everything were done in core (which is of course not the case, since the last value corresponds to a memory requirement of about 40 Gigabytes, assuming four bytes per word). Note that you must initialize your matrices A and B but the time required for this should not be part of the measurements.

5. Memory fragmentation in C: Design, implement, and execute a C-program that does the following: It allocates memory for a sequence of **3m** arrays of size 1 MB each; then it explicitly deallocates all **even**-numbered arrays (about 1.5m such arrays) and allocates a sequence of **m** arrays of size 1.25 MB each. Measure the amounts of time your program requires
1. for the allocation of the 3m arrays of size 1 MB each,
2. for the deallocation of the even-numbered arrays, and
3. for the allocations of the m arrays of size 1.25 MB each.
You must determine **m** so that you exhaust almost all of the main memory available to your program. **Explain your timings!!**

6. Implement a binary search function in **three substantially different** programming languages. In each program (identical, except for the programming language), carry out the **same** 10,000,000 unsuccessful searches for eight different-sized arrays, namely arrays of sizes 100, 200, 400, 800, 1600, 3200, 6400, 12800, 25600, and 51200. Measure in each of the three programs the time it takes to do the 10,000,000 searches for each of the eight arrays. Compare these timings to the theoretical timings the algorithm binary search provides. Are there differences between the three programs? **Explain your timings and observations!!**

7. Design, code, and test a program that implements the optimal Huffman algorithm discussed in class (with a time complexity of $O(n \log_2 n)$, n being the number of code symbols).

Percentage points: **1: 18%** **2: 13%** **3: 12%** **4: 12%** **5:15%** **6:15%** **7:15%**