Studytonight – Algo test 1 – Aditya Jain

1. What is recurrence for worst case of QuickSort and what is the time complexity in Worst case?

a) Recurrence is T(n) = T(n-2) + O(n) and time complexity is O(n^2)
b) Recurrence is T(n) = T(n-1) + O(n) and time complexity is O(n^2)
c) Recurrence is T(n) = 2T(n/2) + O(n) and time complexity is O(nLogn)
d) Recurrence is T(n) = T(n/10) + T(9n/10) + O(n) and time complexity is O(nLogn)

Explanation: The worst case of QuickSort occurs when the picked pivot is always one of the corner elements in sorted array. In worst case, QuickSort recursively calls one subproblem with size 0 and other subproblem with size (n-1). So recurrence is T(n) = T(n-1) + T(0) + O(n)

2. Suppose we have a O(n) time algorithm that finds median of an unsorted array. Now consider a QuickSort implementation where we first find median using the above algorithm, then use median as pivot. What will be the worst case time complexity of this modified QuickSort.

a) O(n^2 Logn)
b) O(n^2)
c) O(n Logn Logn)
d) O(nLogn)

Explanation: If we use median as a pivot element, then the recurrence for all cases becomes T(n) = 2T(n/2) + O(n) The above recurrence can be solved using Master Method. It falls in case 2 of master method.

3. Given an unsorted array. The array has this property that every element in array is at most k distance from its position in sorted array where k is a positive integer smaller than size of array. Which sorting algorithm can be easily modified for sorting this array and what is the obtainable time complexity?

a) Insertion Sort with time complexity O(kn)
b) **Heap Sort with time complexity O(nLogk)**
c) Quick Sort with time complexity O(kLogk)
d) Merge Sort with time complexity O(kLogk)

4. **Which of the following is not true about comparison based sorting algorithms?**

a) The minimum possible time complexity of a comparison based sorting algorithm is O(nLogn) for a random input array
b) Any comparison based sorting algorithm can be made stable by using position as a criteria when two elements are compared
c) Counting Sort is not a comparison based sorting algorithm
d) **Heap Sort is not a comparison based sorting algorithm.**

5. **What is time complexity of fun()?**

```
int fun(int n)
{
  int count = 0;
  for (int i = n; i > 0; i /= 2)
     for (int j = 0; j < i; j++)
        count += 1;
  return count;
}
```

a) O(n^2)
b) O(nLogn)
c) **O(n)**
d) O(nLognLogn)

Explanation: For a input integer n, the innermost statement of fun() is executed following times. n + n/2 + n/4 + ... 1 So time complexity T(n) can be written as T(n) = O(n + n/2 + n/4 + ... 1) = O(n) The value of count is also n + n/2 + n/4 + .. + 1

6. What is the time complexity of fun()?

```
int fun(int n)
{
  int count = 0;
  for (int i = 0; i < n; i++)
    for (int j = i; j > 0; j--)
      count = count + 1;
  return count;
}
```

a) Theta (n)
b) **Theta (n^2)**
c) Theta (n*Logn)
d) Theta (nLognLogn)

Explanation: The time complexity can be calculated by counting number of times the expression "count = count + 1;" is executed. The expression is executed 0 + 1 + 2 + 3 + 4 + ... + (n-1) times. Time complexity = Theta(0 + 1 + 2 + 3 + .. + n-1) = Theta (n*(n-1)/2) = Theta(n^2)

7. The recurrence relation capturing the optimal time of the Tower of Hanoi problem with n discs is.

a) T(n) = 2T(n – 2) + 2
b) T(n) = 2T(n – 1) + n
c) T(n) = 2T(n/2) + 1
d) **T(n) = 2T(n – 1) + 1**

Explanation: Following are the steps to follow to solve Tower of Hanoi problem recursively.

Let the three pegs be A, B and C. The goal is to move n pegs from A to C.

To move n discs from peg A to peg C:

move n-1 discs from A to B. This leaves disc n alone on peg A

move disc n from A to C

move n-1 discs from B to C so they sit on disc n

The recurrence function T(n) for time complexity of the above recursive solution can be written as following. T(n) = 2T(n-1) + 1

8. Let w(n) and A(n) denote respectively, the worst case and average case running time of an algorithm executed on an input of size n. which of the following is ALWAYS TRUE?

(A) $A(n) = \Omega(W(n))$
(B) $A(n) = \Theta(W(n))$
(C) $A(n) = O(W(n))$
(D) $A(n) = o(W(n))$

a) A
b) B
c) C
d) D

Explanation: The worst case time complexity is always greater than or same as the average case time complexity.

9. Which of the following is not O(n^2)?

a) (15^10) * n + 12099
b) n^1.98
c) n^3 / (sqrt(n))
d) (2^20) * n

Explanation: The order of growth of option c is n^2.5 which is higher than n^2.

10. Which of the given options provides the increasing order of asymptotic complexity of functions f1, f2, f3 and f4?

```
f1(n) = 2^n

f2(n) = n^(3/2)

f3(n) = nLogn

f4(n) = n^(Logn)
```

a) f3, f2, f4, f1
b) f3, f2, f1, f4
c) f2, f3, f1, f4
d) f2, f3, f4, f1

11. Consider the following program fragment for reversing the digits in a given integer to obtain a new integer. Let n = D1D2…Dm

```
int n, rev;
rev = 0;
while (n > 0)
{
  rev = rev*10 + n%10;
  n = n/10;
}
```

The loop invariant condition at the end of the ith iteration is:

a) n = D1D2….Dm-i and rev = DmDm-1…Dm-i+1
b) n = Dm-i+1…Dm-1Dm and rev = Dm-1….D2D1
c) n != rev
d) n = D1D2….Dm and rev = DmDm-1…D2D1

Explanation: We can get it by taking an example like n = 54321. After 2 iterations, rev would be 12 and n would be 543.

12. What is the best time complexity of bubble sort?

a) N^2
b) NlogN
c) N
d) N(logN)^2

Explanation: The bubble sort is at its best if the input data is sorted. i.e. If the input data is sorted in the same order as expected output. This can be achieved by using one boolean variable. The boolean variable is used to check whether the values are swapped at least once in the inner loop.

13. What is the worst case time complexity of insertion sort where position of the data to be inserted is calculated using binary search?

a) N
b) NlogN
c) N^2
d) N(logN)^2

14. The tightest lower bound on the number of comparisons, in the worst case, for comparison-based sorting is of the order of

a) N

b) N^2

c) NlogN

d) N(logN)^2

Explanation:

The number of comparisons that a comparison sort algorithm requires increases in proportion to Nlog(N), where N is the number of elements to sort. This bound is asymptotically tight: Given a list of distinct numbers (we can assume this because this is a worst-case analysis), there are N factorial permutations exactly one of which is the list in sorted order. The sort algorithm must gain enough information from the comparisons to identify the correct permutations. If the algorithm always completes after at most f(N) steps, it cannot distinguish more than 2^f(N) cases because the keys are distinct and each comparison has only two possible outcomes. Therefore, 2^f(N) >= N! or equivalently f(N) >= log(N!). Since log(N!) is Omega(NlogN), the answer is NlogN.

15. In a modified merge sort, the input array is splitted at a position one-third of the length(N) of the array. What is the worst case time complexity of this merge sort?

a) N(logN base 3)

b) N(logN base 2/3)

c) N(logN base 1/3)

d) N(logN base 3/2)

Explanation: The time complexity is given by: T(N) = T(N/3) + T(2N/3) + N Solving the above recurrence relation gives, T(N) = N(logN base 3/2)