ANALYSIS OF ALGORITHMS
(SET: 3)

Solutions

Q1. An unordered list contains n distinct elements. The number of comparisons to find an element in this list that is neither maximum nor minimum is:

(a) Θ(nlogn) (b) Θ(n)
(c) Θ(logn) (d) Θ(1)

Solution: Option (d)

Explanation:
We only need to consider any 3 elements and compare them. So the number of comparisons is constants, that makes time complexity as Θ(1).

Q2. Consider the following C function:

```
int fun1 (int n)
{
  int i, j, k, p, q = 0;
  for (i = 1; i<n; ++i)
  {
    p = 0;
    for (j=n; j>1; j=j/2)
      ++p;
    for (k=1; k<p; k=k*2)
      ++q;
  }
  return q;
}
```

Which one of the following most closely approximates the return value of the function fun1?
(a) n^3 (b) $n (\log n)^2$
(c) nlogn (d) nlog(logn)

Solution: Option (d)

Q3. The increasing order of following functions in terms of asymptotic complexity is:

$f_1(n) = n^{0.999999} \log n$
$f_2(n) = 10000000n$
$f_3(n) = 10000000^n$
$f_4(n) = n^2$

(a) $f_1(n)$; $f_4(n)$; $f_2(n)$; $f_3(n)$
(b) $f_1(n)$; $f_2(n)$; $f_3(n)$; $f_4(n)$
(c) $f_2(n)$; $f_1(n)$; $f_4(n)$; $f_3(n)$
(d) $f_1(n)$; $f_2(n)$; $f_4(n)$; $f_3(n)$

Solution: Option (d)

Q4. Consider the following three claims:

1. $(n + k)^m = \Theta(n^m)$, where k and m are constants
2. $2^{n+1} = O(2^n)$
3. $2^{2n+1} = O(2^n)$

Which of these claims are correct?
(a) 1 and 2
(b) 1 and 3
(c) 2 and 3
(d) 1, 2 and 3

Solution: Option (a)

Explanation:
$(n + k)^m$ and $\Theta(n^m)$ are asymptotically same as theta notation can always be written by taking the leading order term in a polynomial expression.

2^{n+1} and $O(2^n)$ are also asymptotically same as 2^{n+1} can be written as $2 * 2^n$ and constant multiplication/addition doesn't matter in theta notation.

2^{2n+1} and $O(2^n)$ are not same as constant is in power.

Q5. The recurrence equation:

$T(1) = 1$
$T(n) = 2T(n - 1) + n, n \geq 2$

evaluates to
(a) $2^{n+1} - n - 2$
(b) $2^n - n$
(c) $2^{n+1} - 2n - 2$
(d) $2^n - n$

Solution: Option (a)

Consult Master's theorem.

Q6. Let A[1, …, n] be an array storing a bit (1 or 0) at each location, and f(m) is a function whose time complexity is θ(m). Consider the following program fragment written in a C like language:

```
counter = 0;
for (i = 1; i < = n; i++)
{
    if (A[i] == 1)
      counter++;
    else {
      f(counter);
      counter = 0;
    }
}
```

The complexity of this program fragment is:

(a) $\Omega(n^2)$
(b) $\Omega(n\log n)$ and $O(n^2)$
(c) $\theta(n)$
(d) $O(n)$

Solution: Option (c)

Explanation:
Please note that inside the else condition, f() is called first, then counter is set to 0.

Consider the following cases:

a) All 1s in A[]: Time taken is $\Theta(n)$ as
only counter++ is executed n times.

b) All 0s in A[]: Time taken is $\Theta(n)$ as
only f(0) is called n times

c) Half 1s, then half 0s: Time taken is $\Theta(n)$ as
only f(n/2) is called once.

Q7. Two matrices M_1 and M_2 are to be stored in arrays A and B respectively. Each array can be stored either in row-major or column-major order in contiguous memory locations. The time complexity of an algorithm to compute $M_1 \times M_2$ will be:

(a) best if A is in row-major, and B is in column- major order
(b) best if both are in row-major order
(c) best if both are in column-major order
(d) independent of the storage scheme

Solution: Option (d)

Explanation:
This is a trick question. Note that the questions ask about time complexity, not time taken by the program. For time complexity, it doesn’t matter how we store array elements, we always need to access same number of elements of M_1 and M_2 to multiply the matrices. It is always constant or O(1) time to do element access in arrays, the constants may differ for different schemes, but not the time complexity.

Q8. Consider the following C-function:

```
double foo (int n)
{
   int i;
   double sum;
   if (n = = 0) return 1.0;
   else
   {
      sum = 0.0;
      for (i = 0; i < n; i++)
         sum += foo (i);
      return sum;
   }
}
```

Suppose we modify the modify the above function foo( ) and store the values of foo(i), 0<=i< n, as and when they are computed. With this modification, the time complexity for function foo() is significantly reduced. The space complexity of the modified function would be:

(a) O(1) (b) O(n)
(c) O(n!) (d) $O(n^n)$

Solution: Option ( )

Q9. Consider the following C-function:

```
 double foo (int n)
{
    int i;
    double sum;
    if (n = = 0) return 1.0;
 else
    {
        sum = 0.0;
        for (i = 0; i < n; i++)
            sum += foo (i);
        return sum;
   }
}
```

The space complexity of the above function is:

(a) O(1) (b) O(n)

(c) O(n!) (d) $O(n^n)$

Solution: Option (a)

Explanation:
Note that the function foo() is recursive. Space complexity is O(n) as there can be at most O(n) active functions (function call frames) at a time.

Q10. Consider the following pseudo code. What is the total number of multiplications to be performed?

```
D = 2
for i = 1 to n do
   for j = i to n do
      for k = j + 1 to n do
           D = D * 3
```

(a) Half of the product of the 3 consecutive integers
(b) One-third of the product of the 3 consecutive integers
(c) One-sixth of the product of the 3 consecutive integers

(d) None of the above

Solution: Option (c)

Q11. Consider the equality:

$$\sum_{i=0}^{n} i^3 = X$$

and the following choices for X:

I. $\Theta(n^4)$
II. $\Theta(n^5)$
III. $O(n^5)$
IV. $\Omega(n^3)$

The equality above remains correct if X is replace by:

(a) Only I
(b) Only II
(c) I or II or IV but not II
(d) II or III or IV but not I

Solution: Option (c)

Explanation:
X = Sum of the cubes of {1, 2, 3, .. n|
X = $n^2 (n+1)^2 / 4$

Q12. Two main measures for the efficiency of an algorithm are:

(a) Processor and memory
(b) Complexity and capacity
(c) Time and space
(d) Data and space

Solution: Option (c)

Q13. The time factor when determining the efficiency of algorithm is measured by:

(a) Counting microseconds
(b) Counting the number of key operations
(c) Counting the number of statements
(d) Counting the kilobytes of algorithm

Solution: Option (b)

Q14. The space factor when determining the efficiency of algorithm is measured by:

(a) Counting the maximum memory needed by the algorithm
(b) Counting the minimum memory needed by the algorithm
(c) Counting the average memory needed by the algorithm
(d) Counting the maximum disk space needed by the algorithm

Solution: Option (a)

Q15. Which of the following case does not exist in complexity theory:

(a) Best case (b) Worst case
(c) Average case (d) Null case

Solution: Option (d)

<u>Practice Problems on Master's Theorem</u>

Practice Problems:

For each of the following recurrences, give an expression for the runtime T(n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

1. $T(n) = 3T\left(\frac{n}{2}\right) + n^2$

2. $T(n) = 4T\left(\frac{n}{2}\right) + n^2$

3. $T(n) = T\left(\frac{n}{2}\right) + 2^n$

4.$T\ (n) = 2^n T\left(\frac{n}{2}\right) + n^n$

5. $T(n) = 16T\left(\frac{n}{4}\right) + n$

6. $T(n) = 2T\left(\frac{n}{2}\right) + n\log n$

7. $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$

8. $T(n) = 2T\left(\frac{n}{4}\right) + n^{0.51}$

9. $T(n) = 0.5T\left(\frac{n}{2}\right) + \frac{1}{n}$

10. $T(n) = 16T\left(\frac{n}{4}\right) + n!$

11. $T(n) = \sqrt{2T}\left(\frac{n}{2}\right) + \log n$

12. $T(n) = 3T\left(\frac{n}{2}\right) + n$

13. $T(n) = 3T\left(\frac{n}{3}\right) + \sqrt{n}$

14. $T(n) = 4T\left(\frac{n}{2}\right) + cn$

15. $T(n) = 3T\left(\frac{n}{4}\right) + n\log n$

16. $T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{2}$

17. $T(n) = 6T\left(\frac{n}{3}\right) + n^2 \log n$

18. $T(n) = 4T\left(\frac{n}{2}\right) + \frac{n}{\log n}$

19. $T(n) = 64T\left(\frac{n}{8}\right) - n^2 \log n$

20. $T(n) = 7T\left(\frac{n}{3}\right) + n^2$

21. $T(n) = 4T\left(\frac{n}{2}\right) + \log n$

22. $T(n) = T\left(\frac{n}{2}\right) + n(2 - \cos n)$

---

[1]most of the time, k=0

Solutions

1. $T(n) = 3T\left(\frac{n}{2}\right) + n^2 \Rightarrow T(n) = \Theta(n^2)$(**Case 3**)

2. $T(n) = 4T\left(\frac{n}{2}\right) + n^2 \Rightarrow T(n) = \Theta(n^2 \log n)$(**Case 2**)

3. $T(n) = T\left(\frac{n}{2}\right) + 2^n \Rightarrow T(n) = \Theta(2^n)$(**Case 3**)

4. $T(n) = 2^n T\left(\frac{n}{2}\right) + n^n \Rightarrow$ Does not apply (**a is not constant**)

5. $T(n) = 16T\left(\frac{n}{4}\right) + n \Rightarrow T(n) = \Theta(n^2)$(**Case 1**)

6. $T(n) = 2T\left(\frac{n}{2}\right) + n\log n \Rightarrow T(n) = n \log^2 n$(**Case 2**)

7. $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$

$\Rightarrow$ Does not apply (non − polynomial difference between f(n) and $n^{\log_{ba}}$)

8. $T(n) = 2T\left(\frac{n}{4}\right) + n^{0.51} \Rightarrow T(n) = \Theta(n^{0.51})$(**Case 3**)

9. $T(n) = 0.5T\left(\frac{n}{2}\right) + \frac{1}{n} \Rightarrow$ Does not apply (**a** $<$ **1**)

10. $T(n) = 16T\left(\frac{n}{4}\right) + n! \Rightarrow T(n) = \Theta(n!)$(**Case 3**)

11. $T(n) = \sqrt{2T}\left(\frac{n}{2}\right) + \log n \Rightarrow T(n) = \Theta(\sqrt{n})$(**Case 3**)

12. $T(n) = 3T\left(\frac{n}{2}\right) + n \Rightarrow T(n) = \Theta(n\log 3)$(**Case 1**)

13. $T(n) = 3T\left(\frac{n}{2}\right) + \sqrt{n} \Rightarrow T(n) = \Theta(n)$(**Case 1**)

14. $T(n) = 4T\left(\frac{n}{2}\right) + cn \Rightarrow T(n) = \Theta(n^2)$(**Case 1**)

15. $T(n) = 3T\left(\frac{n}{4}\right) + n\log n \Rightarrow T(n) = \Theta(n \log n)$(**Case 3**)

16. $T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{2} \Rightarrow T(n) = \Theta(n \log n)$(**Case 2**)

17. $T(n) = 6T\left(\frac{n}{3}\right) + n^2 \log n \Rightarrow T(n) = \Theta(n^2 \log n)$(**Case 3**)

18. $T(n) = 4T\left(\frac{n}{2}\right) + \frac{n}{\log n} \Rightarrow T(n) = \Theta(n^2)$(**Case 1**)

19. $T(n) = 64T\left(\frac{n}{8}\right) - n^2 \log n \Rightarrow$ Does not apply (f(n)is not positive)

20. $\mathrm{T}(\mathrm{n}) = 7\mathrm{T}\left(\frac{\mathrm{n}}{3}\right) + \mathrm{n}^2 \Rightarrow \mathrm{T}(\mathrm{n}) = \Theta(\mathrm{n}^2)$(**Case 3**)

21. $\mathrm{T}(\mathrm{n}) = 4\mathrm{T}\left(\frac{\mathrm{n}}{2}\right) + \log \mathrm{n} \Rightarrow \mathrm{T}(\mathrm{n}) = \Theta(\mathrm{n}^2)$(**Case 1**)

22. $\mathrm{T}(\mathrm{n}) = \mathrm{T}\left(\frac{\mathrm{n}}{2}\right) + \mathrm{n}(2 - \cos \mathrm{n}) \Rightarrow$ Does not apply.

We are in **Case 3**, but the regularity condition is violated.

(Consider $\mathrm{n} = 2\pi\mathrm{k}$, where k is odd and arbitrarily large. For any such choice of n, you can show that $\mathrm{c} \geq \frac{3}{2}$, thereby violating the regularity condition.