Set -2

1. Time - Space Tradeoff is a situation where an algo can be optimized to use either more space to run faster or less space to run slower.
Example - A lookup table. Instead of recomputing a value every time it's needed, a pre-computed table can be stored in memory.

2. They are used to store data that is naturally organized in more than one dimension.
(i) Image Processing - Represented as a 2D array of pixels.
(ii) Game Boards - A chessboard or a tic-tac-toe board can be represented as a 2-D array.
(iii) Matrices

3. Index Formula of 1-D Array :-

Address(A[i]) = BaseAddress + (I - LB) X size
It is significant as it allows for direct & fast access to any element in the array. It enables random access to any element in constant time, making array access highly efficient.

4. 1) Start from 1st element of the Array.
2) Compare the curr. element with the target
3) If element matches return index else move to next
4) Repeat steps 2 and 3 until the end of array.
5) If target is not found, return an indicator (e.g. -1).

Advantages of Linear search :

1) It is simple to implement
2) It doesn't require the data to be sorted
3) It is efficient for small lists

5) Quick sort is a divide-and-conquer algo that sorts the elements in-place. It works by selecting a 'pivot' element from array and partioning the other elements into two sub-arrays. Then they are sorted recursively and repeated until the entire array is sorted.

Example - consider arr : [10, 80, 30, 90, 40, 50, 70]
Let 70 = pivot

i) i starts at -1 and j at 0
ii) iterate j from 0 to second-to-last element
iii) If A[j] <= pivot, i++ and swap A[i] & A[j]
- j=0, A[0] = 10 ≤ 70, i to 0, A[0] with A[0]
Array is = [10 80, 30, 90, 40, 50, 70]
- j = 1, A[1] = 80 > 70, No swap
- j = 2, A[2] = 30 ≤ 70, i to 1, swap A[1] & A[2]
Array = [10, 30, 80, 90, 40, 50, 70]
- j = 3, A[3] = 90 > 70, No swap
- j = 4, A[4] = 40 ≤ 70, i to 2, swap A[2] & A[4]
Array is = [10, 30, 40, 90, 80, 50, 70]
- j = 5, A[5] = 50 ≤ 70 i to 3, swap A[3] & A[5]
Array = [10, 30, 40, 50, 80, 90, 70]

iv) Finally swap pivot with (i+1) = A[4].
Pivot is at its sorted posn.
Array = [10, 30, 40, 50, 70, 90, 80]

6) Types of Recursion :-
1) Direct - A Function calls itself directly.
e.g. fact(n) that calls fact(n-1).
2) Indirect - A Function calls another function, which in turn calls the 1st function
e.g. A() calls B(), and B() calls (A).
3) Tail Recursion
4) Non-Tail Recursion

7) Insertion in DLL :- (Update four ptrs)
i) The next ptr. of new node should point to the next node in the list.
ii) The prev ptr. of new node should point to the previous node.
iii) The next ptr. of previous node should point to the new node.
iv) The prev ptr. of the next node should point to the new node.

Deletion in DLL :- (Update two ptrs.)
i) The next ptr. of the previous node should be updated to point to the node after the one being deleted.
ii) The prev ptr. of the next node should be updated to point to the node before the one deleted.

8) Removal of Recursion by iterative approach.
e.g: fact(n) = n * fact(n-1).

replaced by iterative approach as :

```
function factorial(n):
    result = 1
    for i from 1 to n:
        result = result * i
    return result
```

9).

Feature	Quick Sort	Merge Sort
Worst - Case T.C	$O(n^2)$	$O(n \log n)$
Best - Case T.C	$O(n \log n)$	$O(n \log n)$
Space C	$O(\log n)$	$O(n)$
Stability	Not stable	stable
Real-time suitability	Poor	Better (Guaranteed perf.)

Set-3

1) The amount of resources (time and space) an algo requires to solve a problem. Measured by analyzing the algo's perform as the input size grows expressed using Big O notations describing the upper bound of the growth rate.

2). T.C. measures the amount of time an algo takes to run as a function of the input size. eg.
S.C. measures the amount of memory an algo uses as a function of the input size.

3). The Index - Formula for accessing an element A[i][j][k] in a 3-D array in a Column - major order is :-
$i + j \times D_1 + k \times D_1 \times D_2$
D_1 = no. of rows, D_2 = no. of cols

Address (A[i][j][k]) = Base Add. + (offset × size)
where offset = $i + j \times D_1 + k \times D_1 \times D_2$

4). binarySearch(arr, x, low, high)
{ if high >= low
mid = low + (high - low)/2
if arr[mid] == x
return mid
if arr[mid] > x
return binarySearch (arr, x, low, mid-1)
else
return binarySearch(arr, x, mid+1, high)
else
return -1

5). Bubble sorting is a simple sorting algo that repeatedly steps through the lists, compares adjacent elements, and swaps them if they are in the wrong order. It gets its name

as smaller elements 'bubble' to the top of the list.

Example :
arr = [5, 1, 4, 2, 8].
First Pass :
Compare :
5 & 1 = [1, 5, 4, 2, 8]
5 & 4 = [1, 4, 5, 2, 8]
5 & 2 = [1, 4, 2, 5, 8]
5 & 8 = No swap

Second Pass :-
1 & 4 = No swap
4 & 2 = [1, 2, 4, 5, 8]
4 & 5 = No swap

Third Pass :- No swap
Array = [1, 2, 4, 5, 8]

6)
```
function fact (n)
  if n == 0
    return 1
  else
    return n * fact(n-1)
```

Base case ⇒ n = 0, so fact = 1
else the repetition occurs till the factorial is calculated.

7). Circular Linked List :-
The last node points back to the first node or any other node, creating a circle. This structure allows for continuous traversal of the list without needing to check for a null ptr. at the end.

Algo :-

```
function trav(head)
  if head == null
    return
  current = head
  do
    print(current.data)
  current = current.next
  while current != head
```

8). Fibonacci series :

```
function fibo(n)
  if n <= 1
    return n
  else
    return fibo(n-1) + fibo(n-2)
```

9). Real-World Applications of Merge-Sort.
1). External Sorting
2). Linked List
3) Parallel Computing