FIT2004: Lab questions for week 5

Objectives: This prac provides a platform for you to learn the formal concepts introduced during lectures in weeks 3& 4. Primarily, these concepts include partitioning problems, recursive sorting, divide and conquer and dynamic programming strategies. These programs are to be written in the python programming language.

1. During the lecture 3 we discussed the **three-way** Dutch-National-Flag partitioning problem. Write a program implementing this algorithm.

   Your program must be able to generate a random instance of this partitioning problem and solve it correctly. To generate the random instance, fix the size of the set (to be partitioned) to 1 million and populate it with 1 million *uniform* random integers [0,1,2].[1]

   Once this instance is partitioned, your program should automatically check if the partitioning your programming is generating is correct – that is all 0s, 1s and 2s have been correctly grouped in that order – or assert otherwise. When correct, your program should output the summary of partition sizes, i.e., the number of 0s, 1s and 2s along with their partition boundaries.

2. Now extend the above program to achieve a 4-way partitioning of a set of million random integers in the range [0,1,2,3].

3. If you were to generalize this Dutch National Flag approach to an arbitrary k-way partitioning problem, what would the time-complexity and space-complexity of this algorithm? (**This is NOT a computer-based question**, although eager students **should** attempt to write a generalized version.)

4. Write a Quick Sort program, using the 3-way Dutch-National Flag algorithm. Test your program on how many integers you can feasibly sort (in some practical amount of time) using your implementation.

   To do this, using the background information in the week 4 prac, generate a random permutation over N integers, and compute the time your program takes to run for increasing limits of N.

   Another exercise is to generate the numbers randomly (which may contain varying number of repeat elements) to create instances that are sorted using your program.

   Also test whether your algorithm runs faster or slower if you gave it a fully sorted list of numbers, and fully reversed list of numbers.

[1]Random number generation in Python, see `http://docs.python.org/2/library/random.html#random.randrange`

5. Write a program implmenting a dynamic programming solution to the matrix chain multiplication problem – See Lecture 4 slides. Using your program, find an optimal parenthesization of a matrix chain product, with matrix orders given by the sequence $\{5, 10, 3, 12, 5, 50, 6\}$.

6. Write a program implementing the Edit Distance and Alignment algorithm from Lecture 4. Your program should generate 2 random strings from a binary alphabet $\{A, B\}$. These strings should have a random length between 75 and 100. Using these random 2 strings as input, produce as output their edit distance and **alignment**.[2]

-=oOo=-
END
-=oOo=-

[2]An alignment between two string is derived by backtracking from the sink to the source of the dynamic programmig history matrix.