CS6402 DESIGN AND ANALYSIS OF ALGORITM

2mark Questions with Answer

UNIT-II

BRUTE FORCE AND DIVIDE-AND-CONQUER

1. Define Brute force approach?

Brute force is a straightforward approach to solving a problem, usually directly based on the problem's statement and definitions of the concepts involved. The brute force approach is one that is easiest to apply.

2. What are the advantages of brute force technique?

The various advantages of brute force technique are:

- Brute force applicable to a very wide variety of problems. It is used for many elementary but important algorithmic tasks
- For some important problems this approach yields reasonable algorithms of at least some practical value with no limitation on instance size
- The expense to design a more efficient algorithm may be unjustifiable if only a few instances of problems need to be solved and a brute force algorithm can solve those instances with acceptable speed
- Even if inefficient in general it can still be used for solving small-size instances of a Problem
- It can serve as a yardstick with which to judge more efficient alternatives for solving a problem

3. Define 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 Ai:
    - $A_0 \le A_1 \le \ldots \le A_{i-1} \mid A_i, \ldots, A_{min}, \ldots, A_{n-1}$
    - *in their final positions | the last n − i elements*
- After n − 1 passes, the list is sorted.

4. 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 − 1 passes the list is sorted. Pass i (0 ≤ i ≤ n -2) of bubble sort can be represented by the following: A0, . . . , Aj? ↔Aj+1, ...,An−i−1|An−i≤... ≤ An−1

5. Define Closest-Pair Problem

The closest-pair problem finds the two closest points in a set of n points. It is the simplest of a variety of problems in computational geometry that deals with proximity of points in the plane or higher-dimensional spaces.

6. Define convex set and convex hull

Convex Set

A set of points (finite or infinite) in the plane is called ***convex*** if for any two points *p* and *q* in the set, the entire line segment with the endpoints at *p* and *q* belongs to the set.

Convex hull

The ***convex hull*** of a set *S* of points is the smallest convex set containing *S*. (The smallest convex hull of *S* must be a subset of any convex set containing *S*.)

7. What are the exhaustive search

For discrete problems in which no efficient solution method is known, it might be necessary to test each possibility sequentially in order to determine if it is the solution. Such exhaustive examination of all possibilities is known as exhaustive search, complete search or direct search**.** Exhaustive search is simply a brute force approach to combinatorial problems (Minimization or maximization of optimization problems and constraint satisfaction problems).

Exhaustive Search is applied to the important problems like

- Traveling Salesman Problem
- Knapsack Problem
- Assignment Problem

8. What is meant by traveling salesman problem?

The traveling salesman problem (TSP) is one of the combinatorial problems. The problem asks to find the shortest tour through a given set of *n* cities that visits each city exactly once before returning to the city where it started.

9. Define knapsack problem.

Given *n* items of known weights *w*1*, w*2*, . . . , wn* and values *v*1*, v*2*, . . . , vn* and a knapsack of capacity *W*, find the most valuable subset of the items that fit into the knapsack.

Item	Weight	Value	Capacity
1	4	$40	W=10
2	7	$42	
3	5	$25	
4	1	$10	

Solution: item 1, 3, 4 values = $40+$25+$10=$75.

10. Define assignment problem.

There are n people who need to be assigned to execute n jobs, one person per job. (That is, each person is assigned to exactly one job and each job is assigned to exactly one person.) The cost that would accrue if the ith person is assigned to the jth job is a known quantity C[i, j] for each pair i,j = 1, 2, . . . , n. The problem is to find an assignment with the minimum total cost.

11. Explain divide and conquer algorithm.

What is divide and conquer strategy?

A divide and conquer algorithm works by recursively breaking down a problem into

two or more sub-problems of the same (or related) type (**divide**), until these become simple enough to be solved directly (**conquer**).
Divide-and-conquer algorithms work according to the following general plan:

1. A problem is divided into several subproblems of the same type, ideally of about equal size.
2. The subproblems are solved (typically recursively, though sometimes a different algorithm is employed, especially when subproblems become small enough).
3. If necessary, the solutions to the subproblems are combined to get a solution to the original problem.

Example: Merge sort, Quick sort, Binary search, Multiplication of Large Integers and Strassen's Matrix Multiplication.

12. Write algorithm for merge sort.

Mergesort is based on divide-and-conquer technique. It sorts a given array *A*[0..*n*−1] by dividing it into two halves A[0..[n/2]−1] and A[n/2]..*n*−1], sorting each of them recursively, and then merging the two smaller sorted arrays into a single sorted one.

```
ALGORITHM Mergesort(A[0..n − 1])
    //Sorts array A[0..n − 1] by recursive mergesort
    //Input: An array A[0..n − 1] of orderable elements
    //Output: Array A[0..n − 1] sorted in nondecreasing order
    if n > 1
        copy A[0..[n/2]− 1] to B[0..[n/2]− 1]
        copy A[n/2]..n − 1] to C[0..[n/2] − 1]
        Mergesort(B[0..[n/2] − 1])
        Mergesort(C[0..[n/2] − 1])
        Merge(B, C, A) //see below
```

13. Define quick sort.

Quicksort is the other important sorting algorithm that is based on the divide-and conquer approach. quicksort divides input elements according to their value. A partition is an arrangement of the array's elements so that all the elements to the left of some element *A*[*s*] are less than or equal to *A*[*s*], and all the elements to the right of *A*[*s*] are greater than or equal to it:

14. Differentiate quick sort and merge sort.

Sl. No.	Merge sort	Quick sort
1.	Merge sort is a stable sort.	Quick sort is not a stable sort. Pivot element plays major role in sorting.
2.	It requires extra space (Storage) for sorting.	It does not require extra space (Storage) for sorting.
3.	Merge sort is a not in-place sort.	Merge sort is in-place sort.
4.	Used as external sorting.	Used as internal sorting.
5.	Best to sort big volume of data.	Best for small instances of data.
6.	It is Slower than quick sort.	It is Better than merge sort

15. Define Strassen's matrix multiplication

The Strassen's Matrix Multiplication find the product *C* of two 2 × 2 matrices *A* and *B* with just seven multiplications as opposed to the eight required by the brute-force algorithm.

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

Where

$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$

$m_2 = (a_{10} + a_{11}) * b_{00}$

$m_3 = a_{00} * (b_{01} - b_{11})$

$m_4 = a_{11} * (b_{10} - b_{00})$

$m_5 = (a_{00} + a_{01}) * b_{11}$

$m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01})$

$m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11})$

7 multiplications and 12 additions and 6 subtractions.

16. Compare Strassen's algorithm and brute force algorithm for matrix multiplication.

Strassen's matrix multiplication algorithm

SI. No	**Strassen's matrix multiplication algorithm**	**brute force matri multiplication algorithm**
1.	7 multiplications	8 multiplications
2.	Algorithm and implementation are not simple.	Algorithm and implementation are simple
3.	Time complexity is $\Theta(n^{2.8})$.	Time complexity is $\Theta(n^3)$.
4.	Order of growth is better than brute force matrix multiplication algorithm.	Order of growth is not better than Strassen's matrix multiplication algorithm.
5.	It is faster than the standard matrix multiplication algorithm and is useful in practice for large matrices.	It is slower than Strassen's matrix multiplication algorithm.
6.	Algorithm design technique is divide and conquer.	Algorithm design technique is brute force.
7.	Algorithm works only for $n=2^k$, k=1,2,3,..	Algorithm works for all n.