Data Structures and Algorithms – 2018

Assignment 4 – 25% of Continuous Assessment Mark

Deadline : 5pm Monday 12th March, via Canvas

Question 1 (12 marks)

Sort the array [5, 3, 4, 6, 8, 4, 1, 9, 7, 1, 2] using the Heapsort algorithm. Describe what you are doing and show the array as a Binary Tree at each stage.

Question 2 (16 marks)

Suppose you have an unbalanced binary search tree `t` containing `n` integers. Write a procedure `rebalanceBST(t,n)` that rebalances it by first outputting its contents to give a sorted array using the `fillArray(t,a,j)` algorithm from the Lecture Notes (Section 9.9), and then creating a perfectly balanced binary search tree by repeated calls of the primitive binary tree procedure `makeBT(v,l,r)` that returns a binary tree with root value `v`, left binary sub-tree `l` and right binary sub-tree `r`. [Hint: Consider writing and using a recursive procedure `balancedBST(a,i,j)` that deals with the sub-array of a sorted array `a` that starts at index `i` and ends at index `j`.]

Assuming `fillArray(t,a,j)` and `makeBT(v,l,r)` have both been implemented as efficiently as possible, what are the overall average-case and worst-case time complexities of your algorithm? Explain how you arrived at your answer,

Discuss whether this would be the most efficient approach for putting a whole new array `a` of `n` integers into a perfectly balanced binary search tree. If there is a better approach, explain what it is.

Question 3 (12 marks)

Outline, in no more than 100 words, the general Quicksort procedure for sorting an array. [Hint: This is the kind of "bookwork" question you can expect in the exam, though you will not normally be given a word limit. The answers are easily found in the Lecture Notes, but it is worth practicing writing clear and concise answers to questions like this about all the key topics in the module, without referring to any notes.]

Sort the array [3, 7, 2, 4, 9, 6, 8, 7, 5, 1, 6] using Quicksort with the pivot chosen to be the middle (rounded down) element of the array at each stage, and a partitioning algorithm that leads to a stable sort. Say how your partitioning algorithm works, and show the pivots and state of the array at each stage, i.e. the ordering and partitioning for each recursive call.

Question 4 (14 marks)

Explain, in no more than 150 words, when and why applying two phase Radix Sort to an array is able to produce a sorted array. Begin by stating any conditions that must be satisfied for it to work well.

A library has its books organized primarily according to 20 categories represented by the two digit codes 01, 02, 03, … 20, and secondarily according to the first two letters of the first author's surname Aa, Ab, …, Az, Ba, …, Zz. Use Radix Sort to sort the set of books with keys: [09 Ce, 09 Fa, 16 Mo, 16 Fa, 07 Ce, 13 Fa, 09 Mo, 07 Ba, 13 Ca]. Show the state of the book list and what is being done at each stage.

Question 5 (18 marks)

Suppose you have a hash table of size 19, the keys are words, and the hash map is defined as follows: Each letter is assigned a number according to its position in the alphabet, i.e.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

and the primary hash function is "*x* modulo 19", where *x* is the number corresponding to the first letter of the word. Why is this hash function not ideal?

Suppose instead you have a hash table of size 13 and the primary hash function is "*x* modulo 13", where *x* is the sum of the numbers corresponding to all the letters in the key word. Insert the following list of words into an initially empty hash table using linear probing:

`[computer, science, in, birmingham, dates, back, to, the, sixties]`

What is the load factor of the resulting table, and how many collisions occurred?

What is the effort (i.e. number of comparisons) involved in checking whether each of the following words are in the hash table: `teaching, research, admin`?

Show what the resulting hash table would look like if direct chaining had been used rather than linear probing.

Now what is the effort (i.e. number of comparisons, not including the processing of NULL pointers) involved in checking whether each of the following words are in the hash table: `teaching, research, admin`?

Question 6 (12 marks)

A call centre uses a hash table with open addressing to store customer orders taken during its working hours of 7am to 11pm every day of the year. Orders are kept in the hash table for 5 years and then deleted. It currently has about 5 million entries, and handles about 1 million transactions per year (i.e. between 2 and 3 a minute).

What size hash table would you suggest?

The performance of the hash table has degraded. Why might this have happened?

Suppose someone suggested copying the entries in the hash table into a new hash table, and it takes 2 milliseconds to copy one item. Would that be a sensible thing to do?

Question 7 (16 marks)

Given the efficiency of hash tables and their techniques for dealing with hash collisions, one might expect them to provide a more efficient method of detecting duplicate items than the algorithms in Questions 5 and 7 of Assignment 3. Outline in words how a hash table using direct chaining could be used to detect duplicates in a list of items.

Explain the average-case time complexity of this approach, and compare it with those of the three duplicate detection algorithms in Assignment 3. You may assume that the hash table has a reasonably low load factor and a sensible hash function, and you can use any results from the Lecture Notes without repeating the computations. [Hint: As before, consider the extreme cases of duplicates being very rare and duplicates being very common.]

Comment on the worst-case time complexity and how that compares to the algorithms in Assignment 3.