Q.1 a. What is an algorithm? Give an example with illustration.

Answer:

For example, given the input sequence _31, 41, 59, 26, 41, 58_, a sorting algorithm returns as

An ***algorithm*** is any well-defined computational procedure that takes some value, or set of values, as ***input*** and produces some value, or set of values, as ***output***. An algorithm is thus a sequence of computational steps that transform the input into the output. For example, one might need to sort a sequence of numbers into nondecreasing order. This problem arises frequently in practice and provides fertile ground for introducing many standard design techniques and analysis tools. Here is how we formally define the ***sorting problem***:

Divide-and-Conquer is a top-down technique for designing algorithms that consist of dividing the problem into smaller sub-problems hoping that the solutions of the sub-problems are easier to find and then composing the partial solutions into the solution of the original problem.

Recurrance Equation:
M(n)=M(n-1)+1+M(N-1) for n>1
And M(1)=1

- **Input:** A sequence of n numbers □$a_1, a_2, \ldots, a_n$□.
- **Output:** A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \le a'_2 \le \cdots \le a'_n$.

output the sequence _26, 31, 41, 41, 58, 59_. Such an input sequence is called an ***instance*** of the sorting problem. In general, an ***instance of a problem*** consists of the input (satisfying whatever constraints are imposed in the problem statement) needed to compute a solution to the problem.

b. What is the difference between time complexity and space complexity?

Answer:

The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the size of the input to the problem. The time complexity of an algorithm is commonly expressed using big O notation, which suppresses multiplicative constants and lower order terms. When expressed this way, the time complexity is said to be described *asymptotically*, i.e., as the input size goes to infinity. For example, if the time required by an algorithm on all inputs of size n is at most $5n^3 + 3n$, the asymptotic time complexity is $O(n^3)$.

The space complexity of a program (for a given input) is the number of elementary objects that this program needs to store during its execution. This number is computed with respect to the size n of the input data. For an algorithm T and an input x, DSPACE(T, x) denotes the number of cells used during the (deterministic) computation T(x). We will note DSPACE(T) = O(f (n)) if DSPACE(T, x) = O(f (n)) with n = |x | (length of x).

c. Explain "Divide and Conquer Technique".

Answer:

Divide-and-Conquer paradigm consists of following major phases: Breaking the problem into several sub-problems that are similar to the original problem but smaller in size, Solve the sub-problem recursively (successively and independently), and then Combine these solutions to sub-problems to create a solution to the original problem.

d. Compute the time efficiency for Towers of Hanoi problem.

Answer:

Recurrance Equation:

M(n)=M(n-1)+1+M(N-1) for n>1

And M(1)=1

After Computation: M(n)=2n-1

NP-complete problem definition

e. What is the major difference between shortest path

Answer:
The differences between shortest path and longest path are:
• Shortest path has ***independent*** subproblems.
• Solution to one subproblem does not affect solution to another subproblem of the same problem.
• Longest simple path: subproblems are *not* independent.
• Consider subproblems of longest simple paths *q* _ *r* and *r* _ *t*.
• Longest simple path *q* _ *r* uses *s* and *t*.
• Cannot use *s* and *t* to solve longest simple path *r* _ *t*, since if we do, the path isn't simple.
• But we *have* to use *t* to Þnd longest simple path *r* _ *t*!
• Using resources (vertices) to solve one subproblem renders them unavailable to solve the other subproblem.

f. Describe the insertion sort.

Answer:
A good algorithm for sorting a small number of elements.
It works the way you might sort a hand of playing cards:
• Start with an empty left hand and the cards face down on the table.
• Then remove one card at a time from the table, and insert it into the correct position in the left hand.
• To find the correct position for a card, compare it with each of the cards already in the hand, from right to left.
• At all times, the cards held in the left hand are sorted, and these cards were originally the top cards of the pile on the table.

g. Define NP-complete and NP-hard problems **(7 × 4)**

Answer:
NP-complete problem definition
NP-hard problem definition

Q.2 a. Discuss the fundamental steps involved in the design and analysis of algorithm with a neat diagram. **(8)**

Answer:
Writing the flow chart diagram -
Explanation of the following steps:
Fundamentals of Algorithmic problem solving

i. Understanding the problem
ii. Ascertain the capabilities of the computational device
iii. Exact /approximate soln.
iv. Decide on the appropriate data structure
v. Algorithm design techniques
vi. Methods of specifying an algorithm
vii. Proving an algorithms correctness

Analysing an algorithm

b. Write an algorithm for merging two sorted arrays to a single array. Explain how you will use this algorithm for Merge Sort. **(10)**

Answer:

Q.3 a. Consider the following algorithm. (10)

```
Algorithm: CubeMe(n)
If n = 1 return 1
Else return CubeMe(n-1) + n*n*n
```

(i) What does this algorithm compute?
(ii) What is its basic operation? Justify your choice
(iii) How many times is the basic operation executed? Write down the recurrence relation. Justify
(iv) What is its initial condition? Justify.
(v) Derive the time efficiency class of this algorithm.
(vi) Name the efficiency class of the non-recursive version of the algorithm.
(vii) Between recursive & non-recursive versions, which is better? Justify

Answer:

I. The algorithm computes sum of cubes of natural numbers.

II. Basic operation could be either addition or multiplication. However, multiplication is considered, since it is computationally intensive.

III. $M(n) = M(n-1) + 2$ for $n > 1$. The multiplication operation happens twice in 'n*n*n'

IV. $M(n) = 0$ for $n = 1$. $M(1) = 0$. Multiplication is not executed when '$n = 1$'

V. Using backward substitution, the time efficiency class is derived as:
Generally: $M(n) = M(n-i) + i*2$
Applying Initial Condition: $M(n) = 2(n-1)$

VI. $M(n) \in \theta(n)$. Order of growth is linear.

VII. Better algorithm is the non-recursive version because the order of growth is the same between the two versions. The iterative algorithm avoids the overhead associated with the call-stack.

b. Explain with the help of an example how BFS differs from DFS. (8)

Answer:

Breadth-first search is a way to find all the vertices reachable from the a given source vertex, *s*. Like depth first search, BFS traverse a connected component of a given graph and defines a spanning tree. Intuitively, the basic idea of the breath-first search is this: send a wave out from source *s*. The wave hits all vertices 1 edge from *s*. From there, the wave hits all vertices 2 edges from *s*. Etc. We use FIFO queue Q to maintain the wave front: *v* is in Q if and only if wave has hit *v* but has not come out of *v* yet.

Depth-first search is a systematic way to find all the vertices reachable from a source vertex, s. historically, depth-first was first stated formally hundreds of years ago as a method for traversing mazes. Like breadth-first search, DFS traverse a connected component of a given graph and defines a spanning tree. The basic idea of depth-first search is this: It methodically explores every edge. We start over from different vertices as necessary. As soon as we discover a vertex, DFS starts exploring from it (unlike BFS, which puts a vertex on a queue so that it explores from it later).

Q.4 a. Design an algorithm to sort the given list of elements using Quick Sort incorporating divide and conquer technique. Sort the following list using the same and compute its best case time efficiency : 4, 2, 0, 8, 7, 1, 3, 6 (12)

Answer:

Writing algorithm

```
ALGORITHM Quicksort (A[ l …r ])
//sorts by quick sort
//i/p: A sub-array A[l..r] of A[0..n-1],defined by its left and right indices l and r
//o/p: The sub-array A[l..r], sorted in ascending order
if l < r
s _ Partition (A[l..r]) // s is a split position
Quicksort(A[l..s-1])
Quicksort(A[s+1..r]

ALGORITHM Partition (A[l ..r])
//Partitions a sub-array by using its first element as a pivot
//i/p: A sub-array A[l..r] of A[0..n-1], defined by its left and right indices l and r (l < r)
//o/p: A partition of A[l..r], with the split position returned as this function's value
p _A[l]
i _l
j _ r + 1;
Repeat
repeat i _ i + 1 until A[i] >=p //left-right scan
repeat j _j – 1 until A[j] < p //right-left scan
if (i < j) //need to continue with the scan
swap(A[i], a[j])
until i >= j //no need to scan
swap(A[l], A[j])
return j
```

Sorting the List

Efficieny computation:

Best case:

C(n) = 2C(n/2) + _(n) (2 sub-problems of size n/2 each)

C(n) = 2C(n/2) + _(n).

= _ (n1log n)

= _ (n log n)

b. For the input 30, 20, 56, 75, 31, 19 and hash function h(K) = K mod 11 (6)
(i) Construct the open hash table
(ii) Find the largest number of key comparisons in a successful search in this table.

Answer:

Ans. i

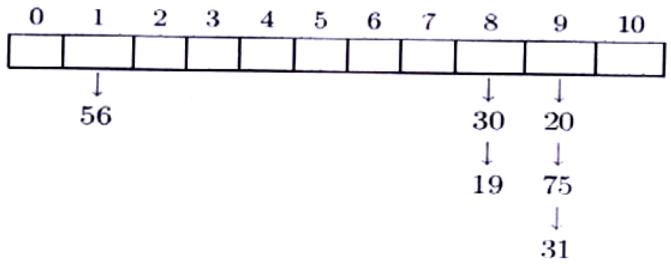
The list of keys: 30, 20, 56, 75, 31, 19

The hash function: $h(K) = K \bmod 11$

The hash addresses:

K	30	20	56	75	31	19
$h(K)$	8	9	1	9	9	8

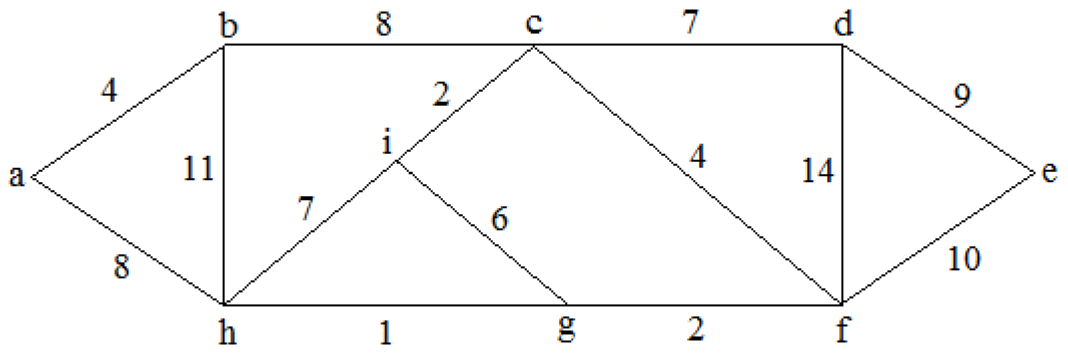
The open hash table:



ii.

b. The largest number of key comparisons in a successful search in thi: table will be 3 (in searching for $K = 31$).

Q.5 a. Write down Prim's Algorithm for finding the Minimum Spanning Tree of a connected graph. Execute your algorithm on the following graph. (10)



Answer:

b. Suppose that a graph G has a minimum spanning tree already computed, how quickly can we update the minimum spanning tree if we add a new vertex and incident edges to G. (8)

Answer:

Q.6 a. Write down Knuth Morris Pratt algorithm for string matching. Compute the prefix function for the pattrn b a c b a b a b a a b c b a b (12)

Answer:

b. What is the best case and worst case complexity of Naive String Matching algorithm (Simple text Searth). Give one example of Pattern and Text for both the cases. (6)

Answer:

Q.7 a. Derive a recursive algorithm for solving the Longest Common Subsequence (LCS) problem. Determine an LCS of < b a a b a b a b > and < a b a b b a b b a > (12)

Answer:

b. Define NP completeness. Prove that the circuit satisfiability problem is NP complete from the definition of NP completeness. (6)

Answer:

TEXT BOOK

I. Introduction to algorithms – Cormen and others, Prentice Hall of India, second edition, 2002
II. Algorithms – Johnsonbaugh and Schaefer , Pearson Education prentice Hall