1806ICT Programming Fundamentals Assignment – Milestone 1

This assignment milestone is worth 20% of your final grade and is due for submission by 11:59pm on Friday, 1st September. You must do this assignment milestone by yourself (including writing ALL the C code) and the normal Griffith University rules for plagiarism apply.

The problem statement for the assignment is as follows:

The ACE software development company requires a program that:

1. Is able to store a very large number of integers (of which there may be duplicates).
2. Allows very quick access for find, add, delete, predecessor, successor, max and min operations.
3. During a normal run of the program, after the data is initialised, 10% of numbers will be randomly deleted and another 10% randomly added. Each of the other operations will occur the same number of times as the adds and deletes and the sequence of operations will be random.

The goal of this assignment is to develop a well structured, memory and computationally efficient C program to simulate the system and investigate possible techniques for achieving the requirements listed above. The program will take 3 integer parameters (*n*, *r1*, *r2*) from the command line with *r1* > 0 and *r2* > *r1*. It will then generate *n* positive random numbers (duplicates allowed) in the range *r1*, … , *r2* inclusive and store them in a data structure. Your program for this assignment will implement all the functionality listed above. Each operation type (e.g. add) will be timed individually and the number of operations, average and total time taken for each operation type displayed.

If a negative value for *n* is submitted then the program will use *abs(n)* and enter a special internal testing mode which demonstrates all operations return the correct result (see Sample Run 3 below).

Investigate and report on your program testing and the following performance aspects of your program:

1. Search techniques (find) that allow a user to efficiently find how many occurrences of a number exist in the data structure. In addition, there must be an efficient facility for determining the predecessor and successor for a nominated number. Note that sequential search is not classified as an efficient search technique.
2. Techniques for efficiently adding more numbers or deleting numbers to/from the data structure once the initial data structure has been created. Note, deleting a number only deletes one occurrence of that number.
3. Techniques for efficiently, at any time, finding the maximum or minimum number stored in the data structure.
4. Discuss data compression techniques that could be used to reduce the size of the data structure and identify the impact of data compression on the techniques identified above in 1 .. 3.

To complete this assignment you must submit your C program as *analyse_nums.c* and the report on your testing and findings in a well-structured Word document titled *nums_results.docx*. The output from your program must be in the same format as shown below. This output was generated by a sample program that was compiled using "cc analyse_nums.c -O3 -o analyse_nums" (all times in seconds):

Sample Run 1 : ./analyse_nums 100000000 1 100000

n = 100000000, r1 = 1, r2 = 100000, Memory used = 0.400000 Mbytes

	Op counts	Total time	Avg. Time
find	9995247	2.338336	2.3394479396e-07
add	10001423	2.346919	2.3465850809e-07
delete	10005569	2.343808	2.3425034598e-07
succ	10000921	2.341604	2.3413883581e-07
pred	9994993	2.330232	2.3313993316e-07
min	10003022	2.224194	2.2235220516e-07
max	9998825	2.221678	2.2219390778e-07

Sample Run 2 : ./analyse_nums 100000000 1 1000000000

n = 100000000, r1 = 1, r2 = 1000000000, Memory used = 440.000000 Mbytes

	Op counts	Total time	Avg. Time
find	9998194	8.355908	8.3574173496e-07
add	10000976	8.804130	8.8032708008e-07
delete	10003965	8.395411	8.3920835389e-07
succ	9999987	8.356767	8.3567778638e-07
pred	9994952	8.354301	8.3585203811e-07
min	10001281	2.254240	2.2539512688e-07
max	10000645	2.253662	2.2535166482e-07

Sample Run 3 : ./analyse_nums -10 1 100

n = 10, r1 = 1, r2 = 100, Memory used = 0.000044 Mbytes
Numbers : 10 10 24 31 41 50 50 59 73 74 74 : min 10 : max 74
find 10 2 : delete 10 1 : find 10 1 : delete 10 1 : add 10 1 : find 10 1 : succ 10 24 : pred 10 -1
find 24 1 : delete 24 1 : find 24 0 : delete 24 0 : add 24 1 : find 24 1 : succ 24 31 : pred 24 10
find 31 1 : delete 31 1 : find 31 0 : delete 31 0 : add 31 1 : find 31 1 : succ 31 41 : pred 31 24
find 41 1 : delete 41 1 : find 41 0 : delete 41 0 : add 41 1 : find 41 1 : succ 41 50 : pred 41 31
find 50 2 : delete 50 1 : find 50 1 : delete 50 1 : add 50 1 : find 50 1 : succ 50 59 : pred 50 41
find 59 1 : delete 59 1 : find 59 0 : delete 59 0 : add 59 1 : find 59 1 : succ 59 73 : pred 59 50
find 73 1 : delete 73 1 : find 73 0 : delete 73 0 : add 73 1 : find 73 1 : succ 73 74 : pred 73 59
find 74 2 : delete 74 1 : find 74 1 : delete 74 1 : add 74 1 : find 74 1 : succ 74 -1 : pred 74 73
Numbers : 10 24 31 41 50 59 73 74 : min 10 : max 74
add 10 1 : find 10 2
add 50 1 : find 50 2
add 74 1 : find 74 2
Numbers : 10 10 24 31 41 50 50 59 73 74 74 : min 10 : max 74

Format is: <op name> <op input> <op result>