Q.1) Max Marks: 1

What is the solution for the following recurrence relation.

$T(n)= 2nT(n/2)+n^2$

A O(nlogn)

B O(log log n)

C Cannot be determined.

D None of the above. Correct Option

Solution: (D)

Solution D

The recurrence relation given is

$T(n)= 2nT(n/2)+n^2$
$=n*(2(n/2) +n)$
The solution to the inner recurrence can be given by O(nlogn)
$=n*nlogn$
$=O(n^2logn)$

Q.2) Max Marks: 1

Which one of the following does NOT represent the array if bubble sort to sort the elements in ascending order is run after one iteration

A 1, 2, 5, 7, 8, 6, 4, 3, 9.

B 3, 2, 5, 9, 8, 6, 4, 1, 7. Correct Option

Solution: (B)
Answer B
After one iteration the greatest element reaches its correct position i.e. the last position, out of all the options only in option b The largest element is not present in the last position.
In the options A and C the largest element that is 9 is present in the last position but in case of the B option it is not the case, therefore B option is not a possible outcome after one iteration of the Bubble Sort.

C 1, 2, 5, 6, 7, 8, 4, 3, 9.

D None of the above

Q.3) Max Marks: 1

Which of the following statements are true with respect to Quick Sort.

I. After calling the partition function for once at most one element will reach the correct position.
II. After calling the partition function the relative inversions with respect to the pivot element are zero.

A Only I is true

B Only II is true Correct Option

Solution: (B)
Statement I
let us consider the example of an unsorted array as below
4, 3, 1, 7, 12, 9.

after calling the partition function on this array considering the first element i.e. 4 is the pivot the resulting array would be

3, 1, 4, 7, 12, 9

As far as statement I is considered, as you can see that 7 and 4 both have reached their correct position we can only say that the pivot will reach its correct position but even a non-pivot element may reach its correct position if it is already in its correct place or due to swapping in the partition routine, hence statement I is false.
Statement II
We know that after we call the partition routine we will always have the elements smaller than the pivot to the left of it and elements greater than the pivot to the right of it, this is the reason why there are no inversions with respect to the pivot element after the partition routine is called.

C Only I and II

D Neither I nor II is true.

Q.4) Max Marks: 1

Let f(n) =O(n), g(n)=Ω(n) and h(n)=Θ(n). Then g(n)+f(n).h(n) is _____?

A $\Omega(n^2)$

B $\Theta(n^2)$

C $\Omega(n)$ Correct Option

Solution: (C)

Solution:
Ω(n) + O(n)*Θ(n)
(minimum time taken : n) + (max time taken : n)*(average time taken : n)
= min time taken : n (or) max time taken : n^2
in the given options we have Ω(n) not $O(n^2)$
So Ω(n)

D Θ(n)

Q.5) Max Marks: 1

If f(n)=O(g(n)) what can be said about the f and g.

I. f(n) grows slower than g(n) for all values of n
II. g(n) grows faster for most of the values of n
III. g(n) grows faster or equal to the f(n) beyond a particular value of n.
IV. f(n) grows slower than or equal to g(n) beyond a particular value on n.

A. II and I are True.
B. III and IV are True.
C. I and IV are True.
D. II and III are True.

A II and I are True.

B III and IV are True. Correct Option

Solution: (B)

If f(n)=O(g(n)), then according to the definition of the the Big Oh notation
$f(n)<=c*g(n)$ for $n>n_o$, for some c,n_0.
According to this definition we have statements III and IV are only true.

C I and IV are True.

D II and III are True.

Q.6) Max Marks: 1

Consider the following two functions

$f_1(n)= n^2$ if $0<n<10000$
$=n^{\log \log n}$ $n>= 10000$

$f_2(n)= n^{\log n}$ if $0<n<1000000$
$=n^{\sqrt{\log n}}$ $n>=1000000$

Which of the following is true with respect to the above two functions.

A $f_1(n)=O(f_2(n))$ Correct Option

Solution: (A)

Solution
Big Oh notation is used for larger values of n, so we should consider this for larger values of n, for n>1000000 $f_1(n)=n^{\log \log n}$ $f_2(n)=n^{\sqrt{\log n}}$
Now we have to compare the two functions

$n^{\log \log n}$ and $n^{\sqrt{\log n}}$
taking log on both the sides
logn *(log log n) and logn *(√logn)
logn is common on both sides and now comparing log logn and √logn
substituting large value for let us take $n=2^{2048}$
log log 2^{2048}=log 2048=11
√logn=√2048=2^5√2=32√2
clearly f2 grows faster than f1.

B $f_2(n)=O(f_1(n))$

C **Cannot be determined**

D **None of the above.**

Q.7) Max Marks: 1

A Modified version of merge sort is devised with a modified merge method is used where the merge takes $O(n^2)$ time to merge lists that sum up to size n, the time complexity for such a version of merge sort is

A O(nlogn)

B $O(n^2 \log n)$ Correct Option

Solution: (B)

Solution:

The Recurrence relation for such an implementation can be given by
$T(n)=2T(n/2)+n^2$
By using masters theorem we get
a=2, b=2 and k=2

$T(n)=\theta(n^2)$
The most appropriate option is as we do not have $\theta(n^2)$ the most appropriate option is $O(n^2 \log n)$.

C O(n)

D $O(\sqrt{n})$

Q.8) Max Marks: 1

In we apply quicksort algorithm on a set of n numbers if in every recursive subroutine of the algorithm, the algorithm chooses the middle element of that set as the pivot. Then the running time of the algorithm is?

A $\Theta(n)$.

B $\Theta(n \log n)$.

C $\Theta(n^{1.5})$.

D $\Theta(n^2)$. Correct Option

Solution: (D)
If the middle element is selected then in the worst case it is always possible that the least or the largest element is selected as the pivot, therefore the worst-case time complexity is given by the recurrence relation T(n)=T(n-1)+O(n) which comes to O(n^2).

Q.9) Max Marks: 1

$T(n)=2T(n/2)+n!$

The time complexity of the above recurrence relation is

A O(nlogn)

B $O(2^{\log n!})$ Correct Option

Solution: (B)
Time complexity of the recurrence relation is given by O(n!) as it is the special case when f(n)=n! or $f(n)=a^n$. It can also be written as $O(2^{\log n!})$.

C $O(n^2)$

D None of the above

Q.10) Max Marks: 1

An e-commerce site on search for a particular keyword returns the search results in order of popularity by default, also the user is allowed to sort by price (low to high) or (high to low) however if two or more products are of the same price then among the same price products should be presented to the user in the same order of popularity. Which of the following sorting algorithms cannot be used for such a scenario.

A Merge Sort

B Bubble Sort

C Quick Sort — Correct Option

Solution: (C)
Answer 3
Quick Sort is not a stable sorting algorithm, the popularity acts as satellite data and the order of satellite data is preserved only by stable sorting algorithms.

D None of the above

Q.11) Max Marks: 2

Arrange the following in non-increasing order of asymptotic complexity.

I. The number of ways to seat n people around a circular table.
II. The number of 4 element subsets of a set of size n.
III. The time to find a given element in a sorted array of length n.
IV. The time to find a given element in a doubly-linked list of length n.
V. The biggest number that can be represented with n bits.
VI. Time to sort a list of n items if you are only allowed to swap adjacent elements
VII. The number of edges in a tree with n nodes.

A III, VII, II, VI, II, IV, I

B III, VII, IV, VI, II, V, I — Correct Option

Solution: (B)

I. Number of ways to seat n people around a circular table.
=(n-1)!=O(n!)

II. Number of 4 element subsets of a set of size n.
No of 4 element subsets = $C(n,4)=(n*(n-1)*(n-2)*(n-3))/(4*3*2*1)=O(n^4)$

III. Time to find a given element in a sorted array of length n.
By using binary search by using binary search we can search for a given element in O(log n)

IV. Time to find a given element in a doubly linked list of length n.
Finding any element in a linked/doubly linked list would require to search in a fashion similar to linear search which takes O(n) time.

V. Biggest number that can be represented with n bits.
Using n bits we can represent 2^n numbers=$O(2^n)$.

VI. Time to sort a list of n items if you are only allowed to swap adjacent elements
If we are allowed to swap only the adjacent elements then it is referring to Bubble Sort
the worst case time complexity for bubble sort is $O(n^2)$.

VII. Number of edges in a tree with n nodes.
The Number of edges in a tree with n nodes is exactly (n-1)=O(n).

If we arrange them in non-decreasing order of asymptotic complexity which is as follows we get option 2 as correct.
III< VII< IV< VI< II< V< I.

C III, VII, IV, VI, II,I, V

D III, VII, IV, II, VI, V, I

Q.12) Max Marks: 2

Time complexity of the below fragment of code is given by

```
for(i=0;i<n;i++)
{
        if(i%2==0)
        {
                for(j=i;j<n;j++)
                {
                        if(i%2==0 && j%2==0)
                                printf("GATE 2020");
                }
        }
}
```

A $O(n^{5/2})$

B $\Theta(log\ n)$

C $\Theta(nlogn)$

D $O(n^2 log\ n)$ **Correct Option**

Solution: (D)

Solution:

The outer loop is executed n times.
The inner loop is executed n/2 times with i is even, inside the inner loop we have statements of constant time and it gets executed n times when i=0, when i=2 then it gets executed n-2 times, and so on when i=n the inner loop gets executed 0 times when n is even otherwise when i=n-1 it will get executed n-1 times, total no of times 0+2+4+..n times,
=(n-1)*n/4= O(n^2) the most appropriate or tightest upper bound is $O(n^2 log\ n)$ among the options as it is the most slowest growing among $n^{5/2}$ and n^2log n.

Q.13) Max Marks: 2

What is the time complexity of the following fragment of code.

```
for (i = 1; i < n*n*n; i *= n )
{
        for (j = 0; j < n; j += 2 )
        {
                for (k = 1; k < n; k *= 3 )
                {
                        // constant number C of operations
                }
}
```

A O(n^4logn)

B O(n^5)

C O(nlogn) **Correct Option**

Solution: (C)

The outer loop will execute 3 times which is constant and the next inner loop will execute n/2 number of times and the inner loop will execute log_3n times as each time the value of k is multiplied by 3.
Therefore the total time complexity is given by
3*(n/2)*log_3n=O(nlogn).

D **None of the Above.**

Q.14) Max Marks: 2

Which of the two DBMS packages, A and B, of the same price should be bought to maintain a large data base having each approximately 10000 records, the maximum requirement for the database is can grow by 10%, As was found empirically, the average time to process n records with A and B is

$T_A(n) = 0.1n \log_2 n$ milliseconds and $T_B(n) = 2.5n$ milliseconds.

A Either Package A or Package B can be chosen

B Package A — Correct Option

Solution: (B)

Solution B

We need to compare the growth of the functions
$0.1n\log_2 n$ and $2.5\,n$
multiplying both by 10

$n\log_2 n$ and $25n$
n is common on both sides ignoring it
$\log_2 n$ and 25
clearly the right side function is a constant and the left side is a function of n will grow faster
$\log_2 n > 25$
$n > 2^{25}$
If the no of records exceed 2^{25} then package A takes more processing time and package B takes less processing time, and if $n < 2^{25}$ the package B takes more processing time and package A takes less processing time.
The maximum requirement of the records is 10000+10%=11000 records which is $< 2^{25}$, for this situation package A is a better choice.

C Package B

D It cannot be predicted beforehand as the number of records are dynamic.

Q.15) Max Marks: 2

You have found empirically that the implemented insertionSort and heapSort methods spent on an average 5 µs and 1 µs, respectively, to sort an array of 1000 elements. If time each algorithm will spend for sorting an array of 1,000, 000, 000 elements is t_1 and t_2 in µs respectively, then the value of $\lfloor \log_{10} t_1 + \log_{10} t_2 \rfloor$=------. Where $\lfloor\ \rfloor$ represents the floor function.

Correct Answer

Solution: (19)

The time complexity of Insertion Sort $O(n^2)$ the average time taken is

$k*n^2=5$ µs
$k*(10^3)^2=5$
$k=5*10^{-6}$

for sorting 1, 000, 000, 000 elements it would require

$t_1=k*n^2$
$=(5*10^{-6})*(10^9)^2$
$=5*10^{12}$ µs

The time complexity of Heap Sort O(nlogn) the average time taken is

$k*n\log n=1$ µs
$k*(10^3 \log 10^3)=1$
$k=(1/3)*(10^{-3})*(1/\log 10)$

for sorting 1, 000, 000, 000 elements it would require

$t_2=k*n\log n$
$=(1/3)*(10^{-3})*(1/\log 10))*(10^9 \log 10^9)$
$=3*10^6$ µs

$\log_{10} t_1 + \log_{10} t_2$
$=12+\log_{10} 5+6+\log_{10} 3$
$=18+\log_{10} 15$
$=\lfloor 19+\log_{10} 1.5 \rfloor$ (as $\log_{10} 1.5<1$)
$=19$

close