Searching and Sorting II

INTRODUCTION TO SORTING

- **Sorting means arranging the elements of an array so that they are placed in some relevant order which may be either ascending or descending.**

- **For example, if we have an array that is declared and initialized as**
- **int A[] = {21, 34, 11, 9, 1, 0, 22};**
- **Then the sorted array (ascending order) can be given as:**
- **A[] = {0, 1, 9, 11, 21, 22, 34};**

- A sorting algorithm is defined as an algorithm that puts the elements of a list in a certain order, which can be either numerical order, lexicographical order, or any user-defined order
- <u>There are two types of sorting:</u>
- Internal sorting- which deals with sorting the data stored in the computer's memory
- External sorting- which deals with sorting the data stored in files. External sorting is applied when there is voluminous data that cannot be stored in the memory

Sorting on Multiple Keys

- For example, in a big organization we may want to sort a list of employees on the basis of their departments first and then according to their names in alphabetical order.
- Other examples of sorting on multiple keys can be
  - Telephone directories in which names are sorted by location, category (business or residential), and then in an alphabetical order.
  - In a library, the information about books can be sorted alphabetically based on titles and then by authors' names.
  - Customers' addresses can be sorted based on the name of the city and then the street.

- Data records can be sorted based on a property. Such a component or property is called a sort key.
- A sort key can be defined using two or more sort keys. In such a case, the first key is called the primary sort key, the second is known as the secondary sort key, etc

- **Consider the data records given below:**

Name	Department	Salary	Phone Number
Janak	Telecommunications	1000000	9812345678
Raj	Computer Science	890000	9910023456
Aditya	Electronics	900000	7838987654
Huma	Telecommunications	1100000	9654123456
Divya	Computer Science	750000	9350123455

- **Now if we take department as the primary key and name as the secondary key, then the sorted order of records can be given as:**

Name	Department	Salary	Phone Number
Divya	Computer Science	750000	9350123455
Raj	Computer Science	890000	9910023456
Aditya	Electronics	900000	7838987654
Huma	Telecommunications	1100000	9654123456
Janak	Telecommunications	1000000	9812345678

Practical Considerations for Internal Sorting

- When analyzing the performance of different sorting algorithms, the practical considerations would be the following:
  - Number of sort key comparisons that will be performed
  - Number of times the records in the list will be moved Best case performance
  - Worst case performance
  - Average case performance
  - Stability of the sorting algorithm where stability means that equivalent elements or records retain their relative positions even after sorting is done

Bubble Sort

- Bubble sort is a very simple method that sorts the array elements by repeatedly moving the largest element to the highest index position of the array segment (in case of arranging elements in ascending order)
- This procedure of sorting is called bubble sorting because elements ‘bubble’ to the top of the list. Note that at the end of the first pass, the largest element in the list will be placed at its proper position (i.e., at the end of the list).

Technique

- **The basic methodology of the working of bubble sort is given as follows:**

(a) In Pass 1, `A[0]` and `A[1]` are compared, then `A[1]` is compared with `A[2]`, `A[2]` is compared with `A[3]`, and so on. Finally, `A[N-2]` is compared with `A[N-1]`. Pass 1 involves `n-1` comparisons and places the biggest element at the highest index of the array.

(b) In Pass 2, `A[0]` and `A[1]` are compared, then `A[1]` is compared with `A[2]`, `A[2]` is compared with `A[3]`, and so on. Finally, `A[N-3]` is compared with `A[N-2]`. Pass 2 involves `n-2` comparisons and places the second biggest element at the second highest index of the array.

(c) In Pass 3, `A[0]` and `A[1]` are compared, then `A[1]` is compared with `A[2]`, `A[2]` is compared with `A[3]`, and so on. Finally, `A[N-4]` is compared with `A[N-3]`. Pass 3 involves `n-3` comparisons and places the third biggest element at the third highest index of the array.

(d) In Pass `n-1`, `A[0]` and `A[1]` are compared so that `A[0]<A[1]`. After this step, all the elements of the array are arranged in ascending order.

Algorithm for bubble sort

```
BUBBLE_SORT(A, N)

Step 1: Repeat Step 2 For 1 = 0 to N-1
Step 2:      Repeat For J = 0 to N - I
Step 3:                    IF A[J] > A[J + 1]
                           SWAP A[J] and A[J+1]
             [END OF INNER LOOP]
         [END OF OUTER LOOP]
Step 4: EXIT
```

Complexity of Bubble Sort

- **Therefore, to compute the complexity of bubble sort, we need to calculate the total number of comparisons. It can be given as:**

$$f(n) = (n - 1) + (n - 2) + (n - 3) + ... + 3 + 2 + 1$$
$$f(n) = n\,(n - 1)/2$$
$$f(n) = n^2/2 + O(n) = O(n^2)$$

- **Therefore, the complexity of bubble sort algorithm is $O(n^2)$. It means the time required to execute bubble sort is proportional to n^2 , where n is the total number of elements in the array**

NSERTION SORT

- Insertion sort is a very simple sorting algorithm in which the sorted array (or list) is built one element at a time.

- The main idea behind insertion sort is that it inserts each item into its proper place in the final list.

- Insertion sort is less efficient as compared to other more advanced algorithms such as quick sort, heap sort, and merge sort.

Technique

Insertion sort works as follows:

- **The array of values to be sorted is divided into two sets. One that stores sorted values and another that contains unsorted values.**
- **The sorting algorithm will proceed until there are elements in the unsorted set.**
- **Suppose there are n elements in the array. Initially, the element with index 0 (assuming LB = 0) is in the sorted set. Rest of the elements are in the unsorted set.**
- **The first element of the unsorted partition has array index 1 (if LB = 0).**
- **During each iteration of the algorithm, the first element in the unsorted set is picked up and inserted into the correct position in the sorted set.**

Algorithm for insertion sort

```
INSERTION-SORT (ARR, N)

Step 1: Repeat Steps 2 to 5 for K = 1 to N - 1
Step 2:      SET TEMP = ARR[K]
Step 3:      SET J = K - 1
Step 4:      Repeat while TEMP <= ARR[J]
                    SET ARR[J + 1] = ARR[J]
                    SET J = J - 1
             [END OF INNER LOOP]
Step 5:      SET ARR[J + 1] = TEMP
         [END OF LOOP]
Step 6: EXIT
```

Complexity of Insertion Sort

- For insertion sort, the best case occurs when the array is already sorted. In this case, the running time of the algorithm has a linear running time (i.e., O(n)).

- Similarly, the worst case of the insertion sort algorithm occurs when the array is sorted in the reverse order
- In the worst case, the first element of the unsorted set has to be compared with almost every element in the sorted set.
- Furthermore, every iteration of the inner loop will have to shift the elements of the sorted set of the array before inserting the next element.
- Therefore, in the worst case, insertion sort has a quadratic running time (i.e., $O(n^2)$).

Advantages of Insertion Sort

- **The advantages of this sorting algorithm are as follows:**
  - **It is easy to implement and efficient to use on small sets of data.**
  - **It can be efficiently implemented on data sets that are already substantially sorted.**
  - **It performs better than algorithms like selection sort and bubble sort. Insertion sort algorithm is simpler than shell sort, with only a small trade-off in efficiency. It is over twice as fast as the bubble sort and almost 40 per cent faster than the selection sort. it requires less memory space (only O(1) of additional memory space).**
  - **It is said to be online, as it can sort a list as and when it receives new elements.**

SELECTION SORT

- **Selection sort is a sorting algorithm that has a quadratic running time complexity of O(n2), thereby making it inefficient to be used on large lists.**

- **Selection sort is generally used for sorting files with very large objects (records) and small keys.**

Technique

- **Selection sort works as follows:**
- **Therefore,**
  - **In Pass 1, find the position POS of the smallest value in the array and then swap ARR[POS] and ARR[0]. Thus, ARR[0] is sorted.**
  - **In Pass 2, find the position POS of the smallest value in sub-array of N–1 elements. Swap ARR[POS] with ARR[1]. Now, ARR[0] and ARR[1] is sorted.**
  - **In Pass N–1, find the position POS of the smaller of the elements ARR[N–2] and ARR[N–1]. Swap ARR[POS] and ARR[N–2] so that ARR[0], ARR[1], ..., ARR[N–1] is sorted.**

Sort the array given below using selection sort

39	9	81	45	90	27	72	18

PASS	POS	ARR[0]	ARR[1]	ARR[2]	ARR[3]	ARR[4]	ARR[5]	ARR[6]	ARR[7]
1	1	9	39	81	45	90	27	72	18
2	7	9	18	81	45	90	27	72	39
3	5	9	18	27	45	90	81	72	39
4	7	9	18	27	39	90	81	72	45
5	7	9	18	27	39	45	81	72	90
6	6	9	18	27	39	45	72	81	90
7	6	9	18	27	39	45	72	81	90

Algorithm for selection sort

```
SMALLEST (ARR, K, N, POS)

Step 1: [INITIALIZE] SET SMALL = ARR[K]
Step 2: [INITIALIZE] SET POS = K
Step 3: Repeat for J = K+1 to N-1
            IF SMALL > ARR[J]
                  SET SMALL = ARR[J]
                  SET POS = J
            [END OF IF]
        [END OF LOOP]
Step 4: RETURN POS
```

```
SELECTION SORT(ARR, N)

Step 1: Repeat Steps 2 and 3 for K = 1
        to N-1
Step 2:     CALL SMALLEST(ARR, K, N, POS)
Step 3:     SWAP A[K] with ARR[POS]
      [END OF LOOP]
Step 4: EXIT
```

Complexity of Selection Sort

- Selection sort is a sorting algorithm that is independent of the original order of elements in the array
- In Pass 1, selecting the element with the smallest value calls for scanning all n elements; thus, n–1 comparisons are required in the first pass.
- Then, the smallest value is swapped with the element in the first position. In Pass 2, selecting the second smallest value requires scanning the remaining n – 1 elements and so on.
- Therefore, (n – 1) + (n – 2) + ... + 2 + 1 = n(n – 1) / 2 = O(n2) comparisons

Advantages of Selection Sort

- It is simple and easy to implement.
- It can be used for small data sets.
- It is 60 per cent more efficient than bubble sort.

- However, in case of large data sets, the efficiency of selection sort drops as compared to insertion sort