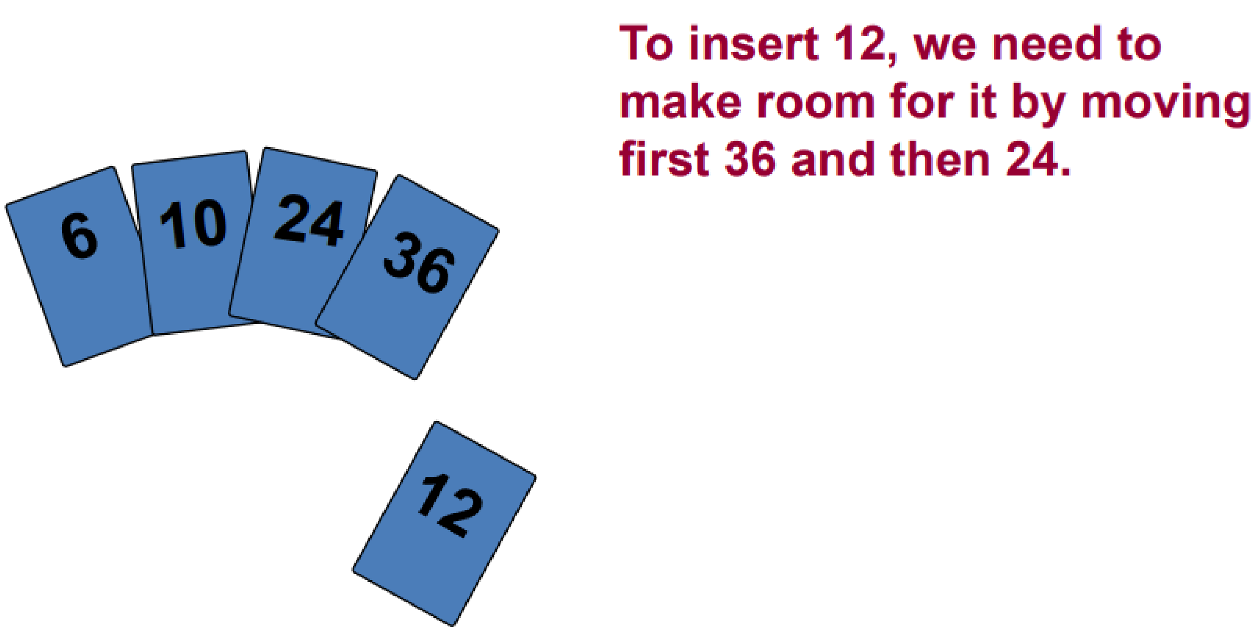
Outline

- Sorting
- Insertion Sort
- Selection Sort
- Merging
- Merge Sort
- Radix Sort
- Searching and Data Modification
- Hashing

Sorting

- Let A be a list of n element $A_1, A_2, A_3, \ldots, A_n$ in memory
- Sorting A refers to the operation of rearrange of the contents of A so that they are increasing in order (numerically or lexicographically), that is, $A_1 \leq A_2 \leq A_3 \leq \ldots \leq A_n$
- There are $n!$ ways that the contents can appear in A

Insertion Sort



Selection Sort

Idea

- Find the smallest element in the array
- Exchange it with the element in the first position
- Find the second smallest element and exchange it with the element in the second position
- Continue until the array is sorted

Searching and Data Modification

Let S be a collection of data

- Maintained in memory by a table using some type of data structure
- Sorted Array:
  - Searching complexity: $O(\log n)$ - Binary Search
  - Inserting and Deleting are very slow i.e., $O(n)$
  - Use: A great deal of searching but only very little data modification
- Linked List
  - Searching Complexity: $O(n)$ - Slow
  - Data Modification: Fast
  - Use: A great deal of data modification
- Binary Search Tree:
  - Average Searching Complexity: $O(\log n)$
  - Data Modification: relatively Fast (Since Linked List is used)
  - Drawback: Unbalanced tree

Hashing

- Searching time depends on the number n of elements in a collect S of data
- Hashing or Hash Addressing - independent of number of n
- Idea: Use key to determine the address of a record
- Drawback: waste of memory
- So we cannot use the key directly as the address of a record
- We need modification, we can write $H : K\ op\ L$
- The function H is called a hash function or hashing function.
- Drawback: H may not yield distinct values; it is possible that two different keys k_1 and k_2 will yield the same hash address. This situation is called collision
- Some method must be used to resolve it.

- Two principal criteria considered to select a hash function
  - H should be very easy and quick to compute
  - H should uniformly distribute the hash address throughout the set L so that there are a minimum number of collisions
  - One technique is to "chop" a key k into pieces and combine the pieces in some way to form the has address $H(k)$
- Division Method
  - Let total number of key is n
  - m is a prime number and $m > n$
  - The hash function $H(k) = k(\ mod\, m)$ or $H(k) = k(\ mod\, m) + 1$
- Midsquare Method
  - The key, k is squared
  - Then the has function, H is defined by $H(k) = l$
    - where l is obtained by deleting digits from both ends of k^2
- Folding Method
  - The key, k is partitioned into a number of parts, $k_1, k_2, \ldots, k_r$
  - Each part has the same number of digits (except the last)
  - Then the parts are added together, ignoring the last carry
  - $H(k) = k_1 + k_2 + \ldots + k_r$
  - Another process:
    - Sometimes for extra 'milling", the even-numbered parts are $k_2, k_4, \ldots$ each reversed before the addition

Collision Resolution

- Let new record R with key k
- $H(k) = l$, suppose the memory location l is already occupied
- This situation is called collision
- Load Factor: $\lambda = n/m$ (small value is better)
  - n = number of keys in K
  - m = the number of hash address in l
- Suppose we have 24 students and have a table with space for 365 records, Load factor = $\frac{24}{365}$ = 7%
- The efficiency of a hash function with a collision resolution
  - The average number of probes needed to find the location of the records with a given key k
- Two quantities
  - $S(\lambda)$ = average number of probes for a successful search

- $U(\lambda)$ = average number of probes for an unsuccessful search

This can be solved using either of the techniques

- Open Addressing
- Chaining

Open Addressing: Linear Probing and Modifications

- Suppose that a new record R with key k is to be added to the memory table T
- But $H(k)$ = h is already filled
- Resolve the collision:
  - assign R to the first available location follow $T[h]$
- Assume table T with m location is circular, so that $T[1]$ comes after $T[m]$
- Search for the record R in the table T by linearly searching location $T[h], T[h+1], T[h+2], \ldots$ until finding R (Successful) or meeting an empty location $(Unsuccessful Search)$

$$S(\lambda) = \frac{1}{2}\left(1 + \frac{1}{1-\lambda}\right)$$

$$U(\lambda) = \frac{1}{2}\left(1 + \frac{1}{(1-\lambda)^2}\right)$$

Quadratic Probing

- Disadvantage of linear probing: tend to cluster that is appear next to one another, the load factor is greater than 50%
- Instead of searching the location addressed, $h, h+1, h+2, \ldots$, we linearly search the location with addressed, $h, h+1, h+4, h+9, h+16, \ldots, h+i^2, \ldots$

Double Hashing

- Here a second hash function H' is used for resolving a collision, as follows
- Suppose a record R with key k has the hash address $H(k) = h$ and $H'(k) = h' \neq m$
- Then we linearly search the location with addressed

$$h, h+h', h+2h', h+3h'$$

Chaining

- Maintain Two table

- Table T as before with addition field $LINK$ which is used so that all records in T with same hash addressed h may be linked together to form a linked list
- Hash address table $LIST$ which contains pointer to the linked list in T
- Suppose a new record R with key k is added to the file F
- Place R in the first available location in the table T
- Add R to the liked list with pointer $LIST[H(k)]$
- If the linked list of records are not sorted, then R is simply inserted at the begging of its linked list.
- Disadvantage: need $3m$ memory cell
- it may be more useful to use open addressing with a table with $3m$ locations, which has the load factor $\lambda \leq \frac{1}{3}$, than to use chaining to resolve collision.