

---

**CS – 331**  
**Week 12**

---

**Readings:**

1. C++ Data Structures, Third Edition by Nell Dale Jones, Ch 10
2. Introduction to Algorithms, 2<sup>nd</sup> Edition, Thomas H. Cormen, Charles Leiserson, Ch 8, Ch 11

**Objectives:**

1. Get complete understanding of sorting algorithms through Linear-Time sorting
2. To understand hash table and creation of hash functions

**Concepts:**

1. Introduction to Linear-Time sorting
2. Types of Linear-Time Sorting
3. Understanding Hash tables
4. Studying different methods to create hash functions
5. Overview of Lab 12

**Outline:**

**1. Linear Time Sorting**

- a. Counting Sort
- b. Radix Sort
- c. Bucket Sort

**2. Introduction to hash table**

- a. Collision
- b. Collision Resolution by Chaining

**3. Methods for Creating Hash Function**

- a. The Division Method
- b. The Multiplication Method
- c. Universal Hashing

**4. Lab12**

**Reference:**

1. <http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/hashTable.htm>
2. <http://www.personal.kent.edu/~rmuhamma/Algorithms/Sorting/linearTimeIntro.htm>
3. Introduction to Algorithms, 2<sup>nd</sup> Edition, Thomas H. Cormen

## 1. Linear Time Sorting

**The Algorithms that follow linear- sorting run faster than  $O(n \log n)$  but they require special assumptions about the input sequence.**

### a. Counting Sort

- Counting sort assumes that each of the elements is an integer in the range 1 to  $k$ , for some integer  $k$ . When  $k = O(n)$ , the Count- sort runs in  $O(n)$  time.

### b. Radix Sort

- It is a small method that many people intuitively use when alphabetizing a large list of names. (Here Radix is 26, 26 letters of alphabet).
- On the first pass entire numbers sort on the least significant digit and combine in an array.
- Then on the second pass, the entire numbers are sorted again on the second least-significant digits and combine in an array and so on.

### c. Bucket Sort

- Runs in linear time on the average. It assumes that the input is generated by a random process that distributes elements uniformly over the interval  $[0, 1)$ .
- The idea of Bucket sort is to divide the interval  $[0, 1)$  into  $n$  equal-sized subintervals, or buckets, and then distribute the  $n$  input numbers into the buckets.

## 2. Introduction to hash table

When the size of the universal set  $U$  is much larger the size of the table would make it impractical to use direct access table. A solution is to map the keys onto a small range data structure called **hash table**.

### a. Collision

- As keys are inserted in the table, it is possible that two keys may hash to the same table slot.
- A hash function  $h$  maps the keys  $k$  and  $j$  to the same slot, so they collide.
- There are two basic methods for handling collisions in a hash table: Chaining and Open addressing

### b. Collision Resolution by Chaining

- When there is a collision (keys hash to the same slot), the incoming keys is stored in an overflow area
- The corresponding record is made to appear at the end of the linked list.
- Each slot  $T[j]$  contains a linked list of all the keys whose hash value is  $j$ . For example,  $h(k_1) = h(k_n)$  and  $h(k_5) = h(k_2) = h(k_7)$ .
- The worst case running time for insertion is  $O(1)$ .
- Deletion of an element  $x$  can be accomplished in  $O(1)$  time if the lists are doubly linked.

### 3. Methods for Creating Hash Function

#### a. The Division Method

- Map a key  $k$  into one of  $m$  slots by taking the remainder of  $k$  divided by  $m$ .
- That is, the hash function is  $h(k) = k \bmod m$ .
- Example : If table size  $m=12$  , key  $k = 100$  then  $h(100)= 100 \bmod 12 = 4$

#### b. The Multiplication Method

- It is a two step process.
- Step1: Multiply the key  $k$  by a constant  $0 < A < 1$  and extract the fraction part of  $kA$ .
- Step2: Multiply  $kA$  by  $m$  and take the floor of the result.
- The hash function using multiplication method is:  $h(k) = \lfloor m(kA \bmod 1) \rfloor$   
where " $kA \bmod 1$ " means the fractional part of  $kA$ , that is,  $kA - \lfloor kA \rfloor$ .

#### c. Universal Hashing

- Open Addressing
- Another way to deal with collisions.
- In this technique all elements are stored in the hash table itself. That is, each table entry contains either an element or NIL.
- When searching for element (or empty slot), we systematically examine slots until we find an element (or empty slot). There are no lists and no elements stored outside the table.
- Advantage of this technique is that it avoids pointers (pointers need space too). Instead of chasing pointers, we compute the sequence of slots to be examined.

### 4. Lab 12

- The purpose of this lab is to understand the Linear time Sorting through:
  - Implementation of Radix sort for lists.
  - Implementation of bucket sort
- Working with Hash tables:
  - Computing values through hash functions and then storing in hash tables.