TYPES OF ALGORITHMS

Solutions

1. Which of the following algorithms is NOT a divide & conquer algorithm by nature?

(a) Euclidean algorithm to compute the greatest common divisor
(b) Heap Sort
(c) Cooley-Tukey fast Fourier transform
(d) Quick Sort

Solution: Option (b)

2. Consider the following C program:

```
int main()
{
   int x, y, m, n;
   scanf ("%d %d", &x, &y);
  /* x > 0 and y > 0 */
  m = x; n = y;
  while (m != n)
   {
      if(m > n)
         m = m - n;
      else
         n = n - m;
   }
   printf("%d", n);
}
```

What does the program compute?

(a) x + y using repeated subtraction
(b) x mod y using repeated subtraction
(c) the greatest common divisor of x and y
(d) the least common multiple of x and y

Solution: Option (c)

3. Consider the polynomial $p(x) = a_0 + a_1x + a_2x^2 + a_3x^3$, where $a_i \neq 0$, for all i. The

minimum number of multiplications needed to evaluate p on an input x is:

(a) 3 (b) 4
(c) 6 (d) 9

Solution: Option (a)

Explanation:
Multiplications can be minimized using following order for evaluation of the given expression:
$p(x) = a_0 + x(a_1 + x(a_2 + a_3x))$

4. Maximum Subarray Sum problem is to find the subarray with maximum sum. For example, given an array {12, -13, -5, 25, -20, 30, 10}, the maximum subarray sum is 45.

The naive solution for this problem is to calculate sum of all subarrays starting with every element and return the maximum of all. We can solve this using Divide and Conquer, what will be the worst case time complexity using Divide and Conquer.

(a) O (n) (b) O (n log n)
(c) O (log n) (d) O (n^2)

Solution: (b)

5. Consider a situation where you don't have function to calculate power (pow( ) function in C) and you need to calculate x^n where x can be any number and n is a positive integer. What can be the best possible time complexity of your power function?

(a) O (n) (b) O (n log n)
(c) O (log log n) (d) O (log n)

Solution: Option (d)

Explanation:
We can calculate power using divide and conquer in O (log n) time.

6. Consider the problem of searching an element x in an array 'arr [ ]' of size n. The problem can be solved in O (log n) time if:

(1) Array is sorted
(2) Array is sorted and rotated by k. k is given to you and $k \leq n$
(3) Array is sorted and rotated by k. k is NOT given to you and $k \leq n$
(4) Array is not sorted

(a) 1 Only
(b) 1 & 2 only
(c) 1, 2 and 3 only
(d) 1, 2, 3 and 4

Solution: Option (c)

7. The secant method is used to find the root of an equation f(x) = 0. It is started from two distinct estimates x_a and x_b for the root. It is an iterative procedure involving linear interpolation to a root. The iteration stops if $f(x_b)$ is very small and then x_b is the solution. The procedure is given below. Observe that there is an expression which is missing and is marked by? Which is the suitable expression that is to be put in place of? So that it follows all steps of the secant method?

Secant:

Initialize: x_a, x_b, ε, N // ε = convergence indicator
$f_b = f(x_b)$ i = 0
while (i < N and $|f_b| > \varepsilon$) do
i = i + 1 // update counter
x_t = ? // missing expression for
// intermediate value
$x_a = x_b$ // reset x_a
$x_b = x_t$ // reset x_b
$f_b = f(x_b)$ // function value at new x_b
end while
if $|f_b| > \varepsilon$
then // loop is terminated with i = N
write "Non-convergence"
else
write "return x_b"
end if

(a) xb – (fb– f(xa)) fb/ (xb – xa)
(b) xa– (fa– f(xa)) fa/ (xb – xa)
(c) xb – (fb – xa) fb/ (xb – fb(xa)
(d) xa – (xb – xa) fa/ (fb – f(xa))

Solution: Option (d)

8. Suppose you are provided with the following function declaration in the C programming language:

int partition (int a[ ], int n);

The function treats the first element of a[ ] as a pivot, and rearranges the array so that all elements less than or equal to the pivot is in the left part of the array, and all elements greater than the pivot is in the right part. In addition, it moves the pivot so that the pivot is the last element of the left part. The return value is the number of elements in the left part. The following partially given function in the C programming language is used to find the kth smallest element in an array a[ ] of size n using the partition function. We assume $k \leq n$

```
int kth_smallest (int a[ ], int n, int k)
{
int left_end = partition (a, n);
  if (left_end+1==k)
  {
      return a [left_end];
  }
  if (left_end+1 > k)
  {
    return kth_smallest (_____________________);
  }
  else
 {
    return kth_smallest (_____________________);
 }
}
```

The missing argument lists are respectively:

(a) (a, left_end, k) and (a+left_end+1, n–left_end–1, k–left_end–1)
(b) (a, left_end, k) and (a, n–left_end–1, k–left_end–1)
(c) (a, left_end+1, N–left_end–1, K–left_end–1) and(a, left_end, k)
(d) (a, n–left_end–1, k–left_end–1) and (a, left_end, k)

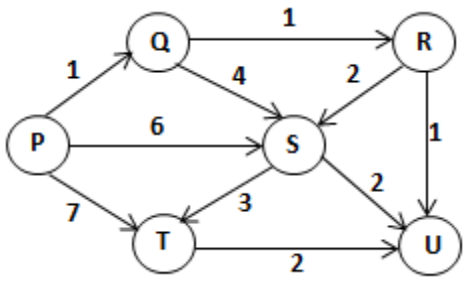
Solution: Option (a)

9. Which of the following standard algorithms is not a Greedy algorithm?

(a) Dijkstra's shortest path algorithm
(b) Prim's algorithm
(c) Kruskal algorithm
(d) Huffman Coding
(e) Bellmen Ford Shortest path algorithm

Solution: Option (e)

10. Suppose we run Dijkstra's single source shortest-path algorithm on the following edge weighted directed graph with vertex P as the source. In what order do the nodes get included into the set of vertices for which the shortest path distances are finalized? (GATE CS 2004)



(a) P, Q, R, S, T, U
(b) P, Q, R, U, S, T
(c) P, Q, R, U, T, S
(d) P, Q, T, R, U, S

Solution: Option (b)

11. A networking company uses a compression technique to encode the message before transmitting over the network. Suppose the message contains the following characters with their frequency:

Character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

If the compression technique used is Huffman Coding, how many bits will be saved in the message?

(a) 224
(b) 800
(c) 576
(d) 324

Solution: Option (c)

Explanation:
Total number of characters in the message = 100.
Each character takes 1 byte. So total number of bits needed = 800.

After *Huffman Coding*, the characters can be represented with:
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111

Total number of bits needed = 224
Hence, number of bits saved = 800 - 224 = 576

12. What is the time complexity of Huffman Coding?

(a) O (N)
(b) O (N log N)
(c) O (N (log N)^2)
(d) O (N^2)

Solution: Option (b)

Explanation:
O (n log n) where n is the number of unique characters. If there are n nodes, extractMin() is called 2*(n – 1) times. extractMin() takes O (log n) time as it called minHeapify().
So, overall complexity is O (n log n).

13. In question #12, which of the following represents the word "dead"?

(a) 1011111100101
(b) 0100000011010
(c) Both A and B
(d) None of these

Solution: Option (a)

15. Which of the following is true about Huffman Coding?

(a) Huffman coding may become lossy in some cases
(b) Huffman Codes may not be optimal lossless codes in some cases
(c) In Huffman coding, no code is prefix of any other code.
(d) All of the above

Solution: Option (c)

Explanation:

Huffman coding is a lossless data compression algorithm. The codes assigned to input characters are Prefix Codes, means the codes are assigned in such a way that the code assigned to one character is not prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding.

16. Suppose the letters a, b, c, d, e, f have probabilities 1/2, 1/4, 1/8, 1/16, 1/32, 1/32 respectively. Which of the following is the Huffman code for the letter a, b, c, d, e, f?

(a) 0, 10, 110, 1110, 11110, 11111
(b) 11, 10, 011, 010, 001, 000
(c) 11, 10, 01, 001, 0001, 0000
(d) 110, 100, 010, 000, 001, 111

Solution: Option (a)

Explanation:
We get the following Huffman Tree after applying Huffman Coding Algorithm.
The idea is to keep the least probable characters as low as possible by picking them first.

17. Suppose the letters a, b, c, d, e, f have probabilities 1/2, 1/4, 1/8, 1/16, 1/32, 1/32 respectively. What is the average length of Huffman codes?

(a) 3
(b) 2.1875
(c) 2.25
(d) 1.19375

Solution: Option (d)

18. An algorithm to find the length of the longest monotonically increasing sequence of numbers in an array A[0: n-1] is given below:

Let L_i denote the length of the longest monotonically increasing sequence starting at index i in the array.
Initialize $L_{n-1} = 1$
For all I such that $0 \leq I \leq n - 2$

$$L_i = \begin{cases} 1 + L_{i+1} & \text{if } A[i] < A[i+1] \\ 1 & \text{otherwise} \end{cases}$$

Finally the length of the longest monotonically increasing sequence is Max (L_0, L_1, …, L_{n-1}).

Which of the following statements is TRUE?

(a) The algorithm uses dynamic programming paradigm
(b) The algorithm has a linear complexity and uses branch and bound paradigm

(c) The algorithm has a non-linear polynomial complexity and uses branch and bound paradigm
(d) The algorithm uses divide and conquer paradigm.

Solution: Option (a)

19. Which of the following standard algorithms is not Dynamic Programming based?

(a) Bellman–Ford Algorithm for single source shortest path
(b) Floyd Warshall Algorithm for all pairs shortest paths
(c) 0-1 Knapsack problem
(d) Prim's Minimum Spanning Tree

Solution: Option (d)

20. We use dynamic programming approach when:

(a) It provides optimal solution
(b) The solution has optimal substructure
(c) The given problem can be reduced to the 3-SAT problem
(d) It's faster than Greedy

Solution: Option (b)