Faculty Of Computers and Artificial Intelligence

Computer Science Department

Programming Language 3 (PL3)

CS 313

SSO Library

**Index**

* Overview
* Sorting algorithms
* Searching algorithms
* Appendix

Overview

The main idea of the project is to implement a library with some of the sorting and searching algorithms using Haskell, and showing Haskell ability in optimizing complexity using monad and lazy evolution.

**Sorting Algorithms**

**insert'** :: Ord a => a -> [a] -> [a]

* Takes number and list as an Input and return list.
* Time Complexity : O(1)

**insertionSort** :: Ord a => [a] -> [a]

* Takes list as an Input and return list.
* Time Complexity : O(x2)

**shellSort** :: (Ord a) => [a] -> [a]

* Takes list as an Input and return list.
* Time Complexity : O(n\*log2 n ).
* Technique : improves on the **insertion** sort by breaking the original list into a number of smaller sublists.

**pbmerge** :: (Ord a, NFData a) => [a] -> [a]

* Takes list as an Input and return list.
* Time Complexity : Θ((log n) 3)
* Technique : Parallelism

**stoogeSort** :: (Ord a) => [a] -> [a]

* Takes list as an Input and return list.
* Time Complexity : O(nlog 3 / log 1.5 )
* Technique : recursive **sorting** algorithm

The **Heap type :**

data **Heap a**

* Heap is an abstract type representing Heap data Structure.

Operations:

HeapSort :: [a] -> [a]

* Takes list as an Input and return list.
* Time Complexity : O(n)
* Technique : Heap Data Structure

**beadSort** :: [Int] -> [Int]

* Takes list as an Input and return list.
* Time Complexity : O(n log n)
* Technique : Positive numbers are represented by a set of beads like those on an abacus.

bubbleSort :: Ord a => [a]->[a]

* Takes list as an Input and return list.
* Time Complexity : O(n2)
* Technique : repeatedly swapping the adjacent elements if they are in wrong order.

**cocktailSort** :: Ord a => [a]->[a]

* Takes list as an Input and return list.
* Time Complexity : O(n2)
* Technique : Cocktail Sort is a variation of bubble sort .but Cocktail Sort traverses through a given array in both directions alternatively.

**introSort** :: Ord a => [a]->[a]

* Takes list as an Input and return list.
* Time Complexity : O(n log n)
* Technique : a hybrid selection algorithm based on quick select (a variant of quicksort), which falls back median of medians.

**merge\_sort**::[Int]->[Int]

* Takes list as an Input and return list.
* Time Complexity : O(n log n)
* Technique : Divide and Conquer

**selection\_sort'**::[Int]->[Int]

* Takes list as an Input and return list.
* Time Complexity : O(n2).
* Technique : sorts an array by repeatedly finding the minimum element.

**gnome\_sort**::[Int]->[Int]

* Takes list as an Input and return list.
* Time Complexity : O(n2).
* Technique : Gnome Sort also called Stupid sort is based on the concept of a Garden Gnome sorting his flower pots.

**tree\_sort**::[Int]->[Int]

* Takes list as an Input and return list.
* Time Complexity : O(n2).
* Technique : Tree sort is a sorting algorithm that is based on Binary Search Tree data structure

**strandSort** :: Ord a => [a] -> [a]

* Takes list as an Input and return list.
* Time Complexity : O(n2).
* Technique : recursive **sorting** algorithm that **sorts** items of a list into increasing order.

**quicksort** :: Ord a => [a] -> [a]

* Takes list as an Input and return list.
* Time Complexity : O(n2).
* Technique : Divide and conquer.

**timesort** :: [Int]->[Int]

* Takes list as an Input and return list.
* Time Complexity : O(n log n).
* Technique : hybrid stable sorting algorithm, derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data.

**Searching Algorithms**

The **Vertex** type **:**

**data Vertex a**

* Vertex is abstract type representing **node** is the fundamental unit of which graphs are formed.

Operations:

**filterVertexNeighbors** :: [Vertex] -> [Vertex] -> [Vertex]

* Takes list of two vertices as an Input and return vertex.

**vertexLabel** ::Vertex -> [Char]

**vertexNeighbors** :: Vertex -> [[Char]]

**vertexDistance** ::Vertex -> Int

**vertexPredecessor** ::Vertex -> [Char]

The **Graph** type **:**

**data Graph a**

* Graph is abstract type is represented by an array of Vertex.

Operations:

**bfs** :: Graph -> Graph -> [Vertex] -> [Vertex] -> Graph

* Search for a specific vertex in a graph
* Time Complexity : O(n2).

**printGraph** :: Graph -> IO ()

* Take a Graph and print it In IO()

Other Algorithms:

**linearsearch** ::Eq a=> a -> [a] -> Bool

* Takes an Element and list as an Input and return Boolean.
* Time Complexity : O(log n).
* Technique : Search linearly in list.

**interSearch** ::[Int] -> Int -> Bool

* Takes an Element and list as an Input and return Boolean.
* Time Complexity : O(log(log n)).
* Technique : The Interpolation Search is an improvement over Binary Search for instances, where the values in a sorted array are uniformly distributed.

**binarySearch** :: Ord a => [a] -> a -> Bool

* Takes an Element and list as an Input and return Boolean.
* Time Complexity : O(log n)
* Technique : Search a sorted array by repeatedly dividing the search interval in half.

**jumpSearch** :: Ord a => [a] -> a -> Bool

* Takes an Element and list as an Input and return Boolean.
* Time Complexity : O(√n)
* Technique : check fewer elements (than linear search) by jumping ahead by fixed steps or skipping some elements in place of searching all elements.

**Appendix**

In this appendix I want to explain some points:

* We use monad and lazy evolution to do this algorithms as it a powerful technique in Haskell.
* We tried to optimitze this algoithms as we can , using Haskell Programming Language.
* Time Complexity isn’t measured accurately
* If you find any mistake In time complexity you can correct it in library repository in github : <https://github.com/Alymostafa/SSO-library>.
* Feel free to copy , edit this algorithms.
* Feel free to edit algorithms if you can make a time complexity better in its repository in github: <https://github.com/Alymostafa/SSO-library>.
* Feel free to Add new algorithms.
* Feel free to contact to me if you find any mistake or to understand anything in algorithms :   
   alimostfa2000@hotmail.com