BCAC 232: Data Structures

SORTING

- **Sorting** is process of arranging the elements in a particular order. The order may be ascending order or descending order. The advantage of sorting is effective data accessing.

- **Types of sorting**: There are 2 types of sorting.

1. Internal sorting
2. External sorting

- **Internal sorting**: If all the elements (records) to be sorted are in the main memory then such a sorting is called **internal sorting.**
- **External sorting:** If some of the elements (records) to be sorted are in the secondary storage or disk such a sorting is called **External sorting**.

Types of sorting techniques:

1. Bubble sort (Exchange sort or Sinking sort).
2. Selection sort.
3. Insertion sort.
4. Quick sort.
5. Merge sort.
6. Heap sort.
7. Shell sort
8. Radix sort.
9. Address calculation sort.

Efficiency of a sorting technique:

- How to select a sorting technique for a given set of elements?
- There are number of sorting techniques available to sort a given array of data items. Each sorting technique has its own advantages and disadvantages. Different techniques are useful in different applications.
- There are 3 most important factors are counted while selecting a sorting technique, which are.

1. **Coding time**: The amount of time invested in writing full length sorting program.
2. **Execution time (Time complicity):** The amounts of time required execute the sorting program. This normally frequency of execution of statements in a program i.e. number of times statements are executed.
3. **Memory requirement (Space complicity):** The amount of memory required to store the entire sorting program in main memory while execution.

Analysis of a sorting technique:

- Analysis of a sorting technique depends of three factors, which are code time, time complicity and space complicity. Among these 3 factors while analyzing a sorting technique we mainly concentrate more on the time complicity.

- The time complicity is amount of time required to execute the sorting program. Which is analyzed in terms of 3 cases

1. Best case
2. Worst case
3. Average case

BUBBLE SORT

- It is most popular sorting technique among all other techniques because is very simple to understand and implement. It is also called **exchange or sinking sort**
- **Working of Bubble Sort**
- The algorithm begins by comparing the element at the bottom of the array with next element. If the first element is grater the second element, then are swapped or exchanged.
- This process in then repeated for next two elements i.e. for second and third element. After n-1 comparisons the largest of all data items bubbles up to the top of the array.
- The first n-1 comparisons constitute first pass. During second pass number of comparison is one les than previous pass i.e. there are n-2 comparisons in the second pass. During second pass second largest element bubbles up to the last but one position.

- Consider following array A of elements.

A	A[0]	A[1]	A[2]	A[3]	A[4]
	30	10	5	20	15

Begin the sort by comparing first two elements

30	10	5	20	15	Compare A[0] and A[1]. Since 30>10, interchange
10	30	5	20	15	Compare A[1] and A[2]. Since 30>5, interchange
10	5	30	20	15	Compare A[2] and A[3]. Since 30>20, interchange
10	5	20	30	15	Compare A[3] and A[4]. Since 30>15, interchange
10	5	20	15	30	Largest element 30 has bubble up to last position

Pass 1

10	**5**	**20**	**15**	**30**

Compare A[0] and A[1]. Since 10>5, interchange

5	**10**	**20**	**15**	**30**

Compare A[1] and A[2]. Since 10<20, no interchange

Pass 2

5	**10**	**20**	**15**	**30**

Compare A[2] and A[3]. Since 20>15, interchange

5	**10**	**15**	**20**	**30**

Second largest element bubbles up to the position last but one.

5	**10**	**15**	**20**	**30**

Compare A[0] and A[1]. Since 5<10, no interchange

Pass 3

5	**10**	**15**	**20**	**30**

Compare A[1] and A[2]. Since 10<15, no interchange

5	**10**	**15**	**20**	**30**

Third largest element is in its right position.

5	**10**	**15**	**20**	**30**

Compare A[0] and A[2]. Since 5>10, no interchange

Pass 4

5	**10**	**15**	**20**	**30**

Final sorted array after n-1 passes.

Algorithm:

Algorithm: BUBBLE_SORT(A, n) This algorithm sort a given array A[n] using bubble sort technique. Variables I and J are used to index the array and temp is a temporary variable.

```
Step1: start
Step2: Input the array A[n]
Step3: [Compute the sorting]
         Repeat For I←0 to n-1
Step4: [Compare the adjacent elements]
         Repeat For J←0 to n-1-I
Step5: If (A[J]>A[J+1])
        [Interchange A[J] and A[J+1]]
        Temp←A[J]
        A[J]←A[J+1]
         A[J+1]←temp
         [End If]
        [End step3 for loop]
         [End step4 for loop]
Step6: [Display output]
         Repeat For I←0 to n-1
         Output A[I]
         [End for]
Step9: stop
```

Analysis of bubble sort:

- **Best case**: If the given array of elements is in the ascending order, the outer for loop will be executed n-1 times. The inner for loop and if statement will be executed n-1 times for the first iteration of the outer for loop, n-2 times for the second iteration of the outer for loop and so on . Only one time during the n-1th iteration of the outer for loop. The interchange part will not be executed even once.

- **Worst case:** : If the given array of elements is in reverse order, the outer for loop will be executed n-1 times. The inner for loop, if statement and interchange part will be executed n-1 times for the first iteration of the outer for loop, n-2 times for the second iteration of the outer for loop and so on. Only one time during the n-1th iteration of the outer for loop. Hence maximum number of comparisons and interchange operations.

- **Advantages:**

1. Simple to understand and implement.
2. Very straight forward.
3. Better than selection sort.

- **Disadvantages:**

1. It runs slowly and hence it is not efficient, because more efficient sorting techniques are available.
2. Even if array is sorted, n-1 comparisons are required.

THANK YOU