Assignment-4

Q.1 What is sorting? Explain different types of sorting with an example.

Ans→ Sorting refers to rearrangement of a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure. Sorting means reordering of all the elements either in ascending or in descending order.

Types of Sorting →
- Selection Sort
- Bubble Sort
- Insertion Sort
- Quick Sort
- Merge Sort

① Selection Sort :

In Selection Sort, the smallest element is selected from the unsorted portion of the list & swaps it with the first element of the unsorted part. This process is repeated for the remaining unsorted portion until the entire list is sorted.

• Advantage →
- Simple & easy to understand
- works well with small datasets.

• Disadvantage →
- It has a time complexity of O(n^2) in the worst & average case.
- Doesn't work well on large datasets.
- Doesn't preserve the relative order of items with equal keys which means it is not stable.

Eg. arr[] =

64	25	12	22	11
0	1	2	3	4

11 → min. value

0 → 1st position.

0 1 2 3 4 → index

1st pass: 1st position → 64 is swapped by 11.
⇒ 1st position is sorted

11	25	12	22	64

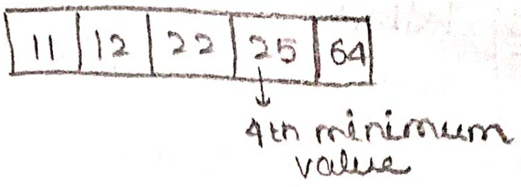
12 → 2nd min value

2nd pass: for 2nd position → 25 is swapped with 12
⇒ 2nd position is sorted

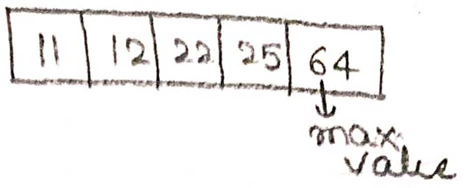
11	12	25	22	64

22 → 3rd min

3rd pass → for 3rd position,
25 is swapped with 22
⇒ 3rd position is sorted.



4th pass → for 4th position,
25 is 4th min value
⇒ 4th position is sorted



5th pass → for 5th position,
64 is 5th min value (max value)
⇒ 5th position is sorted automatically

② Bubble Sort

→ It is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. The algorithm is not suitable for large data sets as its average & worst-case time complexity is quite high.

→ In Bubble Sort,
- ~~[illegible]~~ traverse from left & compare adjacent elements & the higher one is placed at right side.
- In this way, the largest element is moved to the rightmost end at first.
- This process is then continued to find the second largest & place it & so on until the data is sorted.

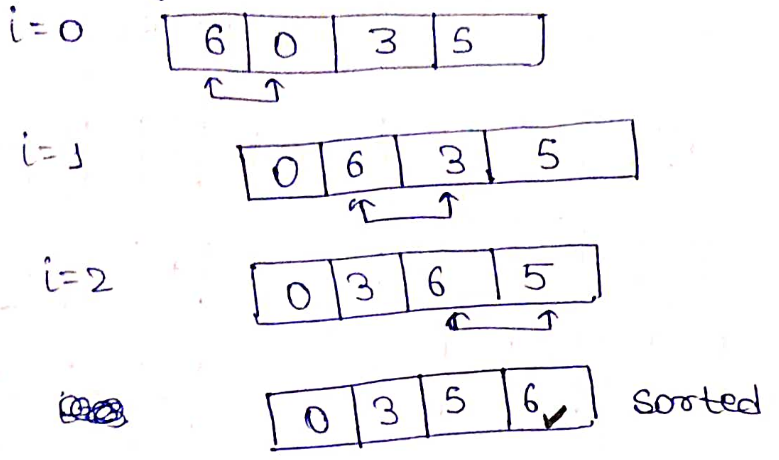
* Advantage
- → It is easy to understand & implement
- → It doesn't require any additional memory space.
- → It is a stable algorithm

* Disadvantage
- → It has a time complexity of O(N2) which makes it very slow for large data sets.
- → It is a comparision - based sorting algorithm.

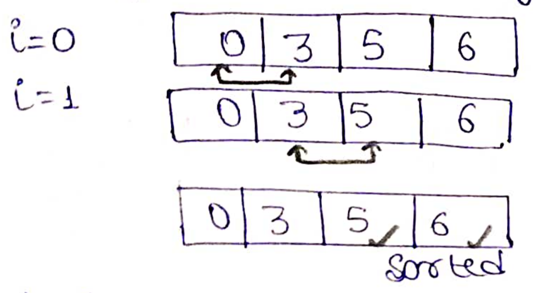
Total steps = $(n-1)$
Total comparisons = $n*(n-1)/2$
} where, n is no. of elements in an array.

eg, Bubble Sort:- arr[] = {6, 0, 3, 5}

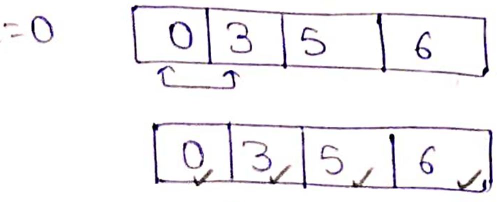
Step 1:- placing the 1st largest element at correct position



Step 2:- placing the 2nd largest element at correct position



Step 3:- placing remaining elements at correct positions



Sorted list.

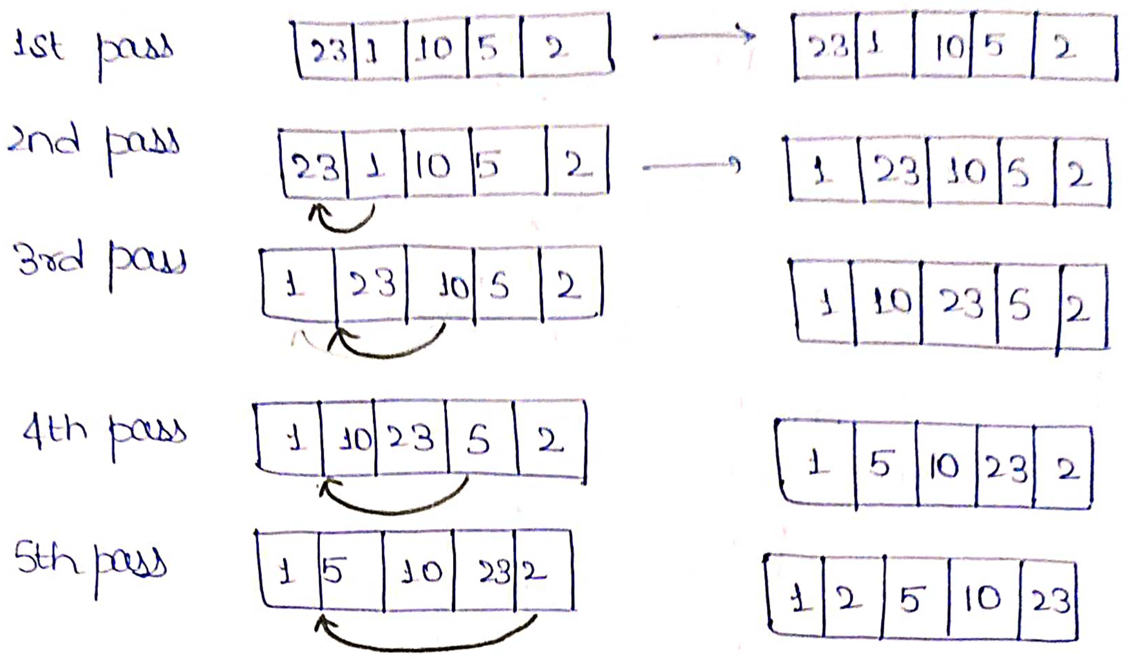
③ Insertion Sort.

→ It is a simple sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list. It is a stable sorting algorithm, meaning that elements with equal values maintain their relative order in the sorted output.

* Advantage
  - → Simple & easy to implement
  - → Stable sorting algorithm
  - → Efficient for small lists & nearly sorted lists.
  - → Space-efficient.

* Disadvantage →
  - Inefficient for large lists.
  - Not as efficient as other sorting algorithms for most cases.

eg: Insertion Sort: A[ ] = {23, 1, 10, 5, 2}



④ Merge Sort :

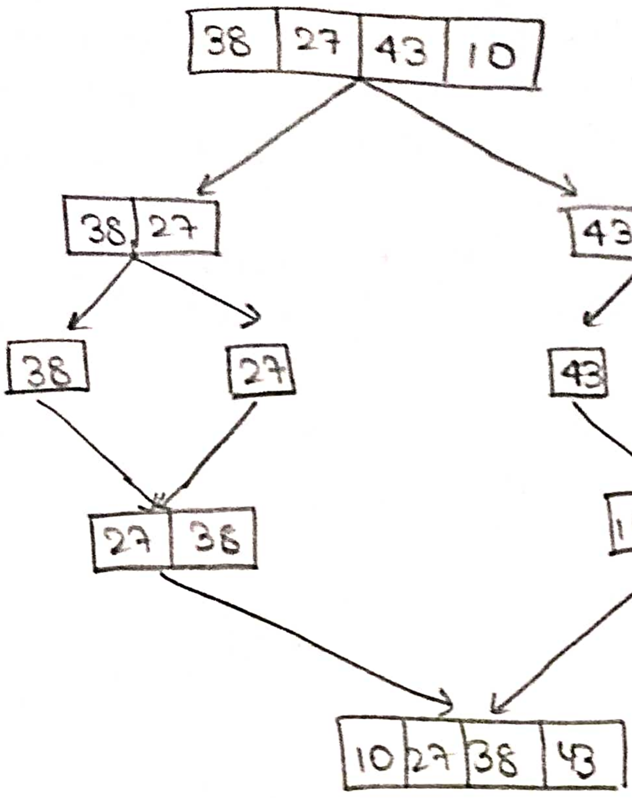
→ Merge Sort is a sorting algorithm that follows the divide & conquer approach.

→ The process of merge sort is to divide the array into 2 halfs, sort each half, & then merge the sorted halves back together. This process is repeated until the entire array is sorted.

* Advantage →
  - It is stable sorting algorithm, ie, It maintains the relative order of equal elements in the input array.
  - It performs well even on large datasets.
  - The divide & conquer approach is straightforward.

* Disadvantage →
  - It requires additional memory to store the merged sub-arrays during the sorting process.
  - It is not in-place sorting algorithm.

eg: Merge sort : A[] = {38, 27, 43, 10}.



5. Quick ~~Merge~~ Sort :

→ It is a sorting algorithm based on the divide & Conquer algorithm that picks an element as a pivot & partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

* Advantage :-
- → It is a divide & conquer algorithm that makes it easier to solve problems.
- → It is efficient on large data sets.
- → It has a low overhead, as it only requires a small amount of memory to function.

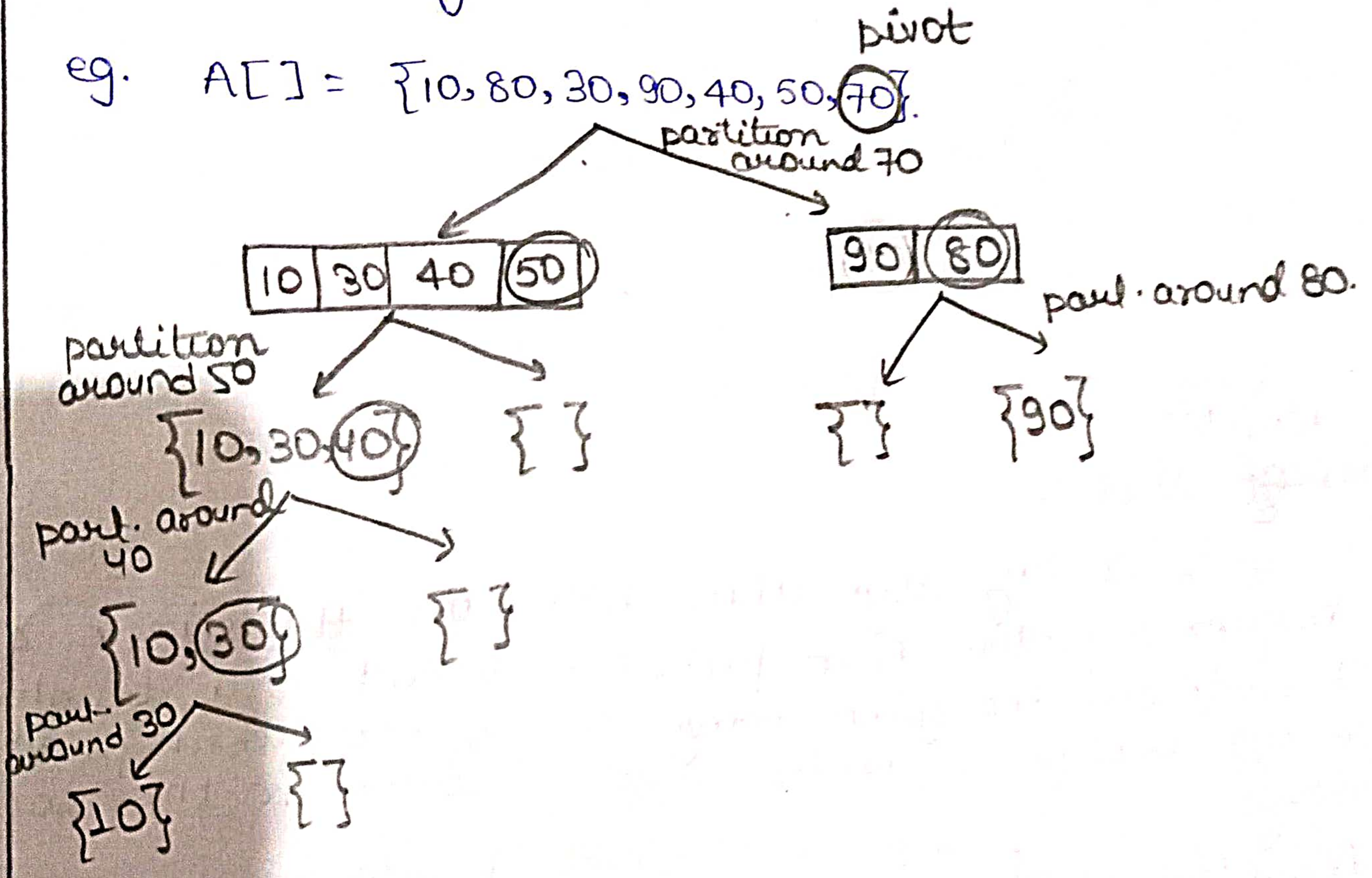
* Disadvantage
- → It has a worst case time complexity, which occurs when the pivot is chosen poorly.
- → It is not a good choice for small data sets
- → It isnot a stable sort.

* Quicksort working:-

→ The key process in quicksort is a partition(). The target of partitions is to place the pivot at its correct position in the sorted array & put all smaller elements to the left of the pivot & all greater elements to the right of the pivot.

→ Partition is done recursively on each side of the pivot after the pivot is placed in its position & this finally sorts the array.

eg. A[] = {10, 80, 30, 90, 40, 50, 70}.



Q.2 What is the searching? Explain Linear search & Binary Search with an example

Ans→ Searching is the fundamental process of locating a specific element or item within a collection of data. The primary objective of searching ~~exists~~ ~~within the~~ is to determine whether the desired element exists within the data, & if so, to identify its precise location to retrieve it.

* Search Techniques → Linear search
→ Binary search

(A) **Linear Search** → It is defined as a sequential search algorithm that starts at one end & gets through each elements of a list until the desired element is found, otherwise the search continu[es] till the end of the data set.

eg, find '20'

0	1	2	3	4	5	6	7	8	→ index
10	50	30	70	80	60	20 ✓	90	40	

→ In linear search algorithm,
- Every element is considered as a potential match for the key & checked for the same.
- If any element is found equal to the key, the search is successful & the index of that element is returned
- If no element is found equal to the key, the search ~~is~~ yields "No match found".

eg. arr [] = {10, 50, 30, 70, 80} and key = 30.

⇒ arr [] =

10	50	30	70	80

Step1: Start with 1st element arr[0] ∵ $arr[0] \neq 30$
⇒ iteration moves to next element

Step2: Compare key with 2nd element ∵ $arr[1] \neq 30$ ⇒ "

Step3:- Compare key with 3rd element ∵ $arr[2] = 30$
⇒ value matched

∴ return the index of element when key is found.

(B) Binary Search.

→ It is defined as a sorting algorithm used in a sorted array by repeatedly dividing the search interval in half.

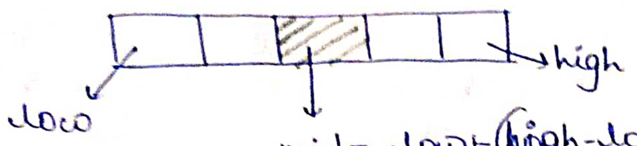
→ Condⁿ: for when to apply binary search in a data structure.

i) the data structure must be sorted.

ii) Access to any element of the data structure takes constant time.

* Binary Search algorithm

→ Divide the search space into 2 halves by finding the middle index "mid".

→ 

mid = low + (high − low)/2

→ Compare the middle element of search space with the key.

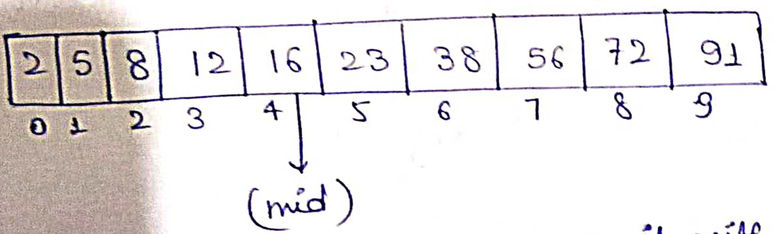
→ If the key is found at middle element, the process is terminated.

→ If the key is not found at middle element, choose which half will be used as the next search space.

- If key is smaller than mid, then left side is used for next search
- If key is greater than mid, then right side is used for next search.

→ This process is continued until the key is found on the total search space is exhausted

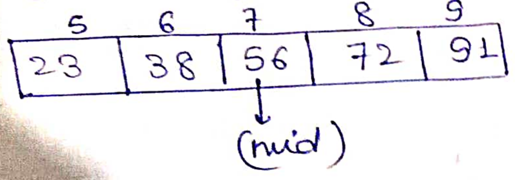
eg. arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91}   target = 23.



Step 1:- Calculate the mid & compare it with target.

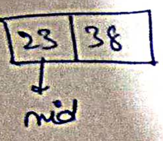
∵ [mid < target] ⇒ right side is used for next search

Now,



∵ [mid > target] ⇒ left side is used for next search

Step 2:- Now,



∵ [mid = target]

⇒ key is found & stop search.