Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Sorting

Insertion Sort:

Small array keep sorting.

Best case $O(n)$
Worst case $O(n^2)$
avarage case $O(n^2)$

Simulation:

12 11 13 5 6

* first make a sub-short array with first two element and sort them

[11 12] 13 5 6

* then take ~~another~~ next element and make another sub array and sort them

[11 12 13] 5 6

keep the process repeating

[11 12 5 13] 6 → [11 5 12 13] 6

Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

5 11 12 13 6

5 11 12 6 13

5 11 6 12 13

5 6 11 12 13

Pseudo Code A: list of sortable items.

```
n = Length(A)
for i = 1 to n-1 do
    j = i
    while j > 0 and A[j-1] > A[j] do
        swap (A[j], A[j-1])
        j = j-1
        end swap
    end for
end procedure.
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Algorithm :

1. Take a single list of sortable items A
2. The variable 'n' is assigned the length of A
3. The outer for loop starts at index '1' and runs for 'n-1' iterations, where 'n' is the length of array.
4. The inner while loop continues to move an element to its left. If an element is smaller than its left element, the elements are swapped
4. The inner while loop starts at the current index i of the outer loop and compares each element to its left. If an element is smaller than its left element, the elements are swapped
5. The inner while loop continues as long as an element is smaller than its left element.
6. Once the inner loop is finished, the element at the current index is in its correct position in the sorted position.
7. The outer loop continues until the array is fully sorted.

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Code

```
for (int i=1; i<=n-1; i++){
    key = arr[i];
  int j = i-1;
  while (j>=0 && arr[j] > key){
arr[j+1] = arr[j];
      j = j-1; }
  arr[j+1] = key;
}
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Selection Sort:

'Swap with minimum'

Time Complexity:

Best case: $O(n^2)$

worst case: $O(n^2)$

Average case: $O(n^2)$

Simulation:

20 12 10 15 2

Set first element as minimum. Compare 2nd element with minimum. if 2nd element smaller than minimum, assign the 2nd element as minimum. keep comparing until last element.

20 12 10 15 2

20 12 10 15 2

20 12 10 15 2

20 12 10 15 2

2 12 10 15 20

swap minimum and first element

Subject _____

Sat Sun Mon Tue Wed Thu Fri

minimum value at index 2 Date : / /

Step-1 → 2 12 10 15 20

12 m

2 12 10 15 20

10 m

2 12 10 15 20

10 m 15 1

2 12 10 15 20

10 m

3 10 12 15 20

Step-3

2 10 12 15 20

12 m

2 10 12 15 20

12 m 15

2 10 12 15 20

step-4: 2 10 12 15 20

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Pseudocode:

```
SelectionSort (Array, size)
 repeat size-1 times
set the first of the unsorted element as the minimum
for each of the ~~first~~ unsorted elements
if element < minimum
set element as new minimum
swap minimum with first unsorted position
end SelectionSort.
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

C++ Code:

```
for (int i=0; i<n-1; i++) {
  int min_idx = i;
  for (int j = i+1; i<n ; i++) {
    if (array[j] < array[min_idx])
      min_idx = i;
    }
    swap (array[min_idx, array[j])
  }
```

Saap Swap

```
int temp;
temp = array[min_idx];
array[min_idx] = array[j];
array[j] = temp;
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

Bubble sort:

Best case: O(n).

worst case: O(n^2).

Average case: O(n^2).

keep swapping ~~to~~ biggest ~~to~~ in last position.

Simulation:

-2 45 0 11 -9

~~keep~~ starting from ~~first~~ zero index compare 1st and second element. if 1st element is smaller ~~to~~ swap the elements. keep swapping until the last element. (2nd with 3rd) (3rd with 4th) .....

-2 45 0 11 -9

-2 ~~45~~ 45 0 11 -9

-2 0 45 11 -9

-2 0 11 45 -9

Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

-2 0 11 -9 45

-2 0 11 -9 45

-2 0 11 -9 45

-2 0 -9 11 45

-2 0 -9 11 45

-2 0 -9 11 45

-2 -9 0 11 45

-2 -9 0 11 45

-2 -9 0 11 45

-9 -2 0 1 1 45

-9 -2 0 11 45

-9 -2 0 11 45

-9 -2 0 11 45

Process will continue 0 to (n-1) time

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Pseudocode

bubblesort (array)

for i = 0 to n-1 times

if leftelement > right element

-swap leftelement and rightelement

end bubblesort.

Code:

```
for (int i = 0 ; i<n ; i++ ) {
 for (int j = 0 ; j<n-i ; j++) {
  if(array [j] = array [j+1]) {
  int temp = array [ j ];
  array [j] = array [j+1];
  array [j+1] = temp; }
    }
}
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

Merge Sort:

Stability yes

Best case: O(nlogn)

Worst case: O(nlogn)

Average case: O(nlogn)

A[P..

* Divide and conquer

A[P.....r]

subindex

A[P.....q]

A[q+1.....r]

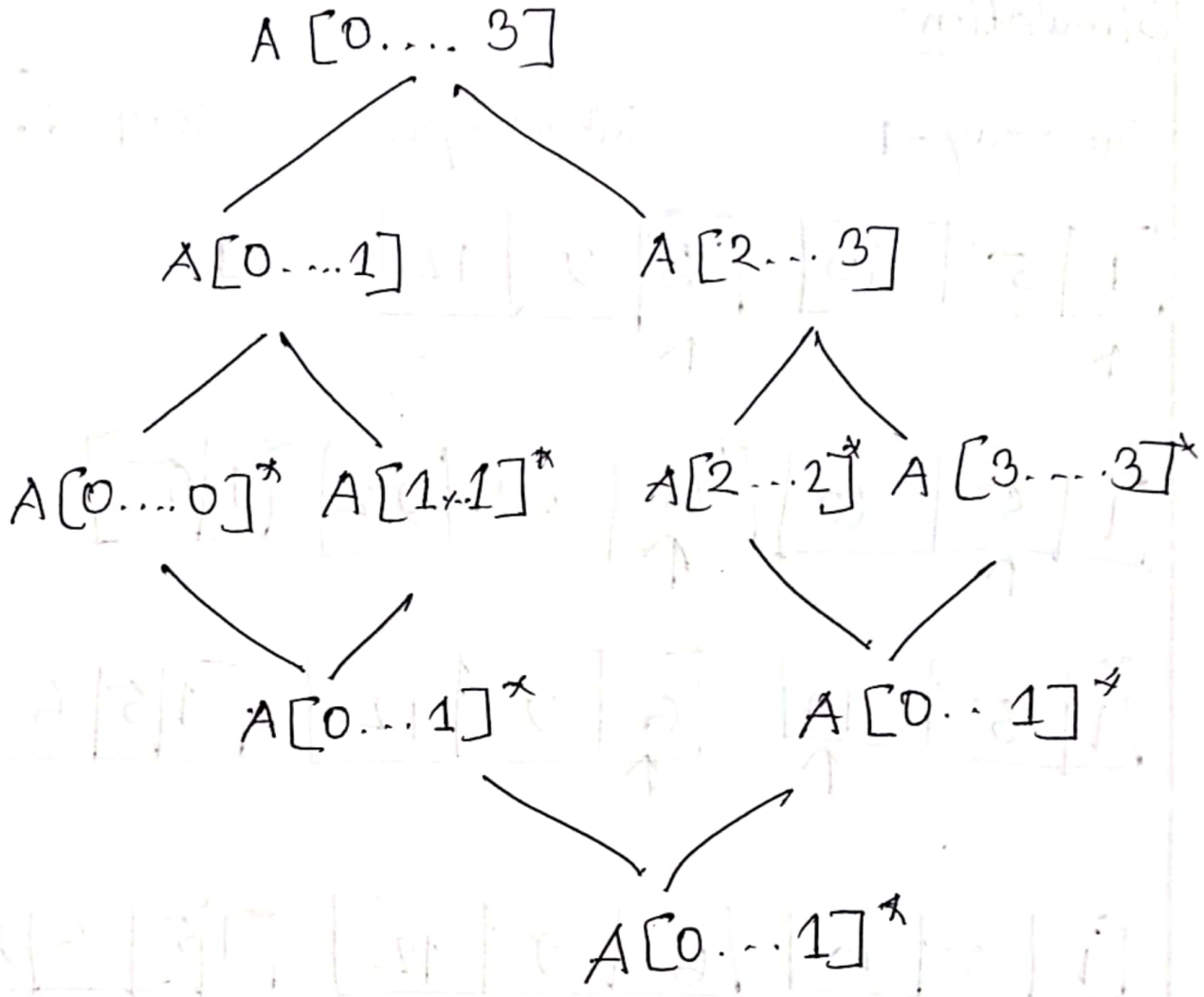
Algo

```
mergesort (A, P, r)
if p > r
   return
   q = (p+r)/2
mergesort (A, P, q)
mergesort (A, q+1, r)
merge (A, P, q, r)
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

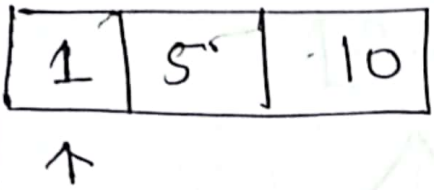
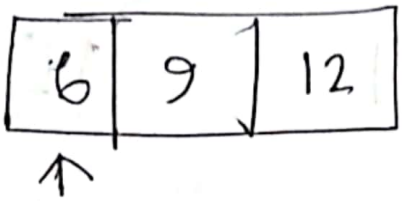
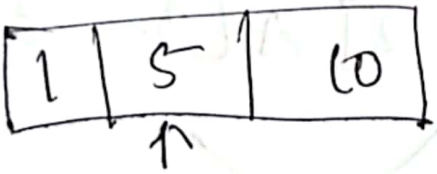
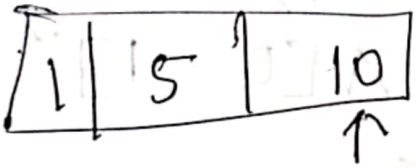
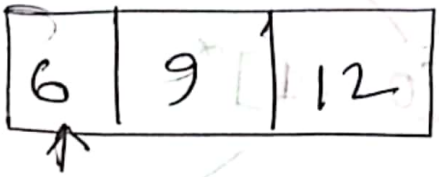
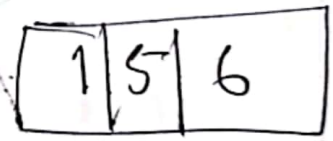
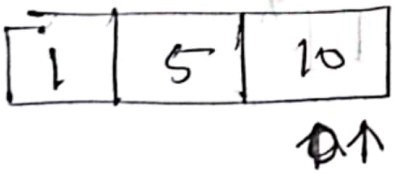
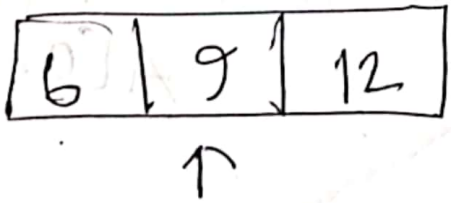
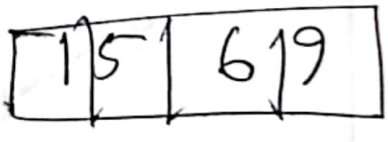
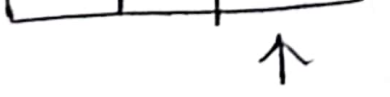
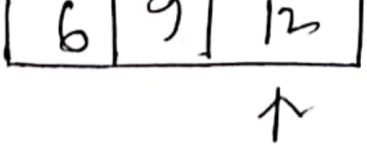
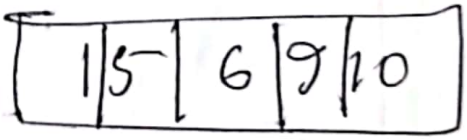


Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

Simulation:

Subarray -1	Subarray 2	Sorted (combined) array
		
		
		
		
		

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Bstnode* insertValue (Bstnode* root, int x) {

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

visualgo.net.

```
void recursion (int n) {
    printf(
    { cout << n << endl; }
    recursion (int n-1);
}
```

```
int recursion (int n) {
    cout << n;
    if (n==0 (n==0) { return 0; }
return recursion (int n-1);
}
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Quicksort:-

* Input → An Array, First element of the array (low), last element of the array (high).
* Output → No return value but when finish the algorithm array will be sorted.
* Step 1 → If the array has less than the element, NO return the array.
* Step 2 → We partitioned the array and last element of the array should suit for for this position.

→ From the first element or low element of the array to (p-1) part then we have to quicksort.

Step-3 → Initialize the PIVOT value.

Subject

Sat Sun Mon Tue Wed Thu Fri

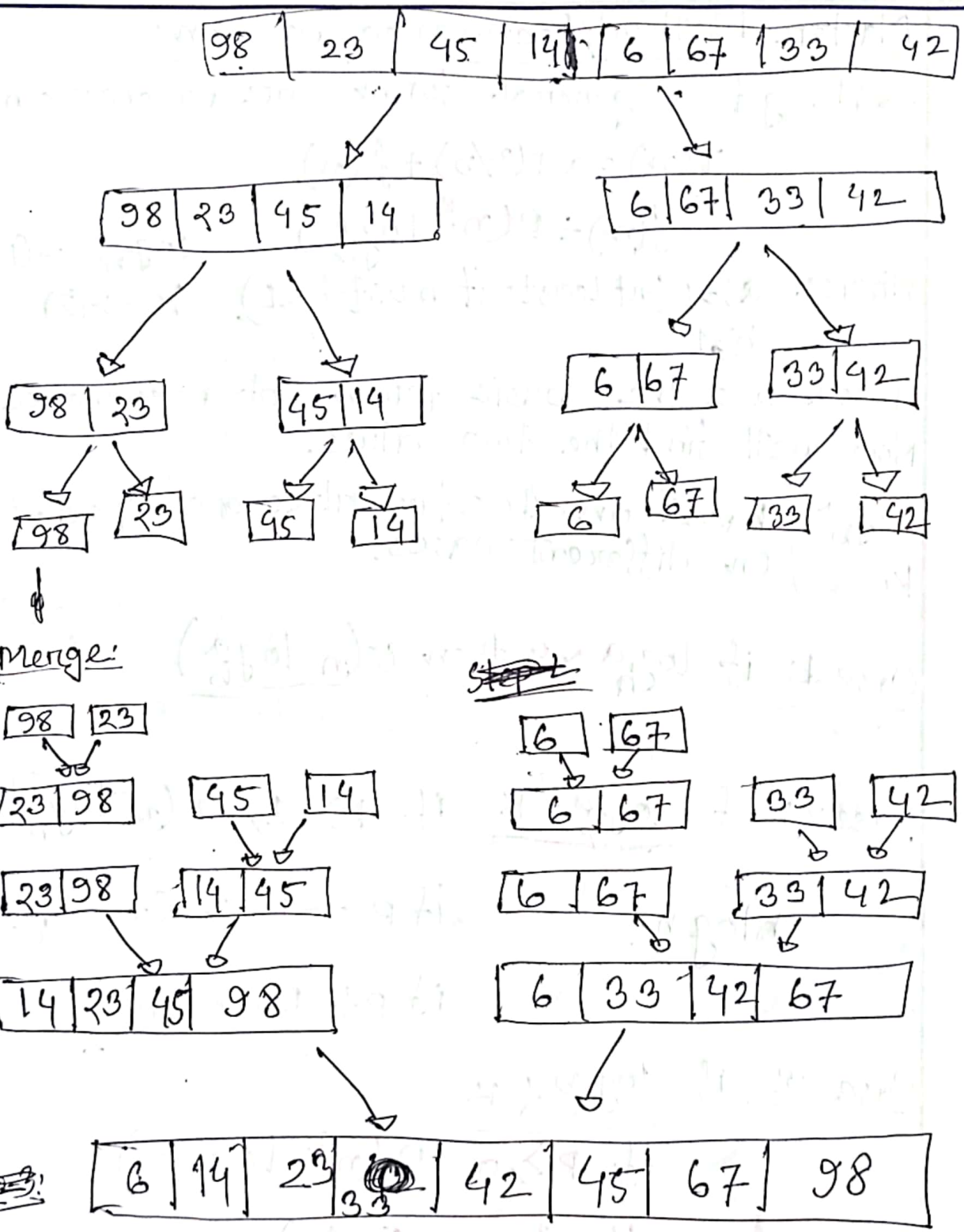
Date : / /

```
int partition (int A[], int low, int high) {
    int pivot, i, j, temp;
    pivot = A[high];
    for (i = low; i <= high; i++) {
        if (A[i] < pivot) {
            for (j = low+1; j >= low; j--) {
                if (A[i] < pivot) {
                    i = i+1;
                    temp = A[j]
                    A[j] = A[i];
                    A[i] = temp;
                }
```

Subject: Merge Sort

Sat Sun Mon Tue Wed Thu Fri

Date : / /



Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Master Method / Recurrance Solution:

→ We get a general form for recurrance Sol^n

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

$$f(n) = \Theta(n^k \log_n^p).$$

$\log_b a \rightarrow (1)$

$k \rightarrow (2)$

Where $a \geq 1$ (atleast it must be 1),

$b > 1$

These are the basic form of recurrance Sol^n.

Now we'll find the two values,

$\log_b a$, k } based on these two values are depends on different cases.

Case 1: if $\log_b a > k$ then $\Theta(n \log_b a)$

Case 2: if $\log_b a = k$ if $p > -1$, $\Theta(n^k \log_n^{p+1})$.

$n \log n$

if $p = -1$, $\Theta(n^k \log \log n)$

if $p < -1$, $\Theta(n^k)$.

Case 3: if $\log_a b < k$

if $p \geq 0$ $\Theta(n^k \cdot \log_n^p)$

if $p < 0$ $\Theta(n^k)$.

Subject _____ Sat Sun Mon Tue Wed Thu Fri

Date : / /

Let us take some recurrence that are satisfied case 1)

$a=2, b=2,$

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

$P = K$

$f(n) = \Theta(n^k \log_n^P)$

$= \Theta(n^0 \log_1^0) =$ (both are zero) $P=K=0$

$= \Theta(1)$

Complexity should consider the case 1.

Solve this ~~comp~~ $f(n)$ based on the case 1.

$T(n) = 8T(n/2) + n$

$\log_b a = \log_2 8 = 3$

$f(n) = \Theta(n^0, n^1, n^2, n^3 \log_n^{P=0,1,2})$

$K = 0, 1, 2.$

$= \Theta(n^3 \log_1^0)$ | $\Theta(n^3 \log_1^1)$

$= \Theta(n^3)$ | $= \Theta($

Best case,

$aT(n/b) + n$

$\Theta(n$

$\log_a^b = 3$

a/b

Qu2 → Insertion, Selection, Merge.
Subject: Simulation.s.

Sat Sun Mon Tue Wed Thu Fri
Date: / /

```
int main() {
int a=5;
int b=10;
int c=a+b;
cout<<c;
}
```

O(1).

} complexity depends on operation }.

for loop O(n).

double for loop O(n)*O(n) = O(n²).

Mid

* dynamic programming. ✓ X
* longest ~~sub~~ common subsequence, (L.C.S) ✓ X
* Matrix chain Multiplication ✓
* ~~Graph~~ ~~Tree~~ * Knapsack algorithm.
* Sorting (Merge / insertion / selection / Quick / linear search / Counting)

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Dynamic Programming

→ Dynamic Programming are used in optimization Problems

→ Dynamic Programming solves the problem is combining the solutions of subproblems.

→ Dynamic programming algorithm solves each sub problem once and then saves it answer. In that case avoiding the works of recomputing.

→ Two types of properties can be given to solve that using The dynamic programming.

* Overlapping sub-problems. approch similare to divide and conquer approach. Dynamic programming also combine the solutions to the sub problems. It is also used to solve the problem using repeated approach.

Example: Fibonacci number have array over-lapping sub problems.

Subject _____

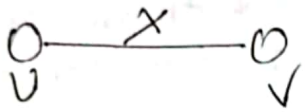
Sat Sun Mon Tue Wed Thu Fri
O O O O O O O

Date : / /

Optimal Sub-Structure:

If the optimal solution of the given problem can be obtained using optimal solutions of it's sub-problems.

Ex: The shortest path problem has the following optimal sub-structure property.



u → x

x → v

Dynamic programming algorithm is used to follow 4 steps.

1. It's optimal Solution.
2. Recursive the values of optimal solutions.
3. Compute the value of optimal solution.
4. Compute an optimal solution from the computed information.

Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

Applications:

Longest common Subsequence.

Matrix Chain Multiplication

Travelling Salesman Problem.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \times B \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

$$C = A \times B$$

$$= \begin{bmatrix} (a_{11} \times b_{11}) + (a_{12} \times b_{21}) + (a_{13} \times b_{31}) & (a_{11} \times b_{12}) + (a_{12} \times b_{22}) + (a_{13} \times b_{32}) \\ a_{21} \times b_{11} + (a_{22} \times b_{21}) + (a_{23} \times b_{31}) & (a_{21} \times b_{12}) + (a_{22} \times b_{22}) + (a_{23} \times b_{32}) \end{bmatrix}$$

Subject Parenthesized

Sat Sun Mon Tue Wed Thu Fri

Date : / /

Matrix Multiplication (A,B):

If columns[A] != rows[B] then
    error "Incompatible dimensions"

else {
    for i=1 to rows[A] do
        for i=1 to column[B], do {
            c[i,j] = 0
            for k=1 to column[A] do
                c[i,j] = c[i,j] + A[i,j] x B[j,i]

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Merge Sort

Pseudocode

bubble sort

```
begin Bubblesort(list)
    for all elements of the list
     if list[i] > list[i+1]
        swap (list[i], list[i+1])
     end if
    end for
  end Bubblesort.
```

```
void bubbsort(int arr[], int n){
    int i,j;
    for (i=0; i<n-1; i++){
      for (j=0;j<n-1-i;j++){
        if(arr[j] > arr[j+1]) {
        swap (arr[j], arr[j+1]);

      }.
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Selection Sort-

```
Start procedure selectionsort (list[], n)
  max = length (list - 1)
  for i from 0 to max
     key = list [i]
     key[j] = i
     for j from i+1 to max
     . if list [j] < key
           key = list [j]
           key [j] = j
     list [key] = list [j]
     list [i] = key
     return list
  end for
  end for
end procedure selectionsort
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Insertion sort:

```
start procedure insertionsort (array [n], n)
  for i from 1 to n
    key = array [i]
    while j >= 0 and a[j] > key
      a[j+1] = a[j]
      j = j - 1
    array [j+1] = key
  end for
end procedure insertionsort;
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Recursion : $f(n) = 2n + 1$ for natural numbers

Basic clause : $f(0) = 1$

Inductive clause $f(n+1) = 2 \cdot f(n) + 2$

Master method

$T(n) = aT(n/b) + f(n)$ $f(n) = \log_b R$

* T(n) function solution :

```
funcA (AarrayA, leftIndex, RightIndex, Increment){
if (leftIndex == RightIndex) return ;
for (i = leftIndex ; i< rightIndex ; i++)
ArrayA[i] = Aarry[i] + incrementfor;
mid = (r+l)/2
funcA (ArrayA, l, m, increment)
funcA ( u , mid, r, increment).
r: ( u , l, m, increment+1);
}.
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Quicksort : O(nlogn)

Algorithm:

1. The Quicksort function is called with input index A, starting index p, ending index r,

if p<r, The PARTITATION sub rou

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Quick sort: (Pivot point) let First element

$n \log n$

Pivot left → small

Pivot right → big

pivot

left++

right++

left < right

$f(n) = n^{\log_b a}$ $\quad n^{\log_b a}$

$f(n) = n^{\log_b a}$ $\quad n \log n$

$f(n) >> n^{\log_b a}$

$T(n) = O(f(n))$

$2T\left(\frac{n}{2}\right) + n$

$2^k T\left(\frac{n}{2^k}\right) +$

Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

left left

4 3 1 9 6 10 2 3

P le right right

left

4	3	1	0	6	9	10	2	3

Merge Sort: (divide and Conqueror)

Repeated Substitute:

Quick Sort

$T(n) = 2T(\frac{n}{2}) + n$

$T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{n}{2}$ 1 in log n

$= 2\left[2T(\frac{n}{4}) + \frac{n}{2}\right] + n$ first substitute

$= 2^2 T(\frac{n}{2^2}) + 2n$

$= 2^2\left[2T(\frac{n}{8}) + \frac{n}{4}\right] + 2n$ — 2nd sub.

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

$2^4\left[T(\frac{n}{2^4})\right] + 4n$

$2^k T(\frac{n}{2^k}) + nk$

Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

$T\left(\frac{n}{2^k}\right) = 1$

$\frac{n}{2^k} = 1$

$2^k = n; \quad k = \log_2 n$

~~$2 \log_2 n + kn$~~

double loop

$T(n)$

$2^{\log_2 n} + \log_2 n \; n$

low $\quad$ more

$= n \log n$

$T(n) =$

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Quick Sort:

```
int partition (int* arr, int low, int high) {
  int left, right, pivot;
  pivot = arr[low];
  left = low;
  right = high;
  while (left < right) {
      while (arr[left] < pivot) {
          left++; }
      while (arr[right] > pivot) {
          right--; }
      if (left < right) { swap (arr[left], right); }
  }
  arr[low] = arr[right];
  arr[right] = pivot;
  return right;
}
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

```
void Quick_Sort (int arr[], int low, int high) {
    int pivot;
    if (low < high) {
        pivot = partition (arr[], low, high);
        quickSort (arr[], low, pivot - 1);
        Quick_Sort (arr[], pivot+1, high);
    }
}
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

```
void Merge(int arr[], int low, int mid, int
                                    high){
int i = low;
int j = mid+1;
int k = 0;
int arr2[high-low+1];
while (i<=mid && j<=high ){
   if arr[i]<arr[j]){
    arr2[k] = arr[i];
    i++;
    k++;}
   else { arr2[k]=arr[j];
          j++;
          k++;}
 }
 while (j<=high)
 { arr2[k] = arr[j]
    j++;
    k++;
 }
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

```
for (i=low; i<=hig; i++){
    arr[i] = arr2[i-low];
  }
}

void mergeSort(int arr[], int low, int high){
int mid = (low+high)/2
if (low<hig){
    mergeSort(arr[], low, ~~high~~ mid);
    mergeSort(arr[], mid+1, high);
    Merge(arr[], ~~mi~~low, mid, high);
}
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Quick Sort Simulation

5 2 4 6 1 3 9 7

pivot = 5, left

partitation the list into two small sub array

low = 5 , high = 7

{2, 4, 1, 3}, [5], {6, 9, 7}

one less than pivot — another higher than pivot.

recursively sort the sub array.

{2, 4, 1, 3}

2 pivot

{1}[2]{4, 3}

1, 2, 3, 4

now sorted

6, 9, 7

6, 9, 7

sub array

6, 7, 9

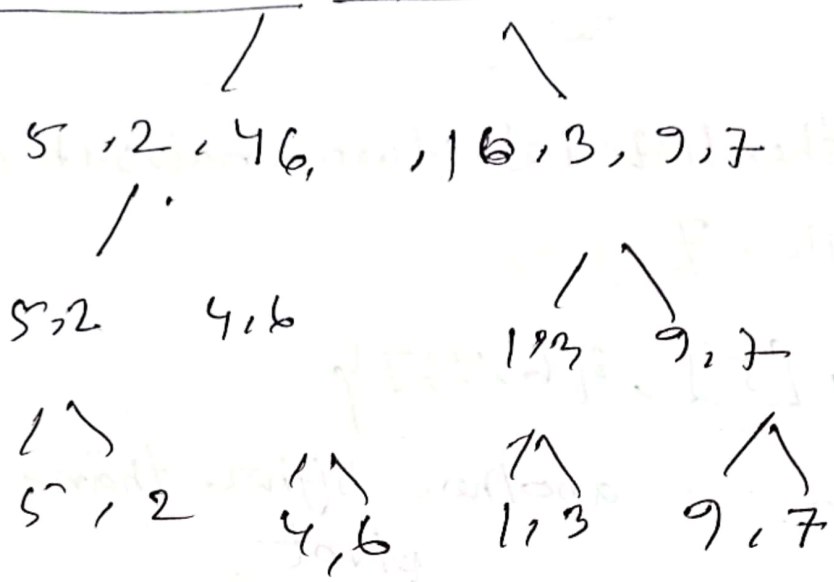
Subject

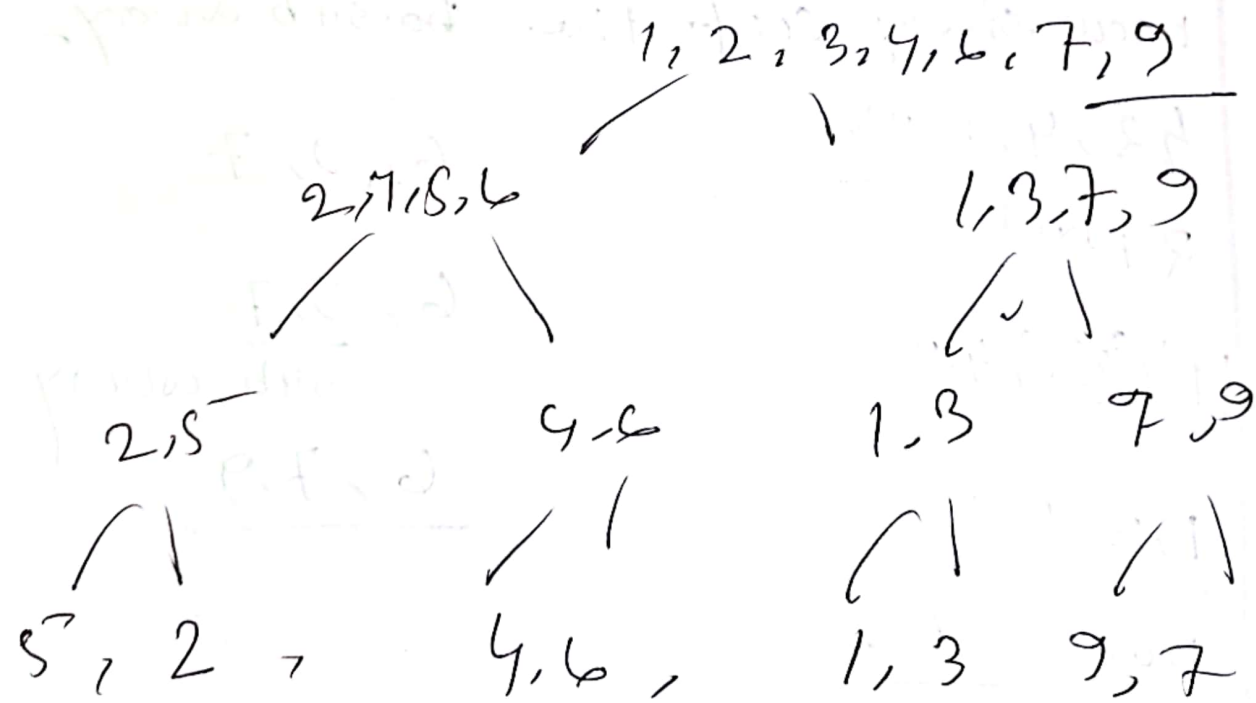
Sat Sun Mon Tue

Date : / /

Merge Sort,

5, 2, 4, 6, 1, 3, 9, 7





Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Merge Sort Time Complexity.

$\Theta(n \log n)$ with set S of n integers and another integer x, two element s, whose sum is x

* Sort the set using merge Sor algorithm,
* Initialize two pointers, left and right,

while left < right

check if the sum of the elements point by the left and right is equal to x.

if yes, return true

if sum less than x, left = left + 1

if sum greater x; right = right - 1,

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Analysing Binary search O(log n)

1. Initialize the search interval by setting the lower bound 'lo' to the first index and upper bound 'hi' to the last.

Analysize → Algorithm, Implement → Code

Recursive (to count zero)

```
int CountZero(int arr[], int size){
  if (size == 0){ return 0; }
  if (arr[0] == 0){
    return 1+ countZero(arr+1, size-1);
  }
  return countZero(arr+1, size-1);
}
```

to find min

```
if (size == 1){ return arr[0]; }
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

$T(n) = 2T(n/2) + O(1)$ recurrence relation

$\log_b^a = \log_2^2 = 1$

$\log_b^a = f(n)$

So, the time complexity is $O(\log(n))$

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

$f(n)$

Using Master method,

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

amount of work outside the recursive calls

$a = 1, b = 2, d = 1$

$\log_2^1 = 0$

$a < b^d$ $O(n^{d \log n}) = O$

Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

Recursive defination of Sum of Integers

$F(0) = 0$ (base case)

$F(n) = n + F(n-1)$ for $n > 0$ (recursive case)

Sum of odd numbers

pseudo code

```
sum_of_odd (n):
  if n == 1;
    return 1
  else;
    return sum_of_odd (n-1) + (2*n - 1);
```

x^n $x^3 = x \times x$

Subject

Sat Sun Mon Tue Wed Thu Fri

Date : / /

Matrix Chain Multiplication : LCM Psadocode-

Matrix

Master Method

$T(n) = 3T(n/4) + n^{2\log_2 n}$

$f(n) = n^{2\log_2 n} = \Theta(n^{\log_2 n})$

$a f(n/b) = 3(n/4)^{2\log_2 (n/4)}$

$a f(n/b) < f(n)$ so,

Time complexity of the algorithm

(a) $O(n^{2\log_2 n})$.

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

$T(n) = 2T(n/3) + n/\log n$

$\log_b a = \log_3 2 = -$ ⊗ $O(n/\log n)$

$4T(n/2) + n^3$ $f(n) = n^c = n^3$

$\log_2 4 = 2$ $O(n^2)$

⊗ $\log_a b <$ ⊙ $n/\log n$

$5T(n/5) + n$ $n\log n$

sub problems = 5

each problem size = $\frac{n}{5}$

$f(n) = n$ (⊗ time complexity of dividing # each problem and combining)

$f(n)$ with n^c compare

$c = \log_5 5 = 1$ $a = \log_b a$

$O(n\log n)$

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

$T(n) = 3T(n/4) + n^2 \log n$

$\log_4 3 = 0.79$ $\quad f(n/b) = (9/16)\, n^2 / \log n - \log 4$

$n^c = n^{\log_4 3}$

$cf(n) = cn^2 / \log n$

$T(n) = \Theta(n) = O(n^2 / \log n)$

$T(n) = 4T(n/2) + \dfrac{n^2}{\log 2n}$

$\log_b a = \log_2 4 = 2$

$f(n) = n^2 / \log 2n = \text{Omega}(n^c)$

Q8 $\quad 2T(n/4) + \sqrt{n}$

$\log_b a = \log_4 2 = \frac{1}{2}$ $\quad O(\sqrt{n})$

$\frac{1}{2} < 0 \sqrt{n}$, so case 3

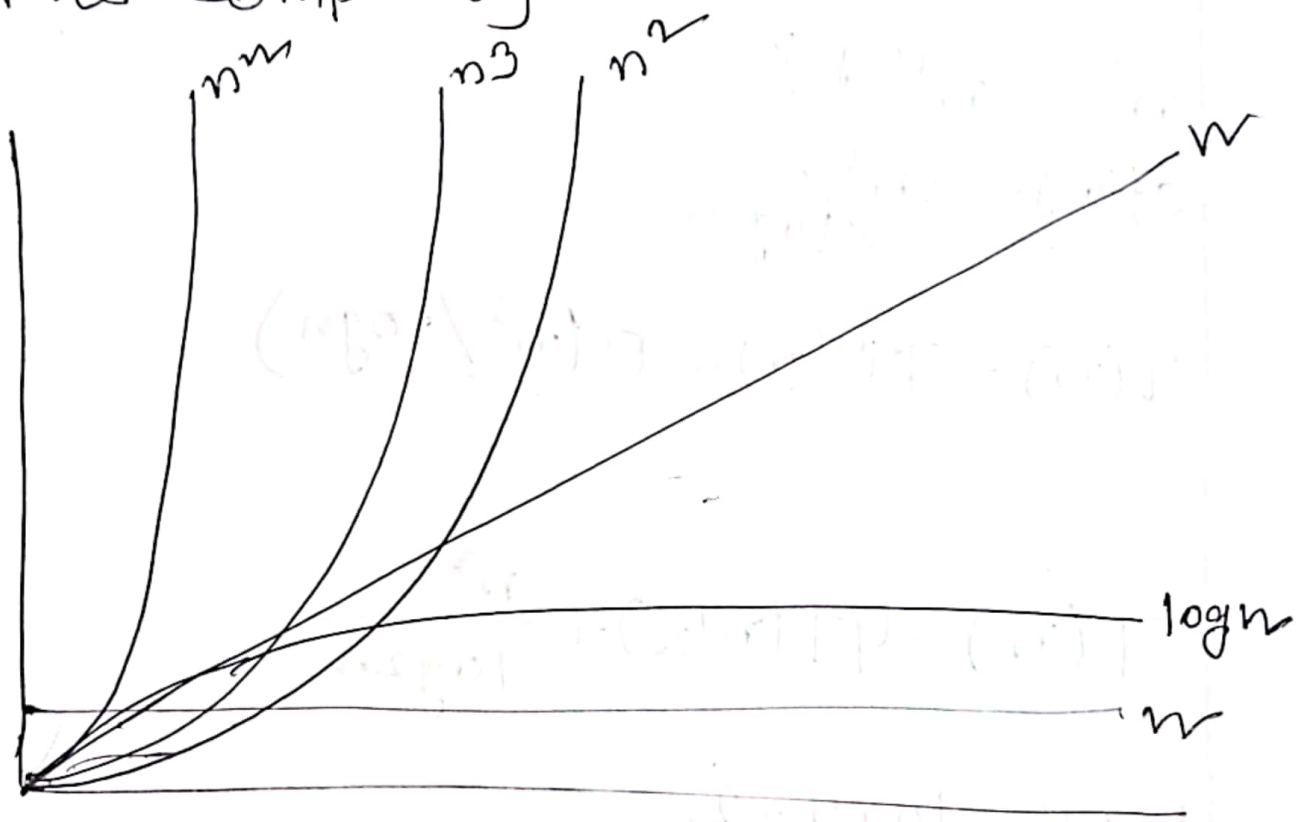
Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

$f(n) - n^{\log_b a}$

Time Complexity



Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Time Complexity

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

$\log_a b = 1$

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

So, time complexity $O(n \log n)$

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

$\log_b a = 2$ $\quad O(n \log n)$

$f(n) = \log_b a$

$\log_3 3 < 1$, polynomially larger !

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Dynamic Problem (Avoid Repeatation)

$T(n-1) + f(n) \Rightarrow O(f(n) \times n)$

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

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

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2 n$

$n \log n$

$k = \log_2 n$

$T(n) = 2^k T(1) + kn$

$= n \times 1 + n \log n$

$= n \log n$

Subject: Master Method

Sat Sun Mon Tue Wed Thu Fri

Date : / /

Q

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

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

if $\log_b a > k$ $O(n^{\log_b a})$

$\log_b a = k$ $O(n \log$

$T(n) = 2T(n/2) + 1$ $\log_b a = 1$

$f(n) = O(1)$

$= O(n^0 \log^0 n)$

$O(n^1)$ $\log_2 4 = 2$

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

$f(n) = O(n^1, k=0, p=0$

$O(n^{0+2})$

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

$T(n) = 8T(n/2) + \cancel{xx} bn^1 \log n$

$T(n)$ $\quad n^k \log_n^p$

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

$\log_2 2 = 1$; $\quad k = 1, \; p = 0$

$O(n \log n)$

if $\log_b a = k$

$O(f(n) \log n)$

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

$T(n) = 2T(n/2) + \frac{n}{\log n}$

$\log_2 2 = 1$, $k = 1$, $p = -1$,

$n \log\log n$

~~$\frac{n}{\log}$~~

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

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

$\log_2 1 = 0$, $k = 2$

$O(n^2)$

$O(n^2 \log n)$

Subject

Sat Sun Mon Tue Wed Thu Fri

Date : / /

$f(n)$ n^0

(n)

$2T(n/2) + 1 \rightarrow O(n)$ Case 1

$T(n) = 4T(n/2) + 1 \rightarrow \rightarrow O(n^2)$

$T(n) = 4T(n/2) + n^1 \rightarrow O(n^2)$

$T(n) = 8T(n/2) + n^2 \rightarrow O(n^3)$

$T(n) = 16T(n/2) + n^2 \rightarrow O(n^4)$

Case - 3

0 $T(n) = 1T(n/2) + n^1$ $O(n)$

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

$T(n) = 2T(n/2) + n^2 \log n \rightarrow O(n^2 \log n)$

$T(n) = 4T(n/2) + n^3 \log^2 n \rightarrow O(n^3 \log^2 n)$

2 $T(n) = 2T(n/2) + \frac{n^2}{\log n} \rightarrow (n^2 \log n)$

$T(n)$ $\frac{n^2 \log n}{\log^2 n}$ $n \log n$

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Case 2 :- n^0

$T(n) = T(n/2) + 1$ $\quad \Theta(\log n)$

$T(n) = 2T(n/2) + n$ $\quad \Theta(n \log n)$

$T(n) = 2T(n/2) + n \log n$ $\quad (n^1 \log^2 n)$

$T(n) = 4T(n/2) + n^2$ $\quad \Theta(n^2 \log n)$

$T(n) = 4T(n/2) + n^2 \log^2 n$ $\quad \Theta(n^2 \log^3 n)$

$T(n) = 2T(n/2) + \frac{n}{\log n}$ $\quad \Theta(n \log \log n)$

$T(n) = 2T(n/2) + \frac{n}{\log^2 n}$ $\quad \Theta(n)$

* Counting Sorting =
* Repeated substitue method =
* bubble sort =

Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /

$T(n) = T(\sqrt{n}) + 1$

$T(n) = T(n^{\frac{1}{2^k}}) + k$

$n^{\frac{1}{2^k}} \quad \frac{n}{2^k} = 1$

$n = 2^k \quad k = \log_2 n$

$n = \log_2 n \quad k = \log\log_2 n$

$O(\log\log_2 n)$

Subject _____ + public private # protected

Sat Sun Mon Tue Wed Thu Fri

Date : / /

Quick Sort:

```
int partition (int arr, int low, int high){
    int left, right, pivot;
    left = low;
    right = high;
while (left < right){
    while (arr[left] < pivot){
        left++;}
    while (arr[right] > pivot){
        right--;}
if (left < right) swap (arr[left], arr[right]);
arr[low] = arr[high]
arr[right] = pivot
return right;
}
```

```
void Quick_Sort (int arr[],
                 int low, int high){
pivot = partition(arr[], low, high);
Quick_Sort(arr[], low, pivot-1);
Quick_Sort(arr
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

```
int Quick_Sort (int arr, int low, int high){
    int left, right, pivot;
    pivot = inarr [low];
    left = low;
    right = high;

while (left < right) {
    if while (arr[left] < pivot) {
            left++;}
    while (arr [right] > pivot) {
            right++;}
if (left < right) { swap (arr [left], arr[right]
                                                }
    arr [low] = arr [high].
    arr [right] = pivot;
return right; }
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

```
void Merge (int *arr, int low, int mid, int high) {

    int i=low;
    int j=mid+1;
    int k=0;
int arr2[high-low+1];
while (i<=mid && j<=high) {
 if(arr[i]<arr[k]) {
   arr2[k]=arr[i];

    i++;
    k++; }
 else {arr2[k]=arr[j];
      j++;
      k++;
 }
while (i<=high) {
      arr2[k]=arr[j];
      j++;
      k++; }
for (i=low; i<=high; i++) {
      arr[i]=arr[i-low]; }
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

```
void Merge (int arr[], int low, int mid, int
                                        high){
    int i = low;
    int j = mid + 1;
    int k = 0
    int arr2[] = int [high - low + 1];
while (~~arr~~ i<=mid && j<=high){
    while (arr[i] < arr[k]){
        arr2[k] = arr[i];
            i++;
            k++;}
else{ arr2[k] = arr[j];
            j++;
            k++;
    }
while (j<high){
        arr2[k] = arr[j];
            j++;
            k++;}
for (int i=0; i<high; arr[i++]){
        arr[i] = arr[i-low];}
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

```
start procedure Merge(arr[], low, mid, high)
  initiate i=low, j=mid+1, k=0, arr2[k] = arr[high-low+1];
  do when i<=mid && j<=high
    do when (arr[i]<arr[k])
        arr2[k]=arr[i];
        i++;
        k++;
    }
  else do arr2[k] = arr[j];
        j++
        k++
  }
  when (j< high)
    do arr2[k] = arr[j]
        j++
        k++ }
  for i=0 to i<high
      do arr[i]= arr[i-low]
  end for
end procedure
```

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Asympotatic Notation

→ It represents the simple form of time complexity.

→ O - Big-oh notation represents the Upper bound notation of the function.

→ Ω - Big Omega notation represents or waves as a lower bound of the function.

→ Θ - Theta Notation works as a average bound of a function.

→ Any functions either upper or lower bound or avarage.

→ when finds the time complexity of any algorithm Time complexity will be one among.

Subject: Binary / linear search
Asymtote

Sat Sun Mon Tue Wed Thu Fri
Date : / /

Big oh

$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 \dots < 2^n < 3^n \dots < n^n$

Let's say:

The function $f(n) = O(g(n))$ if there exist positive constants c and n_0.

Such that $f(n) \leq c * O(g(n))$ for all $n \geq n_0$

$c \cdot g(n)$

if we have a fn eg: $f(n) = 3n+4$

$3n+4 < 5n$ for all $n \geq n_0$

$f(n)$ $\quad$ c $\quad$ $g(n)$

$f(n) = O(n)$

c - values we can write any number but make sure $f(n)$ is less than the value of $c * g(n)$, * we can put any other value

$3n+4 \leq 2n+3n$

$3n+4 \leq 5n$ for all $n \geq n_0$ $\quad O(n)$

* If the function put the different

$3n^2+4 < 3n^4+4n^2$

$3n+4 < 7n^2$ for all $n \geq 1$ $\quad f(n) = O(n^2)$

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

Omega notation Ω

$f(n) = \Omega(g(n))$ if exists positive constant c and n_0 such that $f(n) \geq c * g(n)$ for all $n \geq n_0$

$f(n) = 5n + 3$

$5n + 3 \geq 1 * n$ for all $n \geq 1$

$f(n) = \Omega(n)$

$5n + 3 \geq 1 * \log n$ for all $n \geq 1$ $\quad f(n) = \Omega(\log n)$

$f(n) = 5n + 3$

$f(n) = 5n + 3 = n$, ~~which is~~ we can't write anything beyond the $\log n$

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

theta notation

$f(n) = \theta(g(n))$ iff $\exists$ positive constant c_1 and c_2, and n_0.

$$c_1 * g(n) \leq f(n) \leq c_2 * g(n)$$

$$f(n) = 5n+3$$

$$1 * n < 5n+3 \leq 5n$$

$c_1 = 1$; $f(n) = 5n+3$; $c_2 = 5$

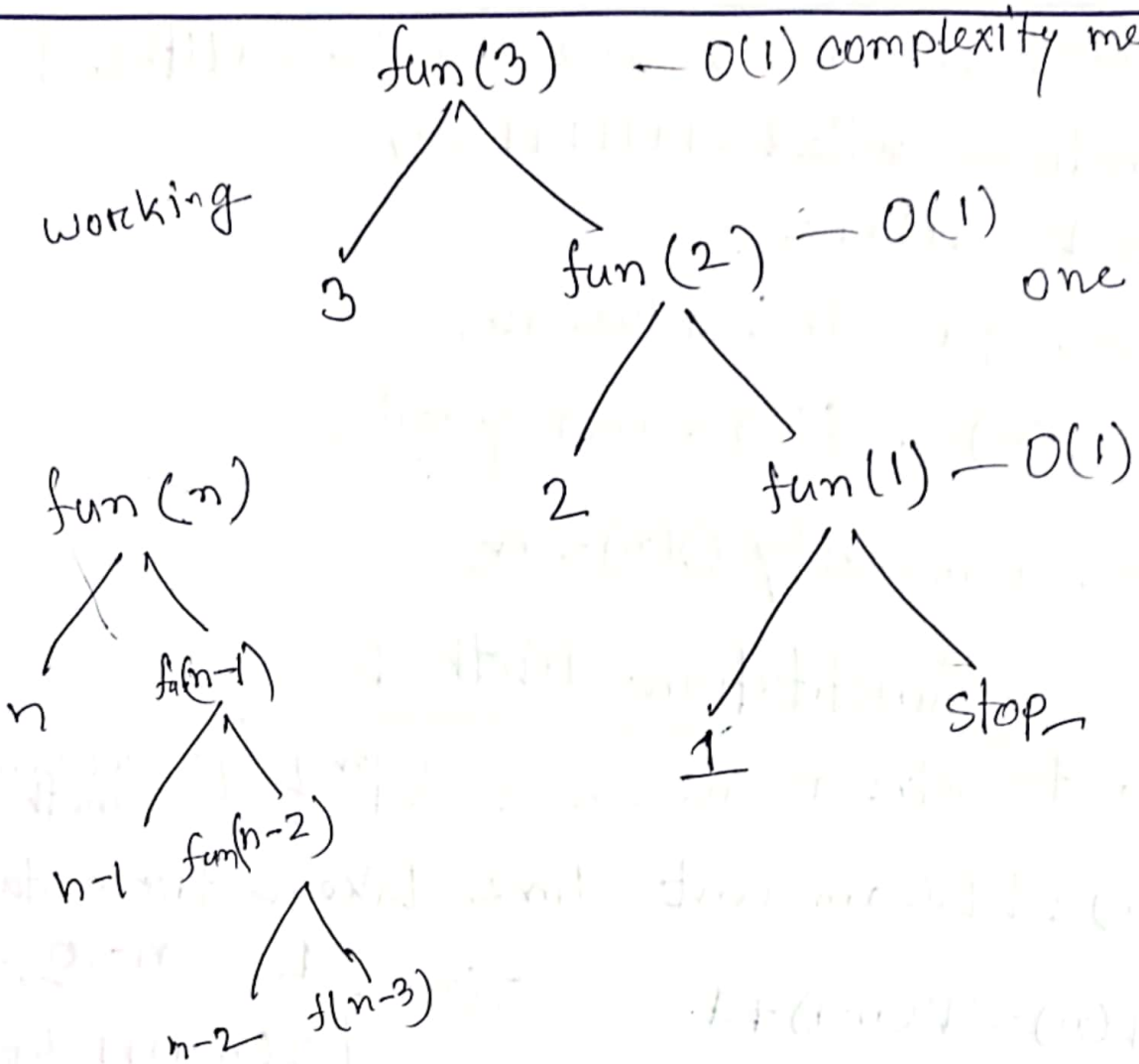
$f(n) = \theta(n)$

$f(n) = 5n+3 = n$ which is the true for less or equal ($\leq$) to the function $f(n)$ for n class. Also is greater than or equal to the $f(n)$ for n class.

Subject _____

Sat Sun Mon Tue Wed Thu Fri

Date : / /



It does not go further. It is called The tracing The Tree, Recursive functions for particular function.

→ Amount of work is called just printing

→ It is doing as 3 times as it's called the value of 4.

⇒ First called and printing.

Subject _____

Sat Sun Mon Tue Wed Thu Fri
○ ○ ○ ○ ○ ○ ○

Date : / /

→ Print the value and again call the function.

function called = 1+1+1+1 = 4

print = 1+1+1 = 3

if we pass the value n.

~~T(n) =~~ f(n) = n+1 ↳ calles

Time complexity O(n) = n

Substitution Method

How to solve recurrance sol^n? [Recurrance method]

T(n) = total amount time taken from statemen

$$T(n) = T(n-1)+1$$

$$T(n) \begin{cases} 1 & n=0; \\ T(n-1)+1 & \text{for all } n>0; \end{cases}$$