IARE '24 DISCORD TEAM

II-II

# DAA MODULE 1 SOLUTIONS

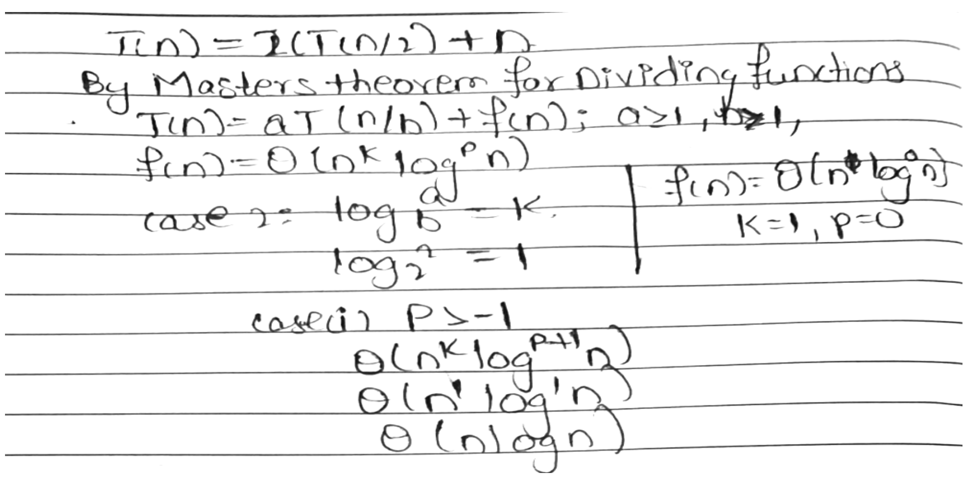
NEHANJALI • SHRESHIKA • UJJWAL • NISHANT • PRANAV

INTRO TO DESIGN AND ANALYSIS OF ALGORITHMS

# DAA MODULE 1

## PART A

**1.Solve the following recurrence relation T(n)=2 T(n/2) + n, and T(1)=2.**



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

By Masters theorem for Dividing functions

$T(n) = aT(n/b) + f(n);\ a \geq 1,\ b > 1,$

$f(n) = \Theta(n^k \log^p n)$

case 2: $\log_b^a = k$ | $f(n) = \Theta(n^1 \log^0 n)$

$\log_2^2 = 1$ | $K = 1,\ p = 0$

case(i) $p > -1$

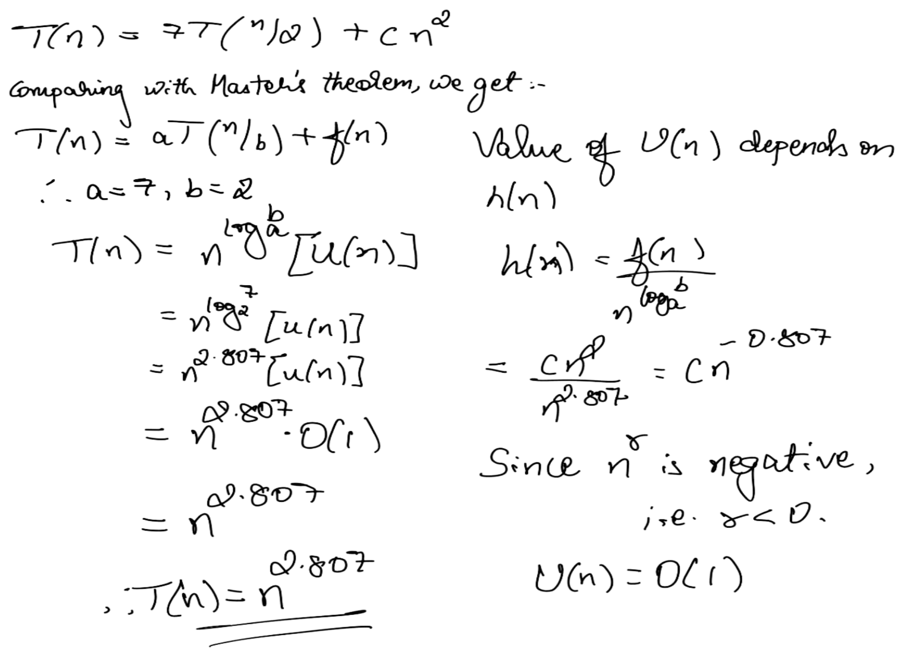
$\Theta(n^k \log^{p+1} n)$

$\Theta(n^1 \log^1 n)$

$\Theta(n \log n)$

+

**2. Solve the following recurrence relation T(n) = 7T(n/2)+cn2**



$T(n) = 7T(n/2) + cn^2$

Comparing with Master's theorem, we get :-

$T(n) = aT(n/b) + f(n)$

$\therefore a = 7,\ b = 2$

$T(n) = n^{\log_a^b}[U(n)]$

$= n^{\log_2^7}[u(n)]$

$= n^{2.807}[u(n)]$

$= n^{2.807} \cdot O(1)$

$= n^{2.807}$

$\therefore T(n) = n^{2.807}$

Value of $U(n)$ depends on $h(n)$

$h(n) = \frac{f(n)}{n^{\log_a^b}}$

$= \frac{cn^2}{n^{2.807}} = cn^{-0.807}$

Since $n^r$ is negative, i.e. $r < 0$.

$U(n) = O(1)$

**3. Solve the recurrence relation T(n)=T(1), n=1 T(n)=T(n/2) + c, n¿1 and n is a power of 2**.

3) $T(n) = T(n/2) + C$ $\quad$ $T(n) = 1$ if $n = 1$

$n$ is power of 2.

$T(n/2) = T(n/4) + C$

$T(n/4) = T(n/8) + C$

$T(n) = T(n/4) + C + C$

$= T(n/8) + C + C + C$

$= T(n/8) + 3C$

K times.

$T(n) = T(n/2^k) + kC$

$\frac{n}{2^k} = 1$ $\quad n = 2^k$ log on b.s.

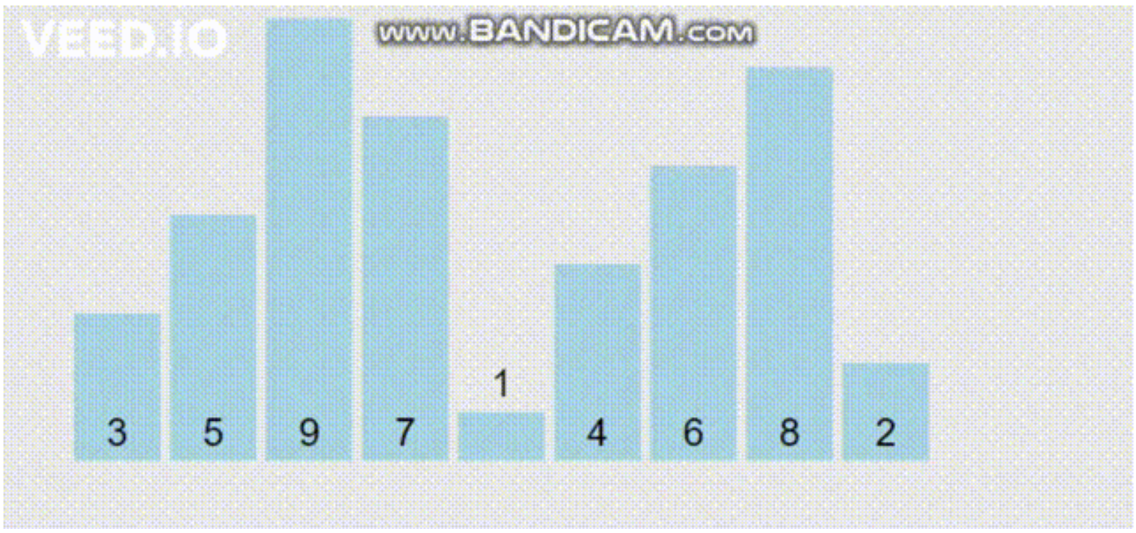
$\log n = \underline{\underline{k}}$

$T(n) = 1 + \log n \cdot C$

$\therefore T(n) = \log_2^n$.

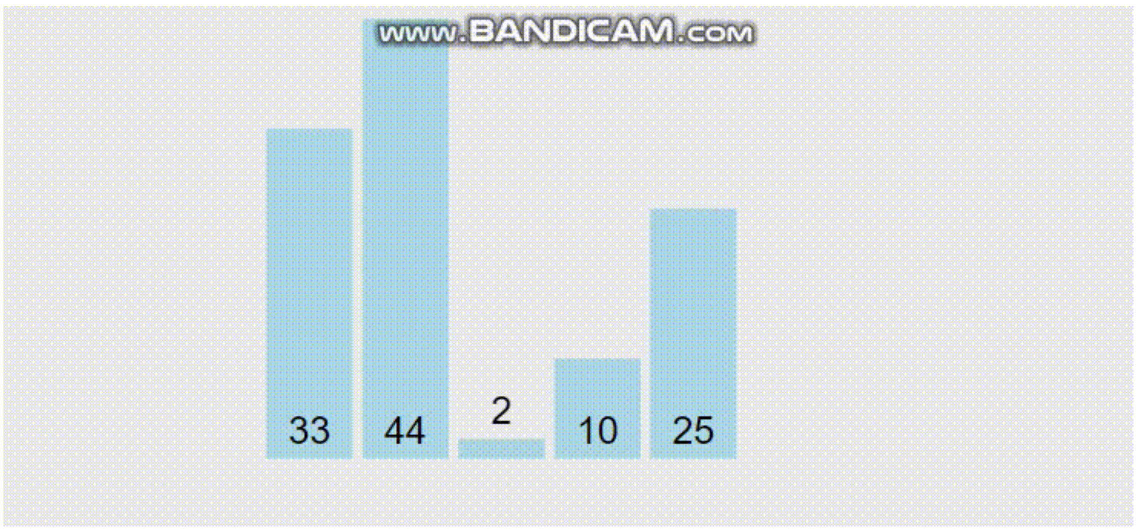
**4. Apply a quick sort algorithm and simulate it for the following data sequence: 3 5 9 7 1 4 6 8 2.**

```
quickSort(array, leftmostIndex, rightmostIndex)
  if (leftmostIndex < rightmostIndex)
    pivotIndex <- partition(array, leftmostIndex, rightmostIndex)
    quickSort(array, leftmostIndex, pivotIndex - 1)
    quickSort(array, pivotIndex, rightmostIndex)
partition(array, leftmostIndex, rightmostIndex)
  set rightmostIndex as pivotIndex
  storeIndex <- leftmostIndex - 1
 for i <- leftmostIndex + 1 to rightmostIndex
 if element[i] < pivotElement
    swap element [i] and element[storeIndex]
    storeIndex++
  swap pivotElement and element[storeIndex+1]
return storeIndex + 1
```



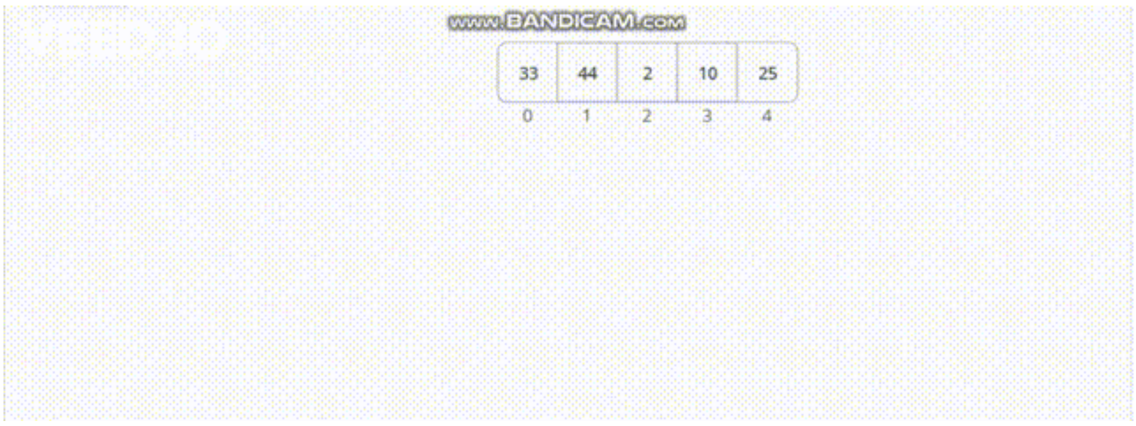
**5. Identify the tracing steps of merge sort and quicksort and analyse the time complexity for the following data: 33, 44, 2, 10, 25**

**Quick Sort:**



Time Complexity:

| Case | Time Complexity |
|---|---|
| Best Case | O(n*logn) |
| Average Case | O(n*logn) |
| Worst Case | O($n^2$) |



Time Complexity:

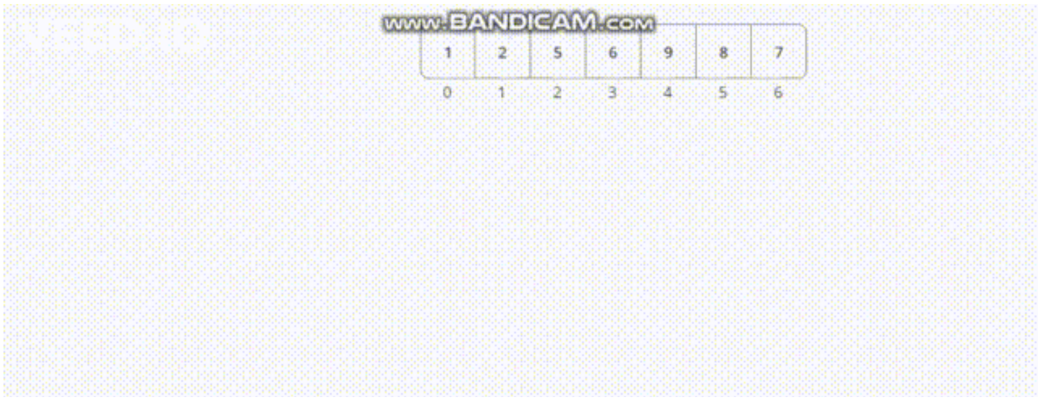
Worst Case Time Complexity [ Big-O ]: **O(n*log n)**

Best Case Time Complexity [Big-omega]: **O(n*log n)**

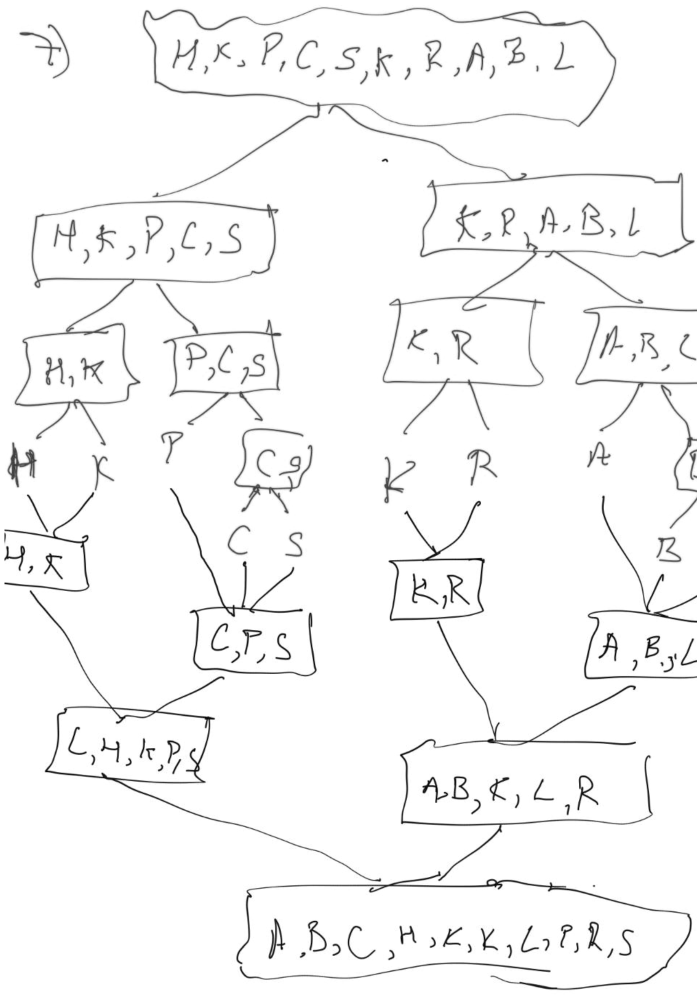
Average Time Complexity [Big-theta]: **O(n*log n)**

Space Complexity: **O(n)**

## 6. Organise the steps in merge sort to arrange following data in non-decreasing order 1,2,5,6,9,8,7



**7. Organise merge sort on following letters H, K, P,C,S,K,R,A,B,L**



**8. Explain Strassen's method outperforms the traditional matrix multiplication method. How many multiplication operations are required during multiplication of two matrices with size of 32 x 32 in Stressen's method.**

Using the traditional method, two matrices (*X* and *Y*) can be multiplied if the order of these matrices are *p × q* and *q × r*. Following is the algorithm.

Algorithm:

```
Matrix-Multiplication (X, Y, Z)
for i = 1 to p do
   for j = 1 to r do
```

```
    Z[i,j] := 0
    for k = 1 to q do
       Z[i,j] := Z[i,j] + X[i,k] × Y[k,j]
```

There are three for loops in this algorithm and one is nested in the other. Hence, the algorithm takes $O(n^3)$ time to execute.

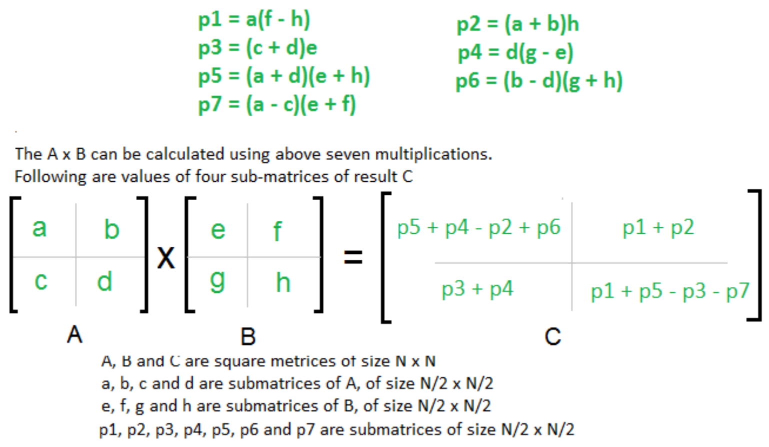
In this context, using Strassen's Matrix multiplication algorithm, the time consumption can be improved a little bit.

Strassen's Matrix multiplication can be performed only on square matrices where n is a power of 2. Order of both of the matrices are n × n, for any order of n x n the algorithm uses only 7 multiplications and 18 additions. Strassen's multiplication divides the nxn matrix into four matrices of n/2 × n/2 and reduces the complexity to $O(n^{2.807})$.

### 9. Explain recurrence relation for Strassen's matrix.

Addition and Subtraction of two matrices takes $O(N^2)$ time. So time complexity can be written as

`T(N) = 7T(N/2) +  O(N`$^2$`)`



p1 = a(f - h)
p2 = (a + b)h
p3 = (c + d)e
p4 = d(g - e)
p5 = (a + d)(e + h)
p6 = (b - d)(g + h)
p7 = (a - c)(e + f)

The A x B can be calculated using above seven multiplications.
Following are values of four sub-matrices of result C

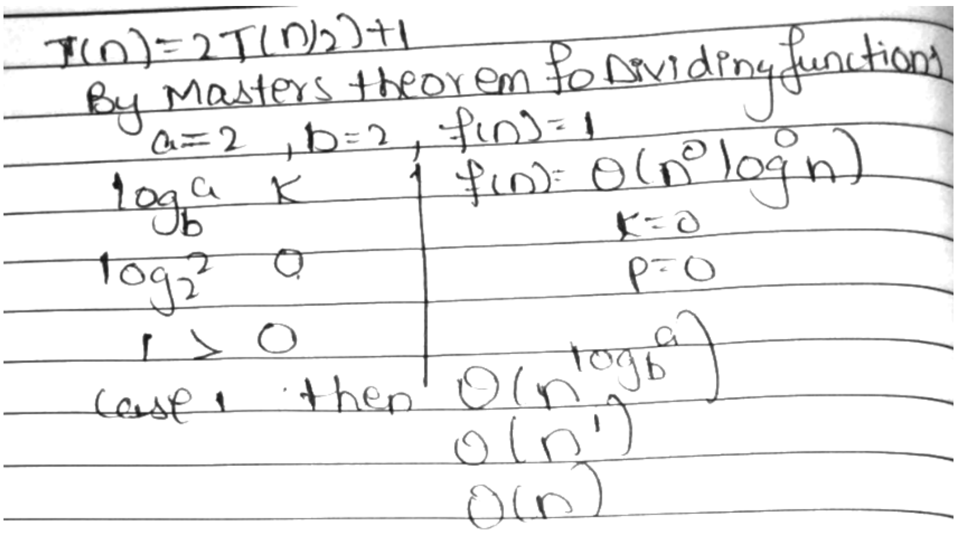
$$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{C}{\begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}}$$

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2
p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

From Master's Theorem, time complexity of above method is

O(NLog7) which is approximately $O(N^{2.8074})$

**10.Solve the following recurrence relation T (n) =2 T (n/2) + 1, and T (1) =2.**



$T(n) = 2T(n/2)+1$

By Masters theorem fo Dividing functions

$a=2$ , $b=2$ , $f(n)=1$

$\log_b a$ $k$ | $f(n) = \Theta(n^0 \log^0 n)$

$\log_2 2$ $0$ | $k=0$

| $p=0$

$1 > 0$

case 1 then $\Theta(n^{\log_b a})$

$\Theta(n^1)$

$\Theta(n)$

## PART B

**1. Define various asymptotic notations used for best case,average case and worst case analysis of algorithms.**

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- O Notation: The notation O(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.
  For example, for a function f(n)

O(f(n)) = { g(n) : there exists c > 0 and $n_0$ such that f(n) ≤ c.g(n) for all n > $n_0$. }

- Ω Notation: The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

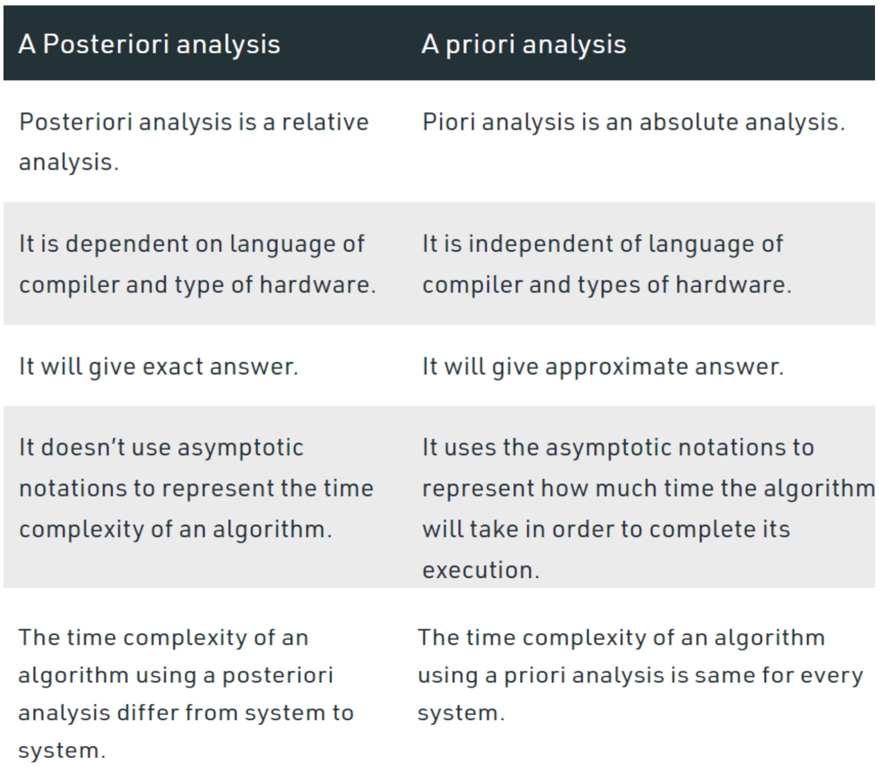
For example, for a function f(n)

$\Omega(f(n)) \geq \{ g(n) :$ there exists $c > 0$ and $n_0$ such that $g(n) \leq c.f(n)$ for all $n > n_0. \}$

- θ Notation: The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time.

$\theta(f(n)) = \{ g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0. \}$

**2. Explain the difference between posteriori analysis and priori analysis.**



| A Posteriori analysis | A priori analysis |
|---|---|
| Posteriori analysis is a relative analysis. | Piori analysis is an absolute analysis. |
| It is dependent on language of compiler and type of hardware. | It is independent of language of compiler and types of hardware. |
| It will give exact answer. | It will give approximate answer. |
| It doesn't use asymptotic notations to represent the time complexity of an algorithm. | It uses the asymptotic notations to represent how much time the algorithm will take in order to complete its execution. |
| The time complexity of an algorithm using a posteriori analysis differ from system to system. | The time complexity of an algorithm using a priori analysis is same for every system. |

**3. Explain Binary search algorithm and analyse its time complexity.**

Algorithm

```
Begin
Set beg = 0
Set end = n-1
Set mid = (beg + end) / 2
while ( (beg <= end) and (a[mid] ≠ item) ) do
    if (item < a[mid]) then
        Set end = mid - 1
    else
        Set beg = mid + 1
    endif
    Set mid = (beg + end) / 2
endwhile
if (beg > end) then
    Set loc = -1
else
    Set loc = mid
endif
End
```

Time Complexity

At **Iteration 1**,
Length of array = **n**
At **Iteration 2**,
Length of array = $n/2$
At **Iteration 3**,
Length of array = $(n/2)/2$ = $n/2^2$
Therefore, after **Iteration k**,
Length of array = $n/2^k$
Also, we know that after
After k iterations, the **length of array becomes 1**
Therefore
Length of array = $n/2^k$**=1**
=> **n = $2^k$**

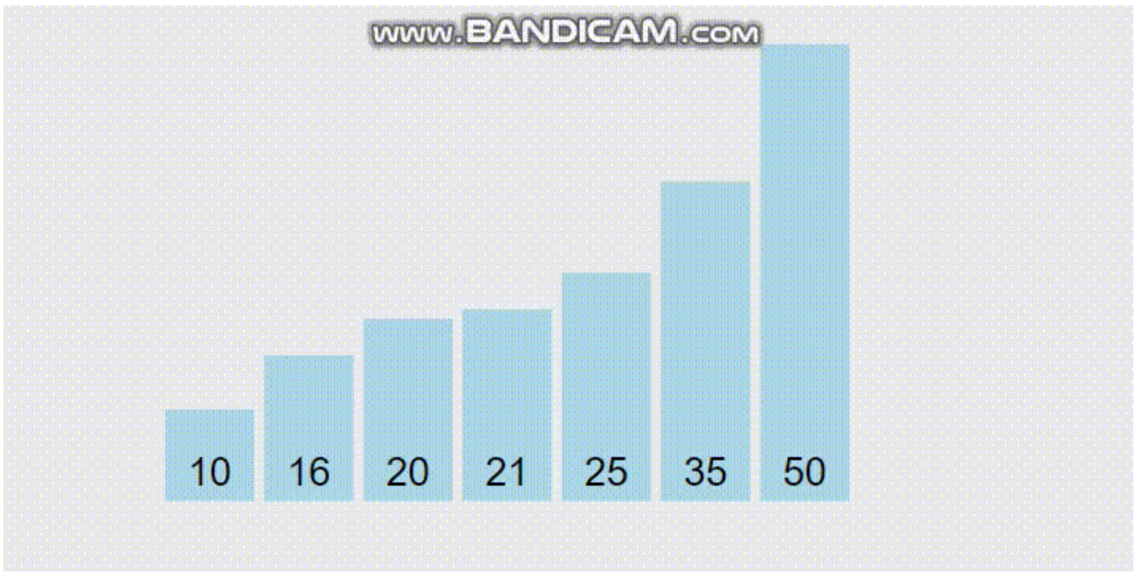
- Applying log function on both sides:
- **$\log_2 (n) = \log_2 (2^k)$**
- **$\log_2 (n) = k \log_2 (2)$**

As **($\log_a (a) = 1$)**
Therefore,
**$k = \log_2 (n)$**

**4. Explain quick sort algorithm and simulate it for the following data: 20, 35, 10, 16, 54, 21, 25**

Algorithm

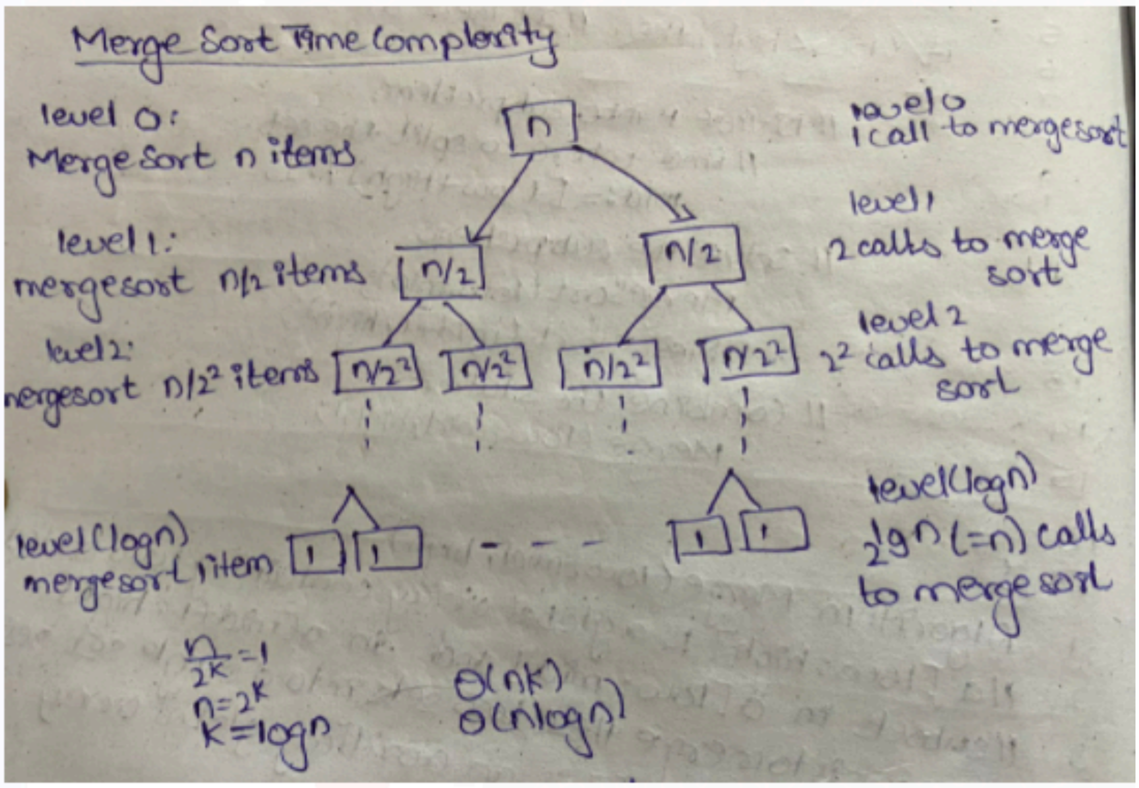
```
quickSort(array, leftmostIndex, rightmostIndex)
  if (leftmostIndex < rightmostIndex)
    pivotIndex <- partition(array, leftmostIndex, rightmostIndex)
    quickSort(array, leftmostIndex, pivotIndex - 1)
    quickSort(array, pivotIndex, rightmostIndex)
partition(array, leftmostIndex, rightmostIndex)
  set rightmostIndex as pivotIndex
  storeIndex <- leftmostIndex - 1
 for i <- leftmostIndex + 1 to rightmostIndex
 if element[i] < pivotElement
    swap element [i] and element[storeIndex]
    storeIndex++
  swap pivotElement and element[storeIndex+1]
return storeIndex + 1
```



**5. Define iterative binary search algorithm.**

Same as the 3rd Question. Click here if you're viewing in Google Docs.

**6. Illustrate merge sort algorithms and discuss time complexity in both worst case and average cases.**


Merge Sort Time Complexity
level 0: Merge Sort n items
level 1: mergesort n/2 items
level 2: mergesort n/2^2 items
level (logn) mergesort 1 item
n
n/2
n/2
n/2^2
1
level 0 1 call to mergesort
level 1 2 calls to merge sort
level 2 2^2 calls to merge sort
level (logn) 2^lgn (=n) calls to merge sort
n/2^k = 1
n = 2^k
k = logn
O(nk)
O(nlogn)

**7. Explain the advantage of Strassen's matrix multiplication when compared to normal matrix multiplication for any two 16 x 16.**

**Refer Part A 8th Ques.**

**8. Explain amortised analysis and discuss how amortised complexity and actual complexity related.**

Amortized Analysis is used for algorithms where an occasional operation is very slow, but most of the other operations are faster. In Amortized Analysis, we analyse a sequence of operations and guarantee a worst case average time which is lower than the worst case time of a particular expensive operation.

The example data structures whose operations are analysed using Amortized Analysis are Hash Tables, Disjoint Sets and Splay Trees.

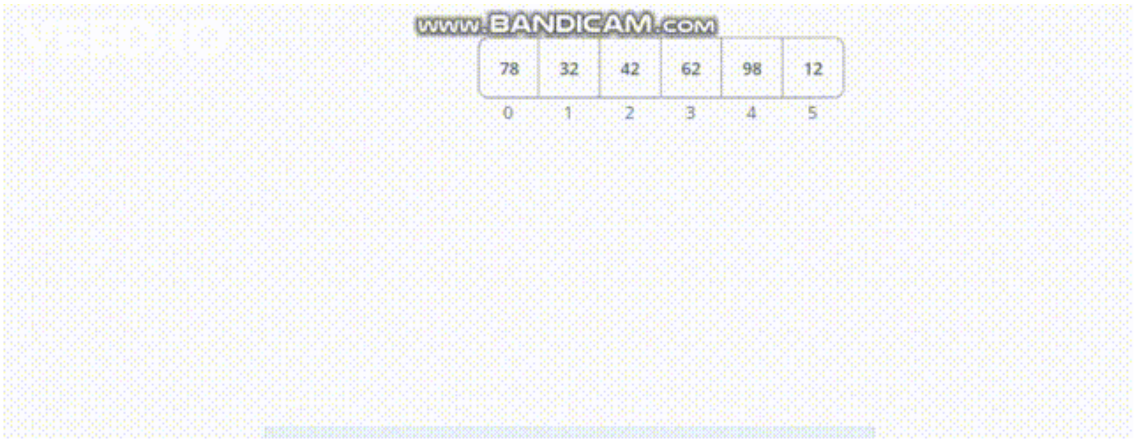
In the Hash-table, most of the time the searching time complexity is O(1), but sometimes it executes O(n) operations. When we want to search or insert an element in a hash table for most of the cases it is constant time taking the task, but when a collision occurs, it needs O(n) times operations for collision resolution.

Classical asymptotic analysis gives worst case analysis of each operation without taking the effect of one operation on the other, whereas amortized analysis focuses on a sequence of operations, an interplay between operations, and thus yielding an analysis which is precise and depicts a micro-level analysis.

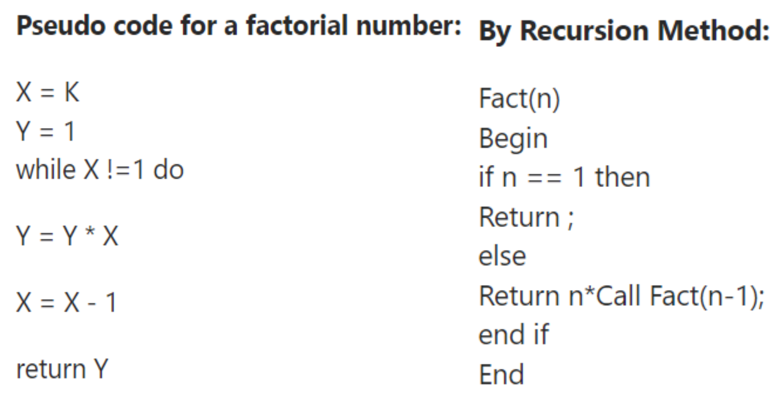
**9. Define probabilistic analysis and randomised algorithms.**

- In analysis of algorithms, probabilistic analysis of algorithms is an approach to estimate the computational complexity of an algorithm or a computational problem. It starts from an assumption about a probabilistic distribution of the set of all possible inputs. This assumption is then used to design an efficient algorithm or to derive the complexity of a known algorithm. This approach is not the same as that of probabilistic algorithms, but the two may be combined.
- A randomised algorithm is an algorithm that employs a degree of randomness as part of its logic or procedure. The algorithm typically uses uniformly random bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the "average case" over all possible choices of random determined by the random bits; thus either the running time, or the output (or both) are random variables.

**10. Organise sorted list of numbers using merge sort: 78, 32, 42, 62, 98, 12.**



**13. Define the Pseudo code conventions for specifying algorithms of recursive and an iterative algorithm to compute n!.**



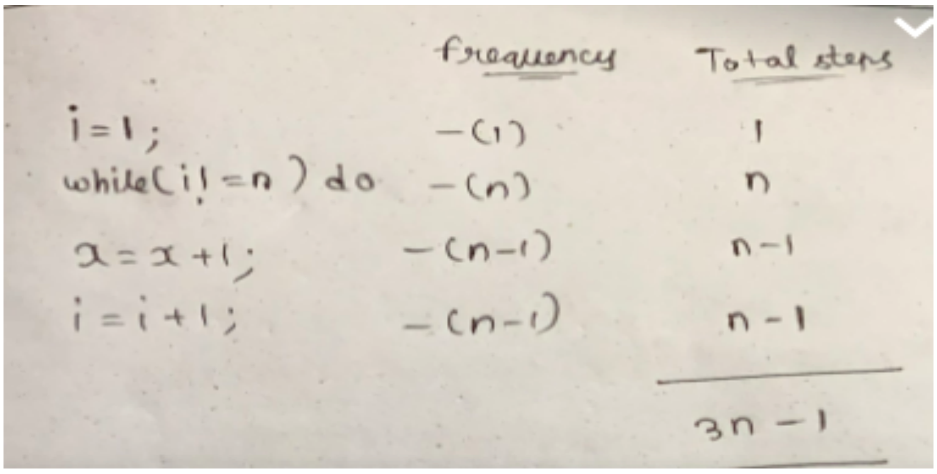
**Pseudo code for a factorial number:**

```
X = K
Y = 1
while X !=1 do

Y = Y * X

X = X - 1

return Y
```

**By Recursion Method:**

```
Fact(n)
Begin
if n == 1 then
Return ;
else
Return n*Call Fact(n-1);
end if
End
```

**14. Explain the frequency counts for all statements in the following algorithm segment. i=1; while(i¡=n) do x=x+1; i=i+1.**



| | frequency | Total steps |
|---|---|---|
| i=1; | –(1) | 1 |
| while(i!=n) do | –(n) | n |
| x=x+1; | –(n-1) | n-1 |
| i=i+1; | –(n-1) | n-1 |
| | | 3n – 1 |

**15. What is a stable sorting method? Is merge sort a stable sorting method? Justify**

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input unsorted array. Some Sorting Algorithms are stable by nature like Insertion Sort, Merge Sort and Bubble Sort etc.

Sorting algorithms are not stable like Quick Sort, Heap Sort etc.

- Merge sort is a stable algorithm but not an in-place algorithm. It requires extra array storage.
- The same element in an array maintains their original positions with respect to each other.
- Overall time complexity of Merge sort is O(nLogn).
- It is more efficient as it is in the worst case also the runtime is O(nlogn)The space complexity of Merge sort is O(n). This means that this algorithm takes a lot of space and may slower down operations for the last data sets.

**16. What is the Bubble sorting method? Is bubble sort a stable sorting method? Justify**

- Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.
- BubbleSort is stable.
- Stable means two equal elements are in the same order before and after the sorting.
- The list 1,1,3,2 is sorted with BubbleSort by only swapping the 3 and the 2. So the two ones are still in the same order.

**17. What is Quick sort? Is quicksort the best sorting method? Justify**

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

The time complexity of Quicksort is O(n log n) in the best case, O(n log n) in the average case, and $O(n^2)$ in the worst case. But because it has the best performance in the average case for most inputs, Quicksort is generally considered the “fastest” sorting algorithm.

Quicksort is a common one for two reasons:

1) it is in-place, i.e. it does not need extra memory when sorting a huge list

2) it performs great on average.

The time complexity of Quicksort is O(n log n) in the best case, O(n log n) in the average case, and $O(n^2)$ in the worst case. But because it has the best performance in the average case for most inputs, Quicksort is generally considered the “fastest” sorting algorithm.

**18. What is the Bubble sorting method? Is bubble sort a stable sorting method? Justify**

**A)** refer q.no.16. click here if you are using Google Docs!

**19. Compare different asymptotic notations**

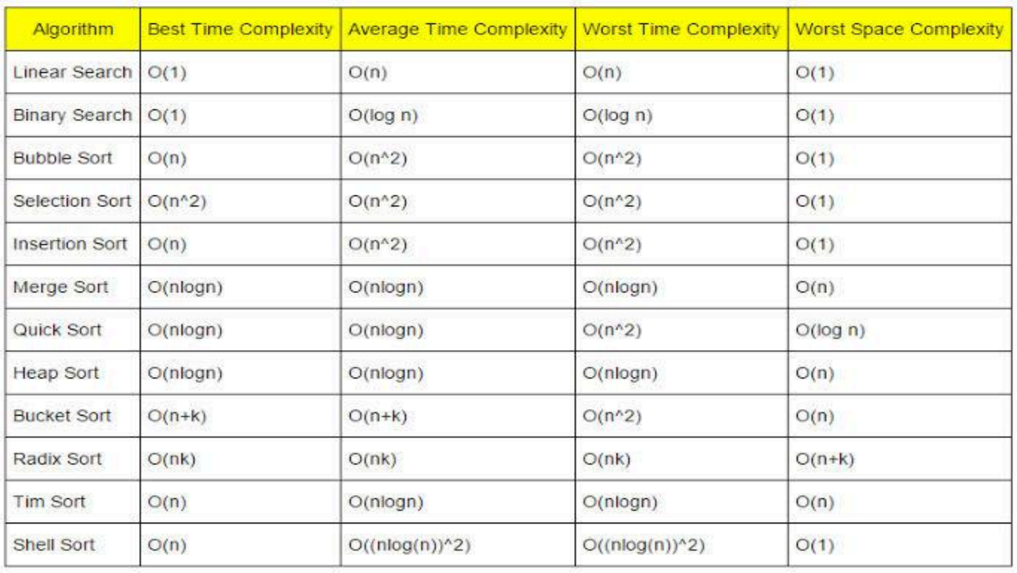
**A)** refer Q.no.1 click here if you are using Google Docs!

***continuation....***

4. Little o notation: is used to describe an upper bound that cannot be tight. In other words, the loose upper bound of f(n).

5. Little Omega (ω) is a rough estimate of the order of the growth whereas Big Omega (Ω) may represent exact order of growth.

**20. Differentiate time and space complexity? Justify**



| Algorithm | Best Time Complexity | Average Time Complexity | Worst Time Complexity | Worst Space Complexity |
|---|---|---|---|---|
| Linear Search | O(1) | O(n) | O(n) | O(1) |
| Binary Search | O(1) | O(log n) | O(log n) | O(1) |
| Bubble Sort | O(n) | O(n^2) | O(n^2) | O(1) |
| Selection Sort | O(n^2) | O(n^2) | O(n^2) | O(1) |
| Insertion Sort | O(n) | O(n^2) | O(n^2) | O(1) |
| Merge Sort | O(nlogn) | O(nlogn) | O(nlogn) | O(n) |
| Quick Sort | O(nlogn) | O(nlogn) | O(n^2) | O(log n) |
| Heap Sort | O(nlogn) | O(nlogn) | O(nlogn) | O(n) |
| Bucket Sort | O(n+k) | O(n+k) | O(n^2) | O(n) |
| Radix Sort | O(nk) | O(nk) | O(nk) | O(n+k) |
| Tim Sort | O(n) | O(nlogn) | O(nlogn) | O(n) |
| Shell Sort | O(n) | O((nlog(n))^2) | O((nlog(n))^2) | O(1) |

## PART-C

**1.Define the term algorithm**

**A)** An algorithm is a set of steps of operations to solve a problem performing calculation, data processing, and automated reasoning tasks.

**2.Define order of an algorithm.**

**A)** Order of growth of an algorithm is a way of saying/predicting how execution time of a program and the space/memory occupied by it changes with the input size.

**3. List asymptotic notations for big ‘Oh’, omega and theta.**

**A)** refer  1ans part b click here if you are using Google Docs!

**4.What do you mean by probability analysis?**

**A)** refer 9 ans part b click here if you are using Google Docs!

**5. Find The best case and worst case analysis for linear search.**

**A)** Linear Search

- Best case: O(1)
- Average case: O(n)
- Worst case: O(n)

**7. Define the recurrence equation for the worst case behaviour of merge sort.**

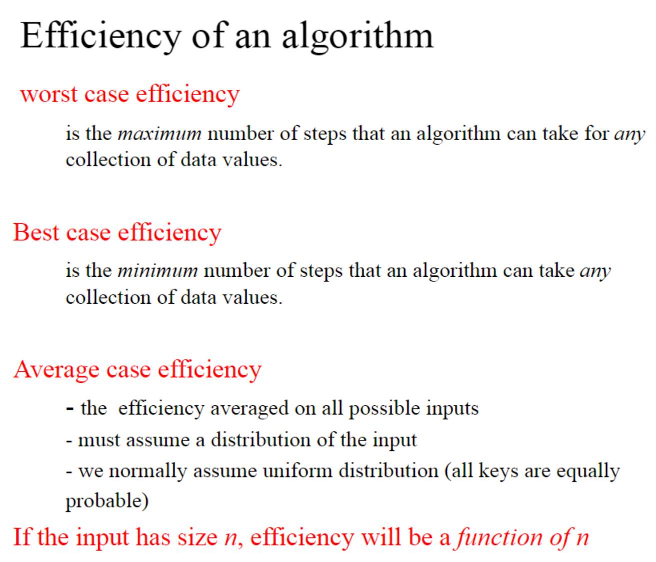
**A)** If T(n) is the time required by merge sort for sorting an array of size n, then the .

**8.Find the average case time complexity of quick sort.**

**A)** O(n log n)

**9.Define algorithm correctness.**

**A)** A correct algorithm is one in which every valid input instance produces the correct output. The correctness must be proved mathematically.



## Efficiency of an algorithm

worst case efficiency

is the *maximum* number of steps that an algorithm can take for *any* collection of data values.

Best case efficiency

is the *minimum* number of steps that an algorithm can take *any* collection of data values.

Average case efficiency

- the efficiency averaged on all possible inputs
- must assume a distribution of the input
- we normally assume uniform distribution (all keys are equally probable)

If the input has size *n*, efficiency will be a *function of n*

**10.Define best case, average case and worst case efficiency of an algorithm.**

**A) Worst case efficiency:** It is the minimum number of steps that an algorithm can take for any collection of data values.

**Best case Efficiency:** It is the minimum number of steps that an algorithm can take any collection of data values.

**Worst case Efficiency:**

- the efficiency averaged on all inputs
- must assume a distribution of the input
- we normally assume uniform distribution(**all keys are equally portable**)

**11. Define the term amortised efficiency.**

**A)** In computer science, amortised analysis is a method for analysing a given algorithm's complexity, or how much of a resource, especially time or memory, it takes to execute. The motivation for amortised analysis is that looking at the worst-case run time can be too pessimistic.

**12. Define order of growth.**

**A)** An order of growth is a set of functions whose asymptotic growth behaviour is considered equivalent.

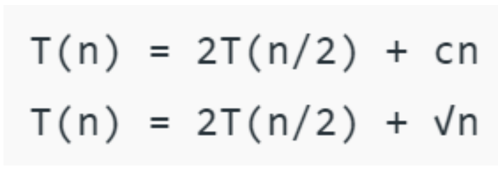
**13. How do you measure the runtime of an algorithm?**

**A)** to calculate the running time, find the maximum number of nested loops that go through a significant portion of the input.

**17. What is meant by divide and conquer? Give the recurrence relation for divide and conquer.**

**A)** In divide and conquer approach, a problem is divided into smaller problems, then the smaller problems are solved independently, and finally the solutions of smaller problems are combined into a solution for the large problem.

Following are some of the examples of recurrence relations based on divide and conquer. //(doubt)



```
T(n) = 2T(n/2) + cn
T(n) = 2T(n/2) + √n
```

**19. Find out any two drawbacks of the binary search algorithm.**

**A)** It employs a recursive approach which requires more stack space.

- Programming binary search algorithms is error prone and difficult.
- The interaction of binary search with memory hierarchy i.e. caching is poor.

**20. Find out the drawbacks of the Merge Sort algorithm.**

**A)** For small datasets, merge sort is slower than other sorting algorithms.

1. For the temporary array, mergesort requires an additional space of O(n).
2. Even if the array is sorted, the merge sort goes through the entire process.