Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Laboratory 8

Title of the Laboratory Exercise: Recursion and Sorting

1. Introduction and Purpose of Experiment

   Students apply recursion and mathematical functions to solve problems using functional programming paradigm

2. Aim and Objectives

   Aim

   To apply recursion and mathematical functions to solve problems using functional programming paradigm

   Objectives

   At the end of this lab, the student will be able to

   - Apply recursion for solving problems
   - Solve problems mathematically and express solution using functional paradigm

3. Experimental Procedure

For the problems listed below, design the data structures, algorithm(s) and write the program(s). Tabulate the output for various inputs and verify against expected values. Compare the programming method in Haskell with C programming language. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome.

a. Create a program to do Quick Sort in Haskell

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Documentation:	
a. Procedure and Algorithm(s):	**Procedure:** 1. Creating a new Haskell File Haskell source files are ASCII based text file with a `.hs` extension. The execution of the Haskell Project usually begins from `Main.hs` and hence we name the file Main.hs, it can be anything if its going to a simple project, for the sake of organization the file that contains the driver function or main function is inside the Main.hs. 2. Design the functions required Since Haskell is a functional programming language, the required functions need to be decomposed into sub functions, the operations that needs to be done in the function also needs to be purely functional, and side-effects needs to be avoided as much as possible. 3. Documentation Write documentation for the functions that are used in the source file along with its input and outputs, write the function signatures to enforce typing in Haskell. 4. Running and testing The functions written can be individually tested in GHCI, which is an interactive version of the Glasgow Haskell Compiler, this will us know test atomic functions, which are a part of more functions, and make it easier to debug the programs, since there are no side-effects testing becomes easier. Load the `.hs` file into GHCI by typing `:l Main.hs` , this will compile and load the function written in the file and each of the function can be called specifically by just typing the name of the function, to see the type of function `:t` can be used. **Algorithms:**

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

	quickSort
	Params: a list of elements
	`Step 1: Start`
	`Step 2: Make the head element as pivot element`
	`Step 3: quick sort the elements less than pivot`
	`Step 4: quick sort the elements greater than pivot`
	`Step 5: return sorted_array = sorted less than pivot ++ pivot ++ sorted greater than pivot`
	`Step 6: Stop`

b. Conclusions :	Recursion is a powerful technique that can make most programs much easier to understand. It breaks down the problems into smaller subproblems, and the solutions of these subproblems gives the result of the larger problems. This technique has been learned and applied in the application of quick sort, which makes the code one of the shortest across many languages. HUnit testing can be used by software developers to test and debug Haskell code, since we have already done functional decomposition it is easy to debug such functions.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Results and Discussions:

Screenshot:

```
quicksort :: Ord t => [t] -> [t]

quicksort [] = []
quicksort (x:xs) =
    smallerthanx ++ [x] ++ largerthanx
        where smallerthanx = quicksort [a | a <- xs, a <=x]
              largerthanx  = quicksort [a | a <- xs, a > x]
```

OUTPUT :

```
*Main> quicksort [9,8,7,6,5,4,3,2,1]
[1,2,3,4,5,6,7,8,9]
*Main> quicksort [123, -123, 432, 745, 563, 1234]
[-123,123,432,563,745,1234]
*Main> quicksort "satyajit ghana"
" aaaaghijnstty"
*Main> quicksort ["satyajit", "ghana"]
["ghana","satyajit"]
*Main>
```

Discussion:

A sorted empty list is an empty list. Now here comes the main algorithm: a sorted list is a list that has all the values smaller than (or equal to) the head of the list in front (and those values are sorted), then comes the head of the list in the middle and then come all the values that are bigger than the head (they're also sorted). Let's take the example to sort a smaller list `[4, 3, 1, 5, 2]`, the algorithm will take the head and put it in the middle of the list, all number less than this head are place to its left and greater are to the right of this head. Hence the list becomes,

`[3,1, 2]++[4]++[5]`

Further,

`[1,2]++[3]++[4]++[5]`

`[1]++[2]++[3]++[4]++[5]`

`[1,2,3,4,5]`

And this is how the list is sorted in a recursive call of the quick sort function.