UNIT-II

BRUTE FORCE AND DIVIDE-AND-CONQUER

Brute Force – Computing an – String Matching – Closest-Pair and Convex-Hull Problems – Exhaustive Search – Travelling Salesman Problem – Knapsack Problem – Assignment problem. Divide and Conquer Methodology – Binary Search – Merge sort – Quick sort – Heap Sort – Multiplication of Large Integers – Closest-Pair and Convex – Hull Problems.

1. BRUTE FORCE

Brute force is a straightforward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts involved.

Selection Sort, Bubble Sort, Sequential Search, String Matching, Depth- First Search and Breadth-First Search, Closest-Pair and Convex-Hull Problems can be solved by Brute Force.

COMPUTING a^n:

1. Computing a^n : a * a * a * … * a (n times)
2. Computing n! : The n! can be computed as n*(n-1)* …*3*2*1
3. Multiplication of two matrices: C=A
4. Searching a key from list of elements (Sequential search)

Advantages:

1. Brute force is applicable to a very wide variety of problems.
2. It is very useful for solving small size instances of a problem, even though it is inefficient.
3. The brute-force approach yields reasonable algorithms of at least some practical value with no limitation on instance size for sorting, searching, and string matching.

Selection Sort

- First scan the entire given list to find its smallest element and exchange it with the first element, putting the smallest element in its final position in the sorted list.
- Then scan the list, starting with the second element, to find the smallest among the last n − 1 elements and exchange it with the second element, putting the second

smallest element in its final position in the sorted list.

Generally, on the ***I*** th pass through the list, which we number from 0 to n − 2, the algorithm searches for the smallest item among the last n−i elements and swaps it with A_i: $A_0 \le A_1 \le ... \le A_{i-1} | A_i, ..., A_{min}, ..., A_{n-1}$ *in their final positions | the last n – I elements*

After n − 1 passes, the list is sorted

ALGORITHM Selection Sort (A[0..n − 1])

```
//Sorts a given array by selection sort
//Input: An array A [0..n − 1] of orderable elements
//Output: Array A [0..n − 1] sorted in non-decreasing order
for i ← 0 to n − 2 do
        min ← i
        for j ← i + 1 to n − 1 do
                if A [j ]<A[min] min ← j
        swap A[i] and A[min]
```

89	45	68	90	29	34	**17**
17 |	45	68	90	**29**	34	89
17	29 |	68	90	45	**34**	89
17	29	34 |	90	**45**	68	89
17	29	34	45 |	90	**68**	89
17	29	34	45	68	| 90	**89**

17 29 34 45 68 89 |90

The sorting of list 89, 45, 68, 90, 29, 34, 17 is illustrated with the selection sort algorithm.

Theanalysisofselectionsortisstraightforward.Theinputsizeisgivenbythenumberof elements n; the basic operation is the key comparison[j]<[NIn]. The number of times it sexecuted depends only on the array size and is given by the following sum:

$$(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1)-(i+1)+1] = \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2}$$

Thus, selection sort is a $\Theta(n^2)$ algorithm on all inputs.

Note: The number of key swaps is only **Θ(n),** or, more precisely n − 1.

Bubble Sort

The bubble sorting algorithm is to compare adjacent elements of the list and exchange them if they are out of order. By doing it repeatedly, we end up "bubbling up" the largest element to the last position on the list. The next pass bubbles up the second largest element, and so on, until after n−1passes the list is sorted. Pass $i(0 \le i \le n-2)$ of bubble sort can be represented by the following: $A_0, \ldots, A_j \leftrightarrow A_{j+1}, \ldots A_{n-i-1} \mid A_{n-i} \le \ldots \le A_{n-1}$

ALGORITHM Bubble Sort (A[0..n − 1])

//Sorts a given array by bubble sort

//Input: An array A [0..n − 1] of orderable elements

//Output: Array A [0..n − 1] sorted in non-decreasing order

for $i \leftarrow 0$ **to** n − 2 **do**

  for $j \leftarrow 0$ **to** $n - 2 - i$ **do**

    if A [j + 1]≤A[j] swap A[j] and A[j + 1]

The action of the algorithm on the list 89, 45, 68, 90, 29, 34, 17 is illustrated as an example.

89	↔	45		68		90		29		34		17	
45		89	↔	68		90		29		34		17	
45		68		89	↔	90	↔	29		34		17	
45		68		89		29		90	↔	34		17	
45		68		89		29		34		90	↔	17	
45		68		89		29		34		17	|	90	
45	↔	68	↔	89	↔	29		34		17	|	90	
45		68		29		89	↔	34		17	|	90	
45		68		29		34		89	↔	17	|	90	
45		68		29		34		17	|	89		90	

etc.

The number of key comparisons for the bubble-sort version given above is the same for all arrays of size n; it is obtained by a sum that is almost identical to the sum for selection sort:

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-2} 1 = \sum_{i=0}^{n-2} [(n-2-i)-0+1] = \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)n}{2}$$

The number of key swaps, however, depends on the input. In the worst case of decreasing arrays, it is the same as the number of key comparisons.

$C_{worst}(n) \in \Theta(n^2)$