Data Structure

SEARCHING & SORTING TECHNIQUES

UNIT - 4

ANKIT VERMA

[WWW.ANKITVERMA.IN](http://www.ankitverma.in)
[WWW.ANKITVERMA.CO.IN](http://www.ankitverma.co.in)

DEPARTMENT OF INFORMATION TECHNOLOGY

Which Book To Follow ?

- Text Book
  - Data Structures, Schaum's Outlines, Seymour Lipschutz
- Reference Book
  - Data Structure Using C, Udit Aggarwal

Sorting

Sorting

- Sorting
  - Sorting refers to the Operation of Arranging Data in some given Order, such as Increasing or Decreasing, with Numerical Data, or Alphabetically, with Character Data
- Sorting Definition
  - Let A be a List of n Elements $A_1, A_2, \ldots, A_n$ in Memory. Sorting A refers to Operation of Rearranging the Contents of A so that they are Increasing in Order (Numerically or Lexicographically), that is, so that $A_1 <= A_2 <= \ldots <= A_n$

Sorting Techniques

- Sorting Algorithms / Techniques
  - Insertion Sort
  - Selection Sort
  - Merge Sort
  - Bubble Sort
  - Quick Sort
  - Heap Sort
  - Radix Sort

Bubble Sort

Bubble Sort

- Sort the following Numbers using Bubble Sort:
  - 32, 51, 27, 85, 66, 23, 13, 57
    - Pass 1:

32	51	27	85	66	23	13	57
32	27	51	85	66	23	13	57
32	27	51	85	66	23	13	57
32	27	51	66	85	23	13	57
32	27	51	66	23	85	13	57
32	27	51	66	23	13	85	57
32	27	51	66	23	13	57	85

Largest

Bubble Sort

- Pass 2:

27	32	51	66	23	13	57	85
27	32	51	23	66	13	57	85
27	32	51	23	13	66	57	85
27	32	51	23	13	57	66	85

Second Largest

Bubble Sort

- Pass 3:

27	32	23	51	13	57	66	85
27	32	23	13	51	57	66	85

- Pass 4:

27	23	32	13	51	57	66	85
27	23	13	32	51	57	66	85

Bubble Sort

- Pass 5:

23	27	13	32	51	57	66	85
23	13	27	32	51	57	66	85

- Pass 6:

13	23	27	32	51	57	66	85

- Pass 7:

13	23	27	32	51	57	66	85

Bubble Sort

- ALGORITHM: BUBBLE (A, N)
  - The Algorithm Sorts the Array A with N Elements

1. Repeat Step 2 and 3 for K = 1 to N – 1.
2. Set PTR := 1
3. Repeat while PTR <= N – K:

   (a) If A[PTR] > A[PTR + 1], then:

   Interchange A[PTR] and A[PTR + 1].

   (b) Set PTR := PTR + 1.
4. Exit

Insertion Sort

Insertion Sort

- Sort the following Numbers using Insertion Sort:
  - 77, 33, 44, 11, 88, 22, 66, 55

Pass	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1	$-\infty$	77	33	44	11	88	22	66	55
K=2	$-\infty$	77	33	44	11	88	22	66	55
K=3	$-\infty$	33	77	44	11	88	22	66	55
K=4	$-\infty$	33	44	77	11	88	22	66	55
K=5	$-\infty$	11	33	44	77	88	22	66	55
K=6	$-\infty$	11	33	44	77	88	22	66	55
K=7	$-\infty$	11	22	33	44	77	88	66	55
K=8	$-\infty$	11	22	33	44	66	77	88	55
Sorted	$-\infty$	11	22	33	44	55	66	77	88

Insertion Sort

- ALGORITHM: Insertion (A, N)
  - The Algorithm Sorts the Array A with N Elements

1. Set A[0] := $-\infty$
2. Repeat Step 3 to 5 for K = 2, 3, . . . , N:
3. Set TEMP := A[K] and PTR := K – 1
4. Repeat while TEMP < A[PTR]:
   (a) Set A[PTR + 1] := A[PTR]
   (b) Set PTR := PTR – 1
5. Set A[PTR + 1] := TEMP
6. Return

Selection Sort

Selection Sort

- Sort the following Numbers using Selection Sort:
  - 77, 33, 44, 11, 88, 22, 66, 55

Pass	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K=1, LOC=4	77	33	44	11	88	22	66	55
K=2, LOC=6	11	33	44	77	88	22	66	55
K=3, LOC=6	11	22	44	77	88	33	66	55
K=4, LOC=6	11	22	33	77	88	44	66	55
K=5, LOC=8	11	22	33	44	88	77	66	55
K=6, LOC=7	11	22	33	44	55	77	66	88
K=7, LOC=7	11	22	33	44	55	66	77	88
Sorted	11	22	33	44	55	66	77	88

Selection Sort

- ALGORITHM: MIN (A, K, N, LOC)
  - Array A is in Memory. This procedure finds the location LOC of smallest element.
    1. Set MIN := A[K] and LOC := K.
    2. Repeat for J = K + 1, K +2, . . . , N:

       If MIN > A[J], then:

       Set MIN := A[J] and LOC := J
    3. Return

Selection Sort

- ALGORITHM: SELECTION (A, N)
  - Algorithm sorts array A with N elements.
    1. Repeat Steps 2 and 3 for K = 1, 2, . . . , N – 1:
    2. Call MIN(A, K, N, LOC).
    3. Set TEMP := A[K], A[K] := A[LOC] and A[LOC] := TEMP.
    4. Exit

Merge Sort

Merge Sort

- Sort the following Numbers using Merge Sort:
  - 66, 33, 40, 22, 55, 88, 60, 11, 80, 20, 50, 44, 77, 30

Merge Sort

- ALGORITHM: MERGE (A, R, B, S, C)
  - Let A and B be sorted arrays with R and S elements respectively. Algorithm merges A and B into an array C with N = R + S elements

1. Set NA := 1, NB := 1 and PTR :=1.
2. Repeat while NA <=R and NB <= S:

    If A[NA] < B[NB], then:

    a) Set C[PTR] := A[NA]

    b) Set PTR := PTR + 1 and NA := NA + 1

    Else:

    a) Set C[PTR] := B[NB]

    b) Set PTR := PTR + 1 and NB := NB + 1
3. If NA > R, then:

    Repeat for K = 0, 1, 2, . . . , S – NB:

    Set C[PTR +K] := B[NB +K]

    Else:

    Repeat for K = 0, 1, 2, . . . , R – NA:

    Set C[PTR +K] := A[NA + K]
4. Exit

Searching

Searching

- Searching
  - Searching refers to Operation of Finding the Location of a given Item in Collection of Data
- Searching Algorithm / Techniques
  - Linear Search
  - Binary Search

Linear Search

Linear Search

- ALGORITHM: LINEAR (A, N, ITEM, LOC)
  - A is linear array with N elements. Algorithm finds location LOC of ITEM in A, or sets LOC := 0 if search is unsuccessful.

1. Set A[N + 1] := ITEM.
2. Set LOC := 1.
3. Repeat while A[LOC] != ITEM:

   Set LOC := LOC + 1.
4. If LOC = N + 1, then Set LOC := 0
5. Exit

Binary Search

Binary Search

- Search Item 40 from following Sorted Data using Binary Search:
  - 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99

Binary Search

- ALGORITHM: BINARY (A, LB, UB, ITEM, LOC)
  - A is sorted array with lower bound LB and upper bound UB. Beginning BEG, end END and middle MID are the location of A. Algorithm finds location LOC of ITEM in A, or sets LOC := NULL if search is unsuccessful.

1. Set BEG := LB, END := UB and MID = INT((BEG + END) / 2)
2. Repeat Step 3 and 4 while BEG <= END and A[MID] != ITEM
3. If ITEM < A[MID], then:
   Set END := MID – 1
   Else:
   Set BEG := MID + 1
4. Set MID := INT((BEG + END) / 2)
5. If A[MID] = ITEM, then:
   Set LOC := MID.
   Else:
   Set LOC := NULL
6. Exit

Hashing

Hashing

- Hashing
  - Search Time of each Algorithm depends on Number n of Elements in the Collection S of Data
  - Hashing is Searching Technique, which is independent of Number n
  - Hashing is also called Hash Addressing
  - Hashing will be Oriented towards File Management:
    - Assume File F of n Records with a Set K of Keys which uniquely determine the records in F
    - Assume F is maintained in Memory by a Table T of m Memory Locations and L is Set of Memory Addresses of Locations in T
    - Assume Keys in K and Addresses in L are Integers

Hashing

- Key is used to determine Address of Record, but Space is not Wasted
- **Hash Function H** from Set K of Keys into Set L of Memory Address:
  - H: K → L

- Collision
  - Hash Function H may not yield Distinct Values, It is possible that two different Keys K1 and K2 will yield Same Hash Address, called Collision
- Hashing is Divided into Two Parts:
  - Hash Function
  - Collision Resolution

Hash Functions

- Hash Functions
  - Two principal criteria used in selecting Hash Function H: K → L are as follows:
    - Function should be very Easy and Quick to Compare
    - As far as possible, Function H should uniformly distribute the Hash Addresses through the Set L, so that there are Minimum Number of Collisions
      - As there is No Guarantee of Second Condition, so General Techniques do Help:
        - “Chop” a Key K into Pieces and Combine the Pieces in some way to form Hash Address H(K).
        - Term “Hashing” comes from technique of “Chopping” a Key into Pieces

Hash Functions

- Popular Hash Functions:
  - Division Method
    - Choose number m Larger than number n of Keys in K
    - To Minimize Number of Collisions, m should be Prime Number or Number without Small Divisors
    - Hash Function H is defined by:
      - $H(K) = K \pmod{m}$ OR $H(K) = K \pmod{m} + 1$
    - $K \pmod{m}$ denotes Remainder when K is divided by m
    - Second Formula used when Hash Addresses Range from 1 to m, rather than 0 to $m - 1$

Hash Functions

- Midsquare Method
  - Key is Squared
  - Hash Function H is defined by
    - $H(K) = l$
  - Where l is obtained by Deleting Digits from both Ends of K^2
  - Same Positions of K^2 must be used for all of the Key

Hash Functions

- Folding Method
  - Key is Partitioned into number of Parts $K_1, K_2, \ldots, K_n$
  - Each Part except possibly the Last has the same number of Digits as the required Address
  - Parts are Added together, Ignoring the Last Carry.
  - Hash Function is defined by:
    - $H(K) = K_1 + K_2 + \ldots + K_r$
  - Sometimes, for Extra "Milling", Even Number Parts $K_2, K_4, \ldots$ Are each reversed before Addition

Hash Functions

- Example
  - Consider a Company, each of whose 68 Employees assigned a unique 4-digit Employee Number. Suppose L consists of 100 two-digit Addresses: 00, 01, 02, . . . , 99. Apply Hash Functions to each of following Employee Numbers:

    3205, 7148, 2345

Hash Functions

- Division Method
  - Choose Primary Number m close to 99, such as m = 97.
  - Divide the Employee Numbers with m and find Remainders

    $H(3205) = 4,$

    $H(7148) = 67,$

    $H(2345) = 17$

  - If Memory Address begin with 01 rather than 00, then we choose $H(K) = K(\bmod m) + 1$ to obtain:

    $H(3205) = 4 + 1 = 5,$

    $H(7148) = 67 + 1 = 68,$

    $H(2345) = 17 + 1 = 18$

Hash Functions

- Midsquare Method
  - Square the Key and choose Fourth & Fifth Digits for Hash Addresses:

K:	3025	7148	2345
K2:	10272025	51093904	05499025
H(K):	72	93	99

Hash Functions

- Folding Method
  - Chopping the Key K into Two Parts and Adding Yields the following Hash Addresses:

    H(3205) = 32+05 = 37,

    H(7148) = 71+48 = 119,

    H(2345) = 23+45=68

  - Alternatively, one may want to reverse the Second Part before Adding, thus produce Hash Addresses:

    H(3205) = 32+50 = 82,

    H(7148) = 71+84= 155,

    H(2345) = 23+54 = 77

Collision Resolution

- Collision
  - If we want to add new Record R with Key K to our File F, but if Memory Location Address H(K) is already occupied, such situation is called Collision
- Collision Resolution
  - Load Factor
    - Load Factor is ratio of number n of Keys in K to number m of Hash Addresses in L
    - Load Factor = $\lambda = \frac{n}{m}$

Collision Resolution

- Efficiency of Hash Function with Collision Resolution Procedure is measured by Average Number of **Probes (Key Comparison)** needed to Find Location of Record with given Key K
- Efficiency depends mainly on Load Factor λ
  - $S(\lambda)$ = Average Number of Probes for Successful Search
  - $U(\lambda)$ = Average Number of Probes for Unsuccessful Search

Collision Resolution

- Linear Probing
  - Suppose New Record R with Key K is added to Memory Table T, but Memory Location with Hash Address H(K) is already filled.
  - To Resolve Collision, Assign R to First available Location following T[h].
  - Record R will be Searched in Table T by Linear Search T[h], T[h+1], T[h+2], . . . Until finding R or meeting Empty Location.
  - Average Number of Probes for Successful Search:

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

  - Average Number of Probes for Unsuccessful Search:

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

Collision Resolution

- Disadvantage of Linear Probing
  - Record tend to Cluster and they Appear Next to One Another, when Load Factor is Greater than 50%
  - Increase Average Time to Search for Record
- Two Techniques of Minimizing Clustering:
  - Quadratic Probing
  - Double Hashing

Collision Resolution

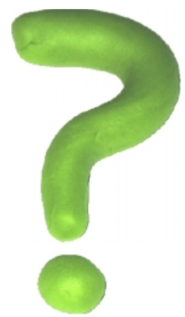
- Chaining
  - Maintain Two Tables in Memory
    - First, Table T contain Records in F, additional LINK field so all records in T have Same Hash Address h linked together to form Linked List
    - Second, Hash Address Table LIST contain Pointers to Linked List in T

Complexity of Algorithms

- Complexity of Algorithms

Algorithm	Worst Case	Average Case
Insertion Sort	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$
Bubble Sort	$O(n^2)$	$O(n^2)$
Quick Sort	$O(n^2)$	$O(n \log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$

THANKYOU



DOUBTS

PRESENTATION BY:
ANKIT VERMA
(IT DEPARTMENT)