**CECS 302 Homework Assignment 4 (Due 4/24/23)**

**1) (30 pts)** This problem deals with **sorting algorithms**.

a. Explain the difference between the terms **Maintained Sorting** and **Sorting On Demand**. Provide at least one example of an application where each one is preferable.

Maintained sorting ensures a data structure is consistently sorted as elements are added and removed. Sorting on demand sort elements that are potentially out of order whenever they need to be. Essentially, maintain sorting keeps the order all times while on demand calls a sort only when needed. Maintained sorting would be preferable when searching is one of the most common operations, as it would allow really quick search time and could be used in data structures like a binary search tree. Sorting on demand would be good for a data structure that is more often added and deleted from then searched.

b. Explain the principal differences between the **Insertion Sort** and **Heap Sort** algorithms. Be certain you explain i. any requirements the algorithms have to be used, ii. a general idea of how the algorithms operate, and iii. which algorithm is the most efficient in the **worst-case scenario**. Your response should be 1-2 paragraphs in length.

Insertion sort is an algorithm that iterates through an array of numbers and rearranges and sorts them as though they were a set of cards in your hand. Using a for loop, It checks to see if the current index is less or more than the next index. If a swap needs to be made it will rearrange the order and check again to see if it needs to be rearranged again. Insertion sort is a very simple algorithm, but it is also a very inefficient one. In the best-case scenario, you would at least have to iterate though every single element, bringing the efficiency O(n) and in the worst case, you’d be dealing with O(n^2). Insertion sort is only practical for small data sets.

Heap sort is a sorting method that utilizes the properties of the heap abstract data type to quickly order a list of elements. It operates by building a heap from an unordered list, using the heapify() function to maintain the order property of the heap (the value of any node in the tree is less than or equal to the values of its children) then removing the top element of the heap and placing the element in an array. Before taking the next root, it calls the reheapdown() function and prepares the next element. Because the sort utilizes reheapdown() the sort can be done entirely in-place and has a worst-case efficiency of O(n log n).

**2) (30 pts)** This problem deals with **sets** and **set implementations.**

a. Explain the principal differences between the **Unsorted List ADT** and the **Set ADT**.

A Set is an abstract data type that contains an unordered collection of items with no duplicate values. An unsorted list ADT can have duplicate values. Storing an item in a set that is already there does not change the set at all, and deleting an item that is not in the set also does not affect anything and does not usually display an error.

b. Provide an example when using the bit-vector implementation of a set can be advantageous, and another example when it is a poor choice.

Bit vector sets can be advantageous with known finite sets. It would be advantageous for tracking whether or not the set has or does not have all possible values. A bit vector implementation would be a poor choice when dealing with a collection of elements that cannot be fully enumerated (infinite sets) or sets that are very large in nature.

**3) (40 pts)** This problem deals with **hash tables** and **hashing functions.** Your responses to each sub-part should be 2-5 sentences in length.

a. Explain the principal efficiency advantage of hash tables in comparison to other data structures we have observed thus far.

Hash tables are data structures that store keys and records after a hash function is applied. What makes hash tables so efficient and effective is the ability to store a list of elements that can be quickly accessed using the key created by the hash function. This can potentially make the search time of the hash table O(1) as long as there are no collisions.

b. Explain what issues can hamper this performance advantage, and what steps are taken to mitigate the slow-down.

Hash-tables can have collisions, which is where a hash function creates the same key for two different values. Too many collisions can seriously slow down a hash tables efficiency. One of the methods for resolving collisions is Linear probing. This method solves collisions by searching the next available space in the hash table and operating there. Hash functions also need to be created in a way that minimized clustering or areas in hash tables where large numbers if key accumulate and cause collisions.

c. Suppose employees at a company are assigned an 8-digit employee ID according to the following criteria:

i. The first four digits correspond to a department code (ex: the accounting department may have code 0051, the IT department code 3505, etc.)  
ii. The second four digits correspond the order in which the employee was hired (ex: the second employee would have the last four digits “0002,” the one-hundredth “0100,” etc.)  
Now suppose you are given the task to store employee information according to employee ID in a small hash table and are given the following options for the hashing function:   
i. Compute the hash index using the **first** four digits of the employee ID, mod 53.

First 4:

Employee 00510002 hash = 51

Employee 35050100 hash = 7

ii. Compute the hash index using the **last** four digits of the employee ID, mod 53.  
Assuming the company has 100+ employees and 5 departments, explain which approach is better for this dataset, and why. Should your selection change if option ii only used the last *two* digits of the ID, mod 53. Explain why or why not.

Last 4:

Employee 00510002 hash = 2

Employee 35050100 hash = 47

Between the two, it would be better to use the last 4 digits. This is because there would be a large amount of clustering with people in the same departments. Everyone in the IT department would have the same key, and so there would be collisions every time someone new is added. On the other hand, the last 4 should be unique to every employee. This means that there may be some clustering, but since 53 is a prime number, it should be low. If you used only the last two digits it would significantly reduce the effectiveness of hash function but it would still might be better than the first 4 digits depending on the amount of departments and how many employees are in those departments.

**Responses to all problems should be in .doc(x) or .pdf format. Please submit your assignment to Blackboard as a single file (not zipped, in this case) with the filename “LN\_FN\_4” where LN is your last name and FN is your first name.**