CS 202 - Data Structures

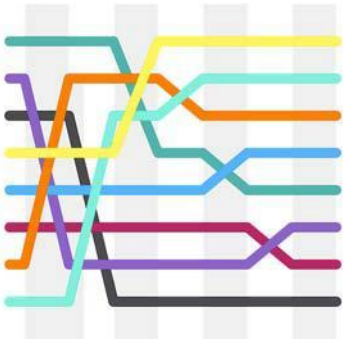
Assignment-4

Sorting Algorithms

Due Date: ***11pm on Sunday, 22nd April, 2018***

This assignment can be submitted till 11pm on Wednesday, 25th April, 2018 with a 10% penalty per day.

Your code must run on the Mars server!



In this assignment, you will implement and evaluate five sorting algorithms.

The course policy about plagiarism is as follows:

1. Students must not share actual program code with other students.
2. Students must be prepared to explain any program code they submit.
3. Students must indicate with their submission any assistance received.
4. All submissions are subject to plagiarism detection.
5. Students cannot copy code from the Internet.

Students are strongly advised that any act of plagiarism will be reported to the Disciplinary Committee.

Note:

This is a long assignment and may take more time than you expect, so start early. There would be no extensions.

This assignment consists of six major tasks, the first one involves implementing the heap data structure. Tasks 2-5 involve the implementation of different sorting algorithms while the last one pertains to the application of one of the sorting algorithms. For tasks 2 to 5, we have provided some code in generator.cpp, which generates a sequence of numbers between 1 and n. Based on user input, the sequence will be random, sorted, reverse sorted or almost sorted. For each task, you have to sort this input. You will be required to sort these numbers in arrays, as well as lists for some of the algorithms. Make sure that your algorithm works for negative values as well.

The header file sorts.h has declarations for all the sort functions. You must implement all of these functions in sorts.cpp. You may also define additional functions if it aids you in any way. But you must implement the functions already declared in sorts.h. The generator.cpp file contains functions that generate random input cases so that you can observe your implementation. You are also given a standard implementation of the Linked List data structure in the list.h and list.cpp files. You must write all your code in sorts.cpp. No other file should be altered. After writing your code in sorts.cpp, just compile generator.cpp and run. For final testing you are provided individual test cases for each sorting algorithm.

Every function that you must implement should accept an integer vector as input. The implementations must internally duplicate the vector into an array or linked list (as mandated by each task). Once the sorting is complete, the numbers must be put back into a vector in sorted order and that vector should be returned e.g., for Task 1, in InsertionSort(), you should take all elements present in the input vector, put them into an array and then apply the insertion sort algorithm. After sorting, put all the elements from the array into a vector (you can overwrite into the old vector or store in a new vector) and then return that.

Task 1 (Heap Data Structure):

In this task, you are expected to implement the heap data structure. You are provided the heap.h, heap.cpp and heap_test.cpp files. You are only required to make changes to the "heap.cpp" file and you are expected to implement all the methods in that file. You can test your implementation using the heap_test.cpp file. You will be using this implementation of heap data structure in task 5.

Task 2 (Insertion Sort – array based):

For this task, implement the in-place Insertion Sort algorithm using an array.

Task 3 (Mergesort – using linked lists):

In this task, you need to implement the Mergesort algorithm using a linked list

Task 4 (Quicksort – both array and linked lists):

Implement both the in-place array-based as well as the linked list method of Quicksort.

I. For the array-based implementation, use the following strategies for selecting a pivot:
   a. First element of the array as the pivot
   b. Then use the median-of-three pivot
   c. Finally, use the last element of the array as pivot

Use the strategy with lowest running time in your test.

II. For the linked list based implementation, use a random pivot.

Essentially this task includes two sub-tasks:

(i) Array based implementation – using the above mentioned techniques
(ii) Linked List implementation – using a random pivot.

Task 5 (Heapsort – using heaps):

For this task, implement the array based Heapsort algorithm using the heap implementation in task 1.

Task 6 (Smart Search):

You are given an unsorted vector that contains N numbers. You are also given a number k. You have to find all pairs of numbers in the vector, which add up to k. For example, given [3, 4, 1, 2, 5] and k=7, you would output:

2, 5
3, 4
4, 3
5, 2

The trick here, however, is that you need to do it in O(N log N) time. Zero marks would be given for doing this in $O(N^2)$ time. Also print the time taken to do this. Implement this task in "smartSearch.cpp".

Note: heap_test.cpp and smartSearch.cpp require no flags.

The following test cases require flags:
g++ test_insertion_sort.cpp -pthread –std=c++11
g++ test_merge_sort.cpp -pthread –std=c++11
g++ test_quickarray_sort.cpp -pthread –std=c++11
g++ test_quicklist_sort.cpp -pthread –std=c++11
g++ test_heap_sort.cpp -pthread –std=c++11

BEST OF LUCK