Unit – 4

Searching and Sorting

D D M M Y Y Y Y

Que- Explain Linear search with Algorithm, Pseudo code, Application and Advantages or disadvantages.

⇒

Linear Search :-

Linear search also known as sequential search, is a simple method for finding a target element within a list or array.

Process :-

The search starts at the beginning of the data structure and examines each element in sequence until the target is found or the end of the structure is reached.

Working of Linear search :-

Let the elements of array are -

0	1	2	3	4	5	6
70	40	30	11	57	41	25

Let the element to be searched is

$k = 11$

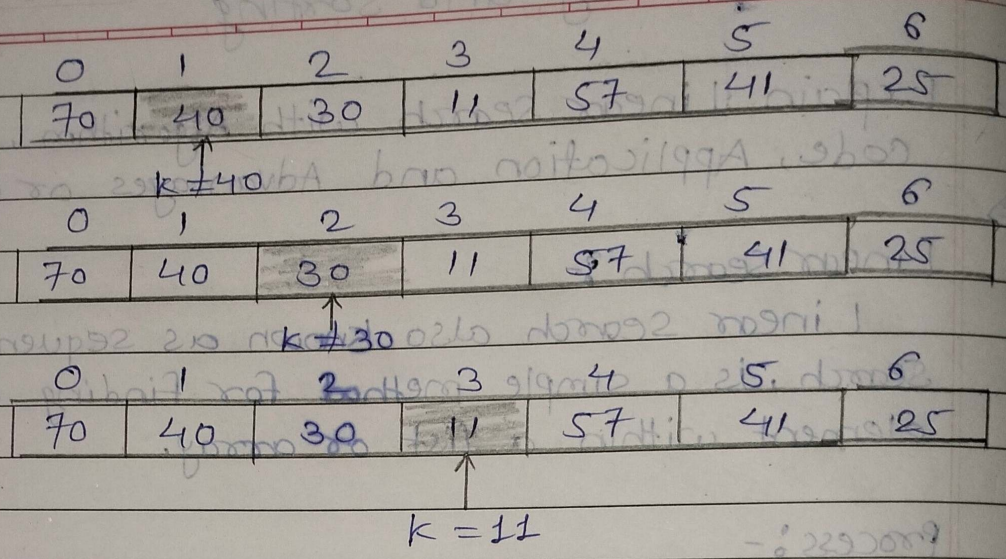
Now, start from the first element and compare k with each element of the array.

0	1	2	3	4	5	6
70	40	30	11	57	41	25

↑

$k \neq 70$

The value of k i.e. 11, is not matched with the first element of the array. So, move to the next element. And follow the same process until the respective element is found.



Now, the element to be searched is found. So algorithm will return the index of the element matched.

Time complexity :-

a) Best case complexity - In linear search, best case occurs when the element we are finding is at the first position of the array.
- The best-case time complexity of linear search is O(1).

b) Average case complexity :-
The average case time complexity of linear search is O(n)

c) Worst case complexity :-
In Linear search, the worst case occurs when the element we are looking is present at the end of the array.
- The worst-case time complexity of linear search is O(n).

Algorithm:-

Linear_search (a, n, val) // 'a' is the given array.

step 1:- set pos = -1

step 2:- set i = 1

step 3: repeat step 4 while i<=n

Step 4: if a[i] == val

set pos = i

print pos

go to step 6

[end of if]

set i = i+1

[end of loop]

Step 5: if pos = -1

print "value is not present in the array"

[end of if]

step 6:

exit

Pseudo code:-

~~for each item in the list~~

~~if matchitem = value~~

~~return the item's location~~

~~end if~~

~~end for~~

~~end procedure.~~

```
int linearsearch (int a[], int n, int val)
{ // Going through array sequencially
  for(int i=0; i<n; i++)
  {
    if(a[i]==val)
      return i+1;
  }
  return -1
}
```

Application:-

1] Phonebook search:-
Linear search can be used to search through a phonebook to find a person's name, given their phone number.

2] Spell checkers:-
The algorithm compares each word in the document to a dictionary of correctly spelled words until a match is found.

3] Finding minimum and maximum values:
Linear search can be used to find the minimum and maximum values in an array or list.

4] Searching through unsorted data:-
Linear search is useful for searching through unsorted data.

Advantages:-
1] Simplicity
2] Works with unsorted data
3] Low memory usage
4] Easy to debug

Disadvantages:-
1] Inefficient for large datasets
2] Limited applicability
3] No early termination
4] Inefficient for sorted data.

Que- Explain Binary search with Algorithm, Pseudo code, Application and Advantages or disadvantages.

⇒

- Binary Search:-

A search algorithm that efficiently locates a target element in a sorted collection by repeatedly dividing the search interval in half, eliminating half of the remaining elements in each step until the target is found or the entire collection is examined.

- Working of Binary search:-

There are two method to implement the binary search algorithm:-

1) Iterative method
2) Recursive method

- The recursive method of binary search follows the divide and conquer approach.
- Let the element of array are-

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69

Let the element to search is k = 56

We have to use the below formula to calculate the mid of the array-

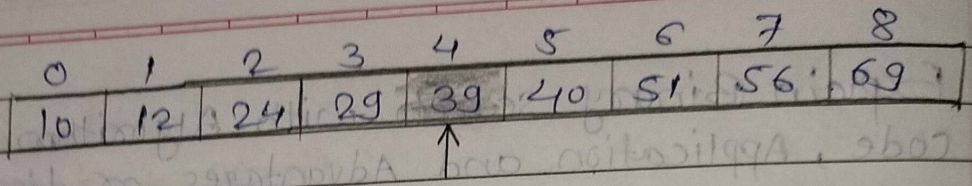
mid = (beg + end)/2

So, in the given array-

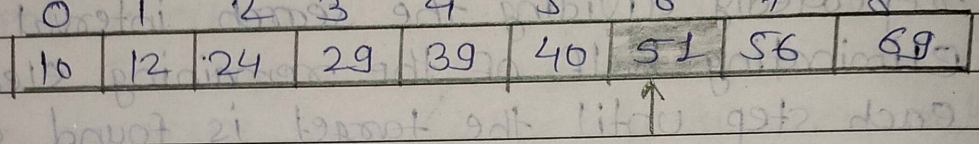
beg = 0

end = 8

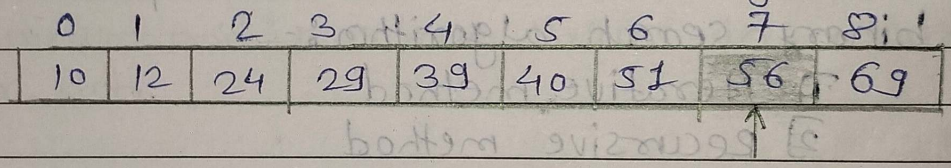
mid = (0+8)/2 = 4 so, 4 is mid of array.



A[mid] = 39
A[mid] < k (or 39 < 56)
so, beg = mid + 1 = 5, end 8
Now, mid = (beg + end)/2 = 13/2 = 6



A[mid] = 51
A[mid] < k (or 51 < 56)
so, beg = mid + 1 = 7, end = 8
Now, mid = (beg + end)/2 = 15/2 = 7



A[mid] = 56
A[mid] = k (or 56 = 56)
so, location = mid.
Element found at 7th location of array

- Now, the element to search is found. So, algorithm will return the index of the element matched.

- Time complexity :-

a] Best case complexity -
- best case occurs when the element to search is found in first comparison.
- The best-case time complexity of Binary Search is O(1).

b] Average case complexity :-
The average case time complexity of Binary search is O(logn).

c] worst case complexity :-
- The worst case occurs, when we have to keep reducing the search space till it has only one element.
- The worst-case time complexity of Binary search is O(logn).

- Algorithm :-

Binary_search(a, lower_bound, upper_bound, val)
step1 :- set beg = lower_bound, end = upper_bound, pos = -1
step2 :- repeat steps 3 and 4 while beg <= end
step3 :- set mid = (beg + end)/2
step4 :- if a[mid] = val
set pos = mid
Print pos
go to step 6
else if a[mid] > val
set end = mid - 1
else
set beg = mid + 1
[end of if]
[end of loop]
step5 :- if pos = -1
print "value is not present in the array"
[end of if]
step 6 : Exit.

Pseudocode:-

```
if (size == 0)
    found = false;
else {
    middle = index of approximate midpoint of array
    if (target == a[middle])
        // target has been found!
else if (target < a[middle])
    search for target in area before midpoint;
else
    search for target in area after midpoint;
}
```

Applications of Binary search:-

1] Binary search can be used as a building block for more complex algorithms used in machine learning, such as algorithms for training neural networks or finding the optimal hyperparameters for a model.
2] It can be used for searching in computer graphics such as algorithms for ray tracing or texture mapping.
3] It can be used for searching a database.

Advantages:-

1] It is better than a linear search algorithm since its run time complexity is O(logN).
2] At each iteration, the binary search algorithm eliminates half of the list and significantly

reduces the search space.

3] The binary search algorithm works even when the array is rotated by some position and finds the target element.

Disadvantages :-

1] The recursive method uses stack space.

2] Binary search is error-proone.

3] Off-by-one errors : While determining the boundary of the next interval, there might be overlapping errors.

Que- Explain Bubble sort with Pseudocode, Example, Application and advantage or disadvantage.

⇒

Bubble sort :-

Bubble sort is a simple sorting algorithm that repeatedly steps through a list, compares adjacent elements, and swaps them if they are in the wrong order.

- The pass through the list is repeated until the list is sorted.
- It is named for the way smaller elements "bubble" to the top of the list during each iteration.
- Bubble sort has a time complexity of $O(n^2)$, making it less efficient than many other sorting algorithms for large datasets.

Pseudo Code :-

```
Void bubbleSort (int [] a)
{
    int stage, inner, temp;
    for(stage = a.length - 1; stage > 0; stage --)
        {                                   // counting down
        for (inner = 0; inner < stage; inner ++) // bubbling up
            {
            if (a[inner] > a[inner + 1]) { // if out of order
            temp = a[inner]; // then swap
            a[inner] = a[inner + 1];
            a [inner + 1] = temp;
        }
    }
}
```

Finding near by element is called Buddy system

D D M M Y Y Y Y

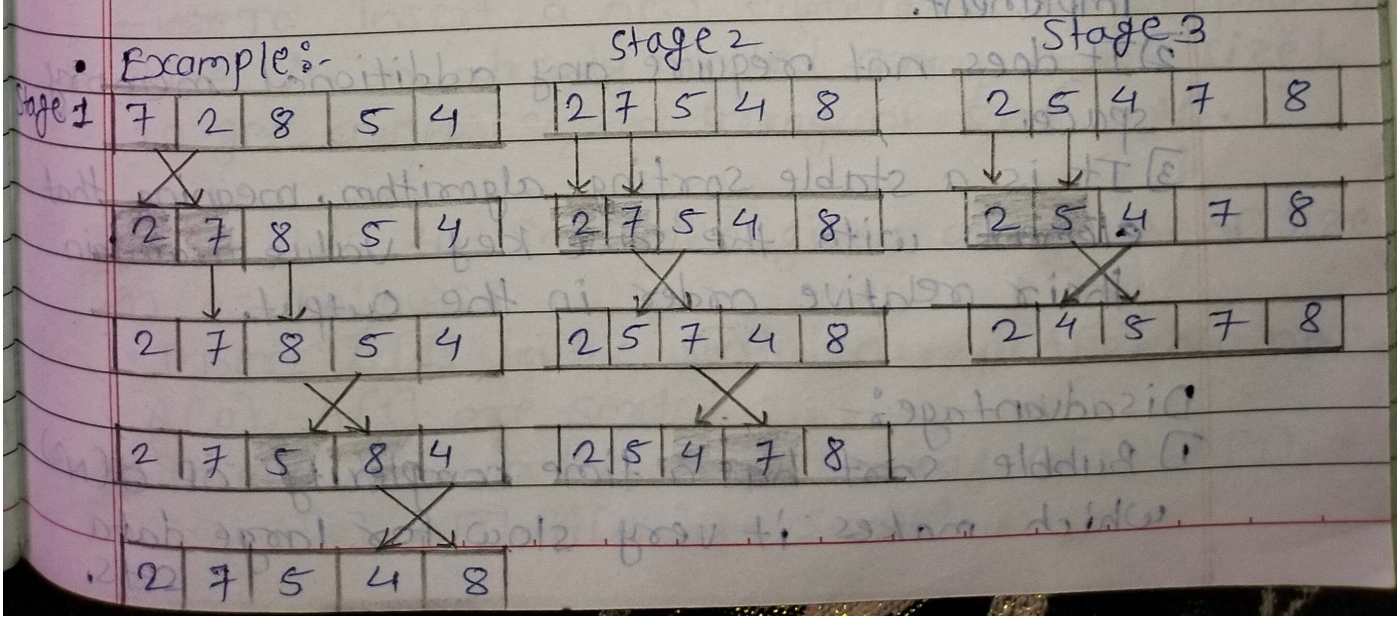
- Stepwise representation of Bubble sort :-

Stage 1 :- Compare each element (except the last one) with its neighbor to the right.
- If they are out of order, swap them
- This puts the largest element at the very end.
- The last element is now in the correct and final space.

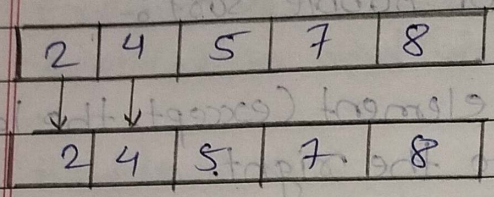
Stage 2 :- Compare each element (except the last two) with it's neighbor to the right
- If they are out of order, swap them
- This puts the second largest element next to last.
- The last two elements are now in their correct and final places.

Stage 3 :- Compare each element (except the last three) with its neighbor to the right.
- Continue as above until you have no sorted element on the left.

- Example :-



Stage 4



(done)

Application:-

1) Bubble sort is a sorting algorithm that is used to sort the elements in an ascending order.
2) It uses less storage space
3) Bubble sort can be beneficial to sort the unsorted elements in a specific order.
4) It can be used to sort the students on basis of their height in a line.
5) To create a stack, pile up the elements on basis of their weight.

Advantages:-

1) Bubble sort is easy to understand and implement.
2) It does not require any additional memory space.
3) It is a stable sorting algorithm, meaning that elements with the same key value maintain their relative order in the output.

Disadvantage:-

1) Bubble sort has a time complexity of $O(N^2)$ which makes it very slow for large data sets.

2] Bubble Sort is a comparison-based sorting algorithm, which means that it requires a comparison operator to determine the relative order of elements in the input data set. It can limit the efficiency of the algorithm in certain cases.

Que- Explain insertion sort with Pseudocode, Example, Application and advantages or disadvantages.

⇒

Insertion Sort :-

- Insertion Sort works similar to the sorting of playing cards in hands.
- The idea behind the insertion sort is that First take one element, iterate it through the sorted array.
- Although it is simple to use, it is not appropriate for large data sets as the time complexity of insertion sort in the average case and worst case is $O(n^2)$, where n is the number of items.
- To insert a new element x in its proper place in an already sorted array A of size k to get a new sorted array of size k+1.

- Use this to Sort the given array A of size n as Follows :-
- Insert A[1] in the sorted array A[0]. so now A[0], A[1] are sorted.
- Insert A[2] in the sorted array A[0], A[1]. So now A[0], A[1], A[2] are sorted.

Space complexity - O(1)

Time complexity - bast case O(N)
worst case O(N²)
average case O(N²)

- Insert A[3] in the sorted array A[0], A[1], A[2]. So now A[0], A[1], A[2], A[3] are sorted.
- Insert A[i] in the sorted array A[0], A[1], ---, A[i-1]. So now A[0], A[1], ---, A[i] are sorted.
- continue until i = n - 1

Example:-

A	7	5	13	11	22	20

Insert 5 in 7

5	7	13	11	22	20

Insert 22 in 5, 7, 11, 13

5	7	11	13	22	20

Insert 13 in 5, 7

5	7	13	11	22	20

Insert 20 in 5, 7, 11, 13, 22

5	7	11	13	20	22

Insert 11 in 5, 7, 13

5	7	11	13	22	20

Pseudo Code:-

```
Void InsertionSort(int A[], int size)
{
  int i, j, item;
  for(i=1; i<size; i++)
  { /* Insert the element in A[i] */
    item = A[i];
    for(j=i-1; j>=0; j--)
      if(item < A[j])
```

```
    /* Push elements down */
    A[j+1] = A[j];
    A[j] = item; /* can do this once finally also */
  }
  else break; /* inserted, exit loop */
 }
}
```

steps of insertion sort :-

1] start with the second element :-
- Assume the first element is sorted, and begin with the second element as the first element's potential partner.

2] compare and insert :-
- Compare the current element with the elements in the sorted portion.
- Insert the current element into its correct position in the sorted part, shifting larger elements to the right.

3] Repeat :- Continue this process for each unsorted element until the entire array is sorted.

Application :-

1] Card Games
2] small Datasets
3] online Form submission
4] Dynamic Data
5] live chat application
6] sensor Data processing
7] online auctions
8] network routing tables.
9] Real time event processing.

Advantages :-

1) Simple implementation
2) Efficient for small data sets
3) Adaptive, i.e. it it appropriate for data sets that are already substantially sorted.
4) stable
5) In-place sorting.

Disadvantages :-

1) Inefficient for large Datasets
2) sensitive to initial order
3) Not suitable for Linked Lists.

Que- Explain merge sort with Pseudo code, example, Applicatian and Advantages or disadvantages.

⇒

Merge sort :-

- merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the element.
- It is one of the most popular and efficient sorting algorithm.
- It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves.

Time complexity :-

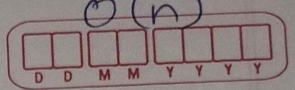
Bast case - $O(n*\log n)$

Averge case - $O(n*\log n)$

~~worst case - $O(n*\log n)$~~

Space complexity -

$O(n)$



steps :-

1) Divide the array into two halves.
   - sort the two sub-arrays
   - merge the two sorted sub-arrays into a single sorted array.

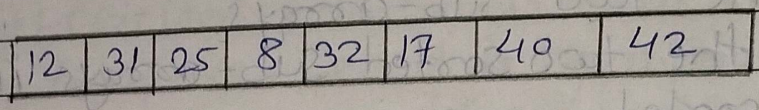
2) (Sorting the sub-arrays) is done recursively (divide in two, sort, merge) until the array has a single element (base condition of recursion).

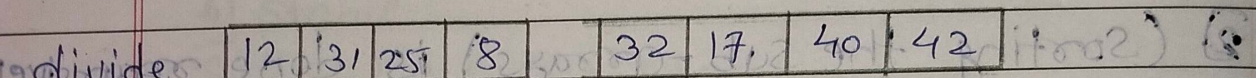
Pseudocode :-

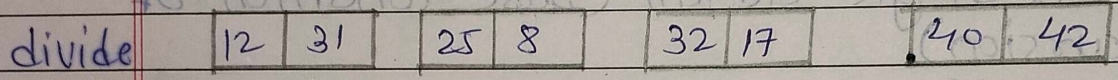
```
void mergesort [int A[], int n]
{
  int i, j, B[max];
  if (n<=1) return;
  i=n/2;
  mergesort (A, i);
  mergesort (A+i, n-i);
  merge (A, A+i, B, i, n-i);
  For (j=0; j<n; j++)
   A[j] = B[j];
  Free (B);
}
```

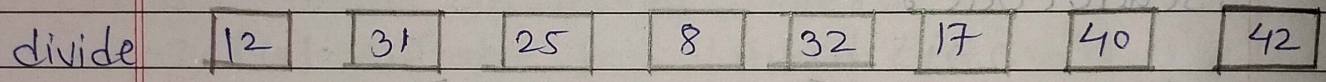
Example:-

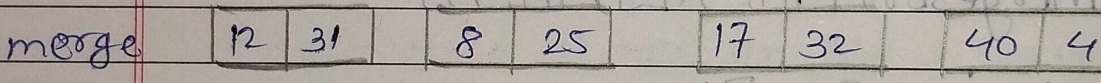
elements of array are:-

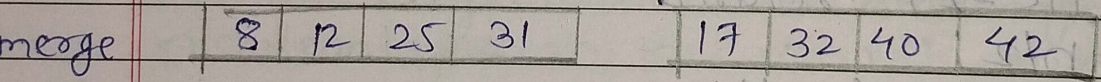


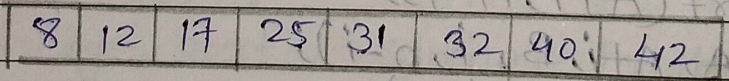












array is completely sorted.

Application:-

1. Sorting large datasets:- merge sort is particularly well-suited for sorting large datasets due to its guaranteed worst-case time complexity of O(n log n).

2] External sorting :-
- merge sort is commonly used in external sorting, where the data to be sorted is too large to fit into memory.

3] custom sorting :-
- Merge sort can be adapted to handle different input distributions, such as partially sorted, nearly sorted, or completely unsorted data.

4] Inversion count Problem.

Advantages :-

1) Stability :- merge sort is a stable sorting algorithm, which means it maintains the relative order of equal elements in the input array.
2) Guaranteed worst-case performance :- merge sort has a worst-case time complexity of O(N log N), which means it performs well even on large datasets.
3) parallelizable :- Merge sort is a naturally Parallelizable algorithm, which means it can be easily parallelized to take advantages of multiple processors or threads.

Disadvantages :-

1) Space complexity :- Merge sort requires additional memory to store the merged sub-arrays during the sorting process.
2) Not-in-place :- Merge sort is not an in-place

Sorting algorithm, which means, it requires additional memory to store the sorted data.

3] Not always optimal for small datasets :-
For small datasets, merge sort has a higher time complexity than some other sorting algorithms, such as insertion sort.

Que - Explain selection sort with Pseudo code, example, Application and Advantages or Disadvantages.

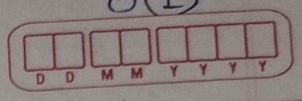
→

selection sort :-
- In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array.
- In selection sort, the First smallest element is selected from the unsorted array and placed at the first position.
- After that second smallest element is selected and placed in the second position.
- The process continues until the array is entirely sorted.
- selection sort is generally used when -
  - A small array is to be sorted
  - swapping cost doesn't matter.
  - It is compulsory to check all elements.

Steps :-
1] Find the minimum value in the list

Time complexity:-
Best case - $O(n^2)$
Average case - $O(n^2)$
worst case - $O(n^2)$

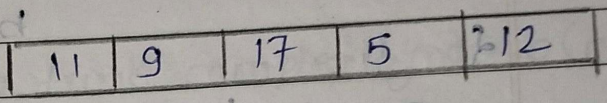
space complexity - $O(1)$



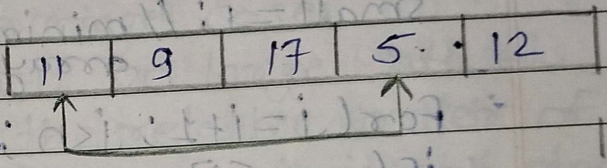
2] swap it with the value in the first position
3] sort the remainder of the list (excluding the first value).

Example:-

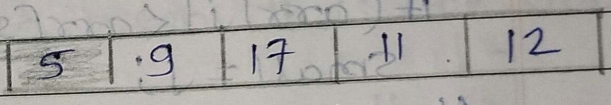
Initial State



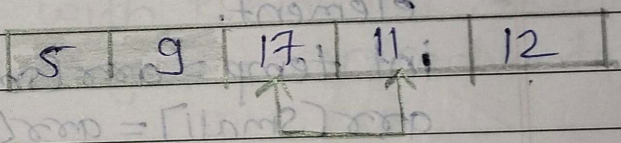
Begin First Pass
min value for Pass 1 is 5:



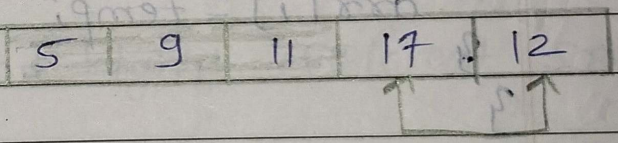
Begin Second Pass
min value for Pass 2 is: 9



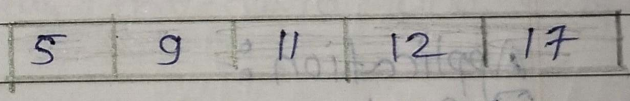
Begin third Pass
min value for Pass 3 is: 11



Begin fourth Pass
min value for Pass 4 is: 12



Final state:-



-collection is now selection sorted in ascending order.

Pseudocode :-

```
void selection (int arr[], int n)
{
   int i, j, small;
   for(i=0; i<n-1; i++) //one by one move
                        boundary of unsorted subarray.
   {
     small = i; // minimum element in unsorted
                   array.
     for(j=i+1; j<n ; j++)
     if(arr[j] < arr[small])
       small = j;
   // swap the minimum element with the first
     element.
     int temp = arr[small];
     arr[small] = arr[i];
     arr[i] = temp;
   }
}
```

Application :-

1) Product price sorting :- use selection sort to arrange products based on their prices.
2) user Interface Display :-
   - In graphical user interfaces (GUIs), selection sort can be applied to sort list or tables of data.
3) classroom Ranking :- Implement selection sort to rank students in a classroom based on their exam scores.
4) Programming utility.

Advantages:-
1) simple and easy to understand
2) works well with small datasets

Disadvantages:-
1) selection sort has a time complexity of $O(n^2)$ in the worst and average case.
2) Does not work well on large datasets
3) Does not preserve the relative order of items with equal keys which means it is not stable.

Que- Explain Quick sort with Pseudocode, example, Application and Advantages or disadvantages.

⇒

Quick Sort:-
- It is a faster and highly efficient sorting algorithm.
- This algorithm follows the divide and conquer approach.
- Quick sort Picks an element as Pivot, and then it Partitions the given array around the Picked Pivot element.
- In quick Sort, a large array is divided into two arrays in which one holds values that are smaller than the specified value (Pivot) and another array holds the values that are greater than the Pivot.

Time complexity :-
Best case - $O(n * \log n)$
Average case - $O(n * \log n)$
worst case - $O(n^2)$

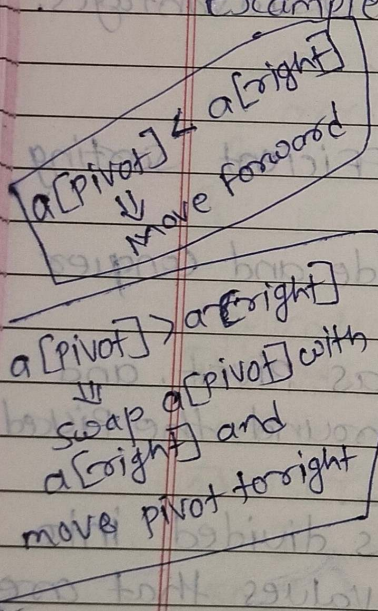
Space complexity - $O(n * \log n)$

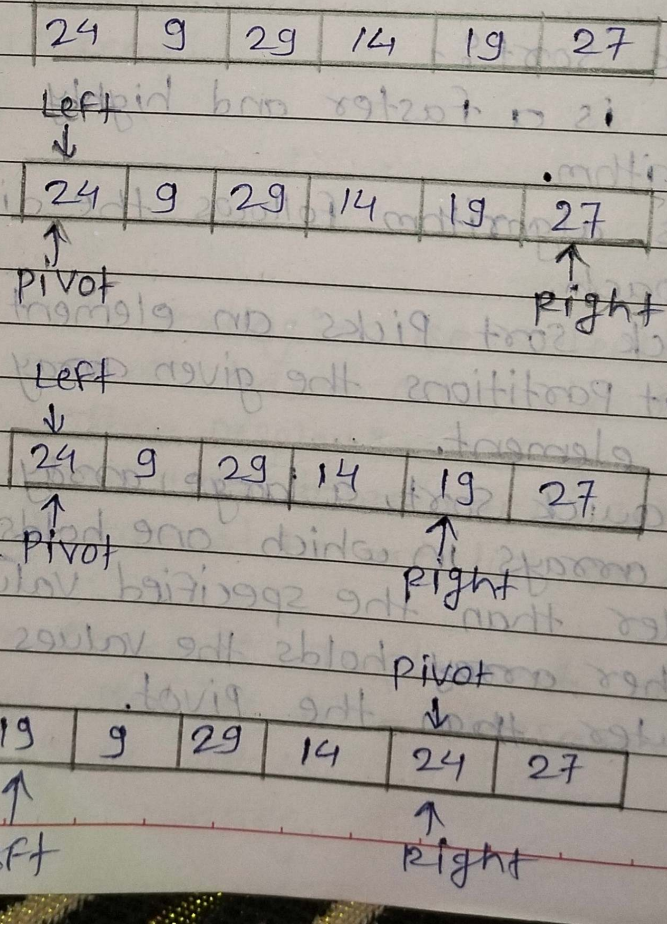
D D M M Y Y Y Y

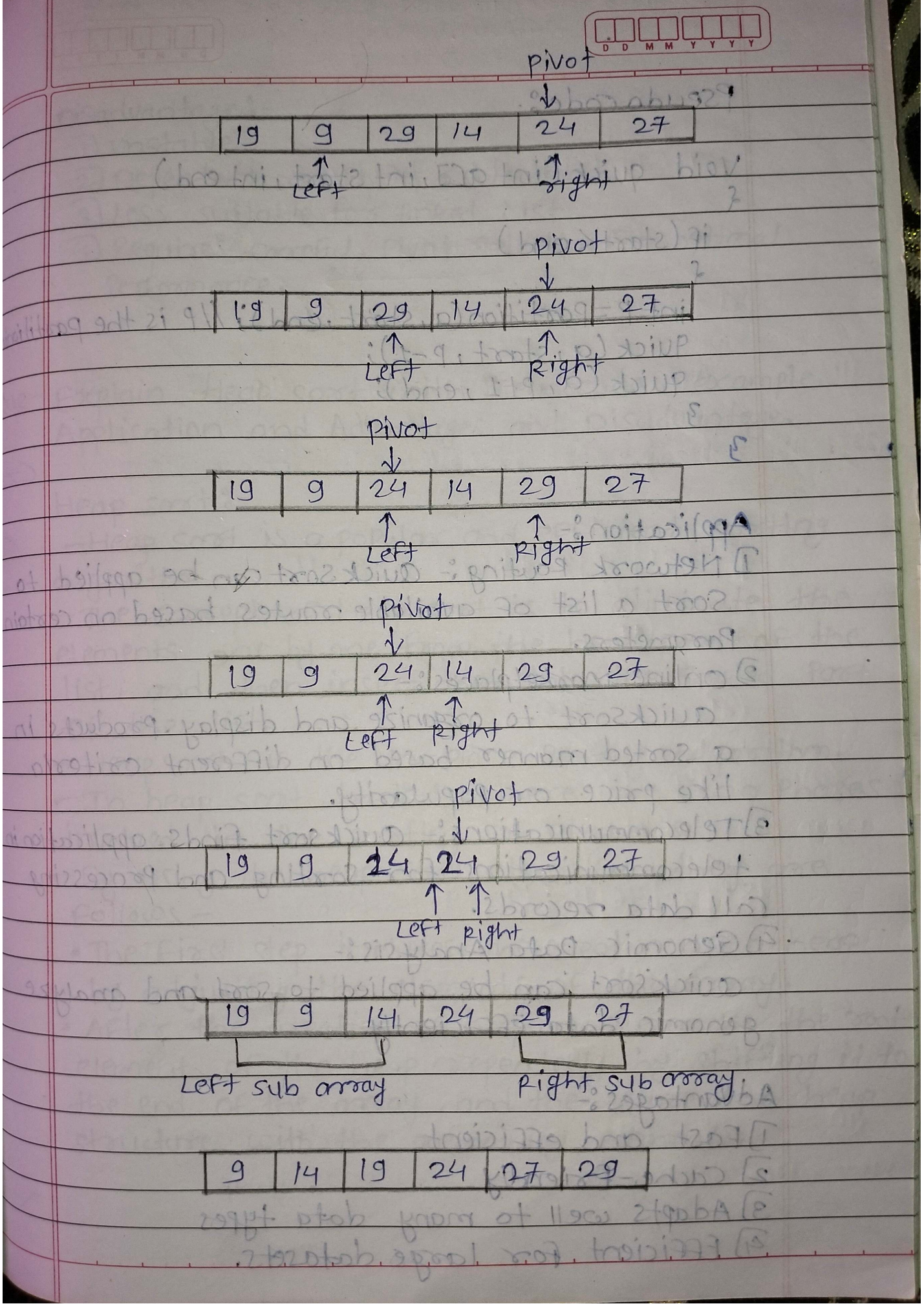
- Choosing the Pivot :-
  1] pivot can be random i.e. select the random pivot from the given array.
  2] Pivot can either be the rightmost element of the leftmost element of the given array.
  3] select median as the pivot element.

- Partitioning Strategy :-
  - we want to partition array A[left ... right]
  - first, get the pivot element out of the way by swapping it with the last element (swap pivot and A[right]).
  - Let i start at the first element and j start at the next to last element.

Example :-






pivot
19 9 29 14 24 27
Left
Right
pivot
19 9 29 14 24 27
Left
Right
pivot
19 9 24 14 29 27
Left
Right
pivot
19 9 24 14 29 27
Left
Right
pivot
19 9 24 24 29 27
Left
right
19 9 14 24 29 27
Left sub array
Right sub array
9 14 19 24 27 29

Pseudo code:-

```
Void quick(int a[], int start, int end)
{
  if(start<end)
  {
   int p=Partition(a, start, end); //p is the partitioning
   quick(a, start, p-1);
   quick(a, p+1, end);
  }
}
```

Application:-

1) Network Routing:- Quicksort can be applied to sort a list of available routes based on certain Parameters.
2) online Marketplaces:-
   quicksort to organize and display products in a sorted manner based on different criteria like price or popularity.
3) Telecommunications:- quicksort finds application in telecommunications for sorting and processing call data records.
4) Genomic Data Analysis:-
   quicksort can be applied to sort and analyze genomic data efficiently.

Advantages:-

1) Fast and efficient
2) Cache-friendly
3) Adapts well to many data types
4) Efficient for large datasets.

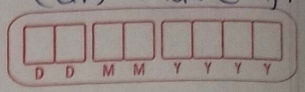
Disadvantage:-
1] Unstable
2] Dependent on Pivot selection
3] Less suitable for linked List
4] Requires careful Pivot selection for optimal Performance.

Que- Explain Heap sort with Pseudocode, example, Application and Advantages and disadvantage.

⇒

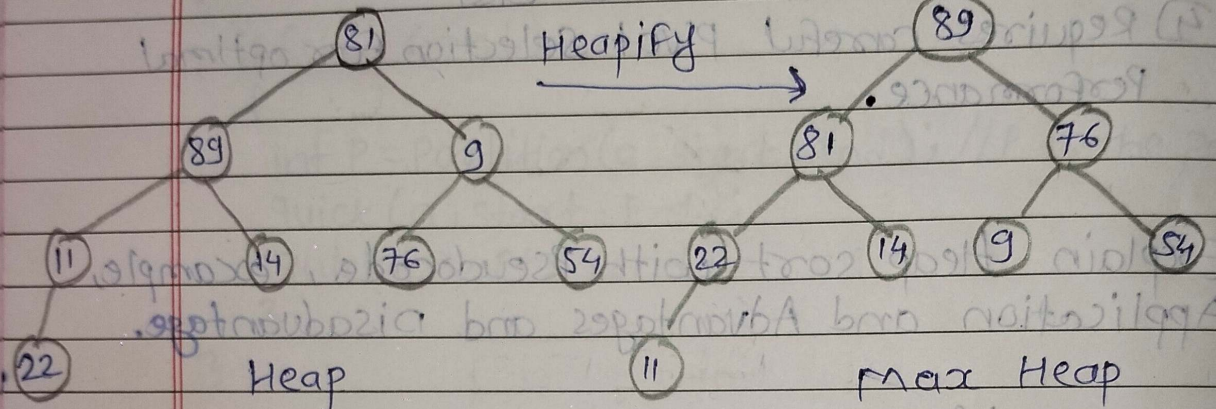
Heap sort:-
- Heap sort is a popular and efficient sorting algorithm.
- The concept of heap sort is to eliminate the elements one by one from the heap part of the list, and then insert them into the sorted part of the list.
- Heap sort is the in-place sorting algorithm.
- In heap sort, basically, there are two phases involved in the sorting of elements.
- By using the heap sort algorithm, they are follows -
  - The First step includes the creation of a heap by adjusting the elements of the array.
  - After the creation of heap, now remove the root element of the heap repeatedly by shifting it to the end of the array, and then store the heap structure with the remaining elements.

Heap :- A heap is a complete binary tree, and the binary tree is a tree in which the node can have the most two children.



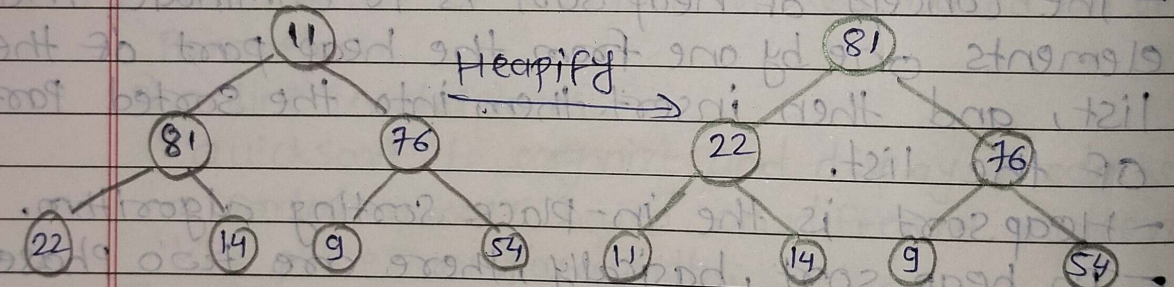
Example :-

81	89	9	11	14	76	54	22



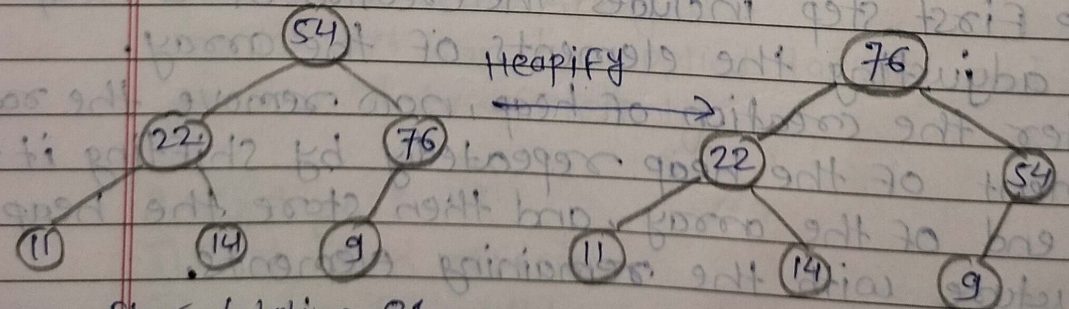
Heap Max Heap

89	81	76	22	14	9	54	11



Heap after deleting 89 Max Heap

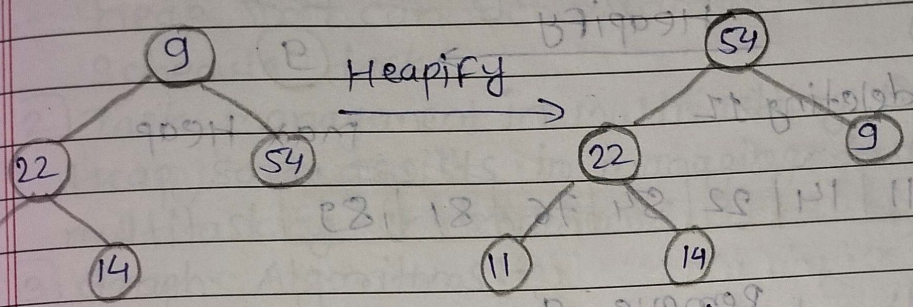
81	22	76	11	14	9	54	89



Heap after deleting 81 Max Heap

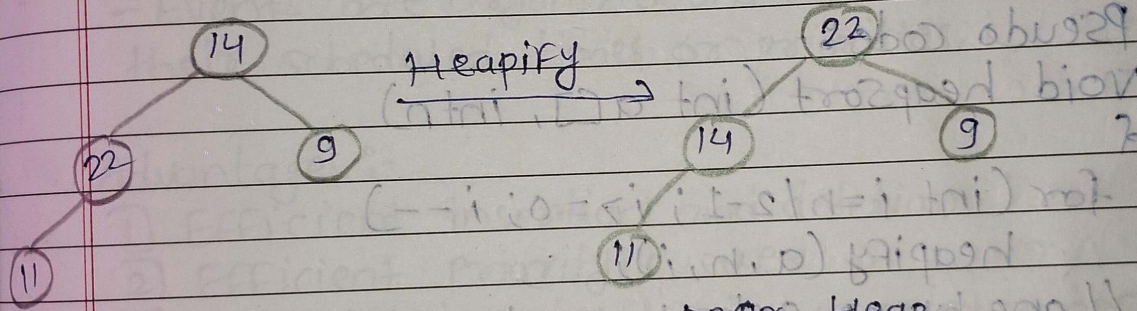
- A complete binary tree is a binary tree in which all the levels except the last level i.e. leaf node should be completely filled, and all the nodes should be left-justified.

76	22	54	11	14	9	81	89



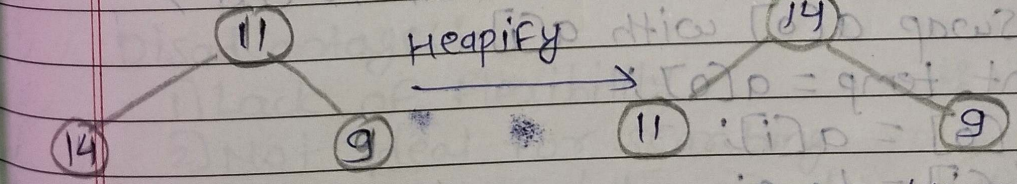
Heap after deleting 76 — max Heap

54	22	9	11	14	76	81	89



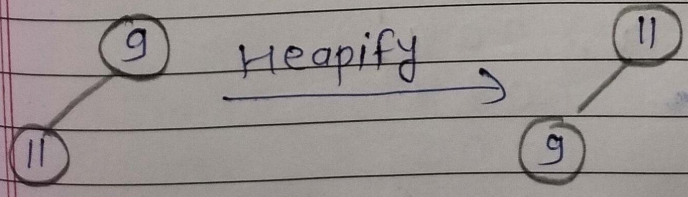
max Heap

22	14	9	11	54	76	81	89



Heap after deleting 22 — max Heap

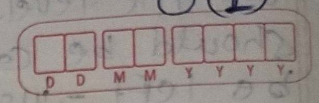
14	11	9	22	54	76	81	89



Heap after deleting 14 — max Heap

Time complexity:-
Base case - O(n log n)
Average case - O(n log n)
worst case - O(n log n)

Space complexity - O(1)



11	9	14	22	54	76	81	89

(9) ~~Heapify~~ → (9)

Heap after deleting 11 — max Heap

9	11	14	22	54	76	81	89

(9) Remove 9 → Empty

9	11	14	22	54	76	81	89

Pseudo code:-

```
void heapSort (int a[], int n)
{
 for (int i=n/2-1; i>=0; i--)
    heapify (a,n,i);
 // one by one extract an element from heap
 for( int i=n-1; i>=0; i--)
 {
   /* Move current root element to end */
   // Swap a[0] with a[i]
    int temp = a[0];
    a[0] = a[i];
    a[i] = temp;
    heapify (a,i,0);
  }
}
```

Application:-

1) Job scheduling:-
   Heap sort can be used to prioritize and schedule tasks.
2) Task management in multitasking Environments:-
   Heap sort assists in managing tasks in multitasking environments.
3) Graph Algorithms:-
   Heap sort is useful in graph algorithms for tasks like finding the shortest path.
4) Event Management in simulation software:-
   - Events are organized and executed based on their scheduled times or priorities.

Advantages:-

1) Efficient insertion and deletion
2) Efficient priority queue
3) Guaranteed access to the maximum or minimum element
4) space efficiency.

Disadvantages:-

1) Lack of flexibility
2) Not ideal for searching
3) Not a stable data structure.

7/12/2023

que- Explain Radix sort with Pseudocode, example, Application and Advantages or Disadvantages.

⇒

Radix sort:-

- The Radix sort, there is digit by digit sorting is performed that is started from the least significant digit to the most significant digit.
- The process of radix sort works similar to the sorting of students names, according to the alphabetical order.
- It is non-comparison algorithm

Time complexity :-

Best case :- $\Omega(n+k)$
Average case :- $O(nk)$
worst case :- $O(nk)$

Space complexity :-
$O(n+k)$

Example :-

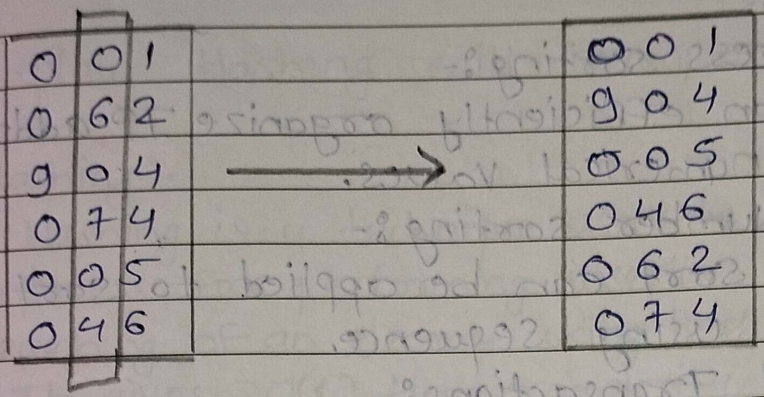
904	46	5	74	62	1

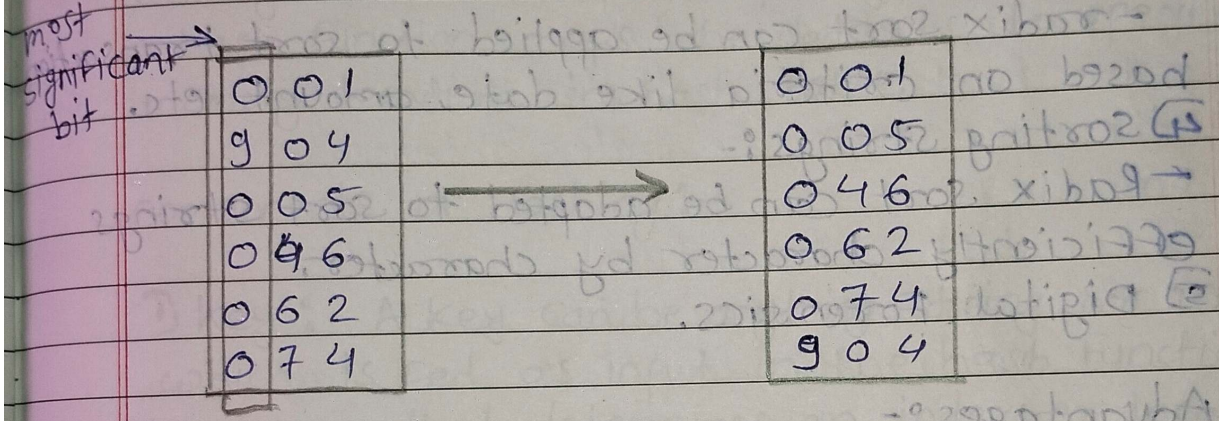
Least significant digit →

9	4
0	6
0	5
7	4
6	2
0	1

→

001
062
904
074
005
046





1	5	46	62	74	904

(done)

Pseudocode :-

```
Void radixsort (int a[], int n)
{
  // get maximum element from array
   int max = getmax(a,n);

// Apply counting sort to sort elements based on
   place.
for(int place = 1; max/place > 0; place *= 10)
  countingsort (a,n, place);
}
```

Application:-
1) IP Address sorting:-
It can efficiently organize IP addresses by their numerical values.
2) Phone Number sorting:-
- radix sort can be applied to sort Phone numbers using sequence.
3) Financial Transactions:-
- radix sort can be applied to sort transaction based on criteria like date, amount etc.
4) Sorting strings:-
- Radix sort can be adapted to sort strings efficiently character by character.
5) Digital Forensics.

Advantages:-
1) Linear Time complexity
2) Stable sorting
3) No Need for comparisons
4) suitable for Fixed-Length keys
5) memory Efficiency.

Disadvantages:-
1) space complexity
2) Not Always In-place
3) Not Always the Fastest.
4) Difficulty with Floating-Point Numbers.
5) Lack of universality.
6) Limited Applicability to Variable-Length keys.

Que- Explain Hashing with example and application.

⇒

Hashing :-

- Hashing is a technique used to compute memory address for performing insertion, deletion and searching of an element using hash function.
- It gives O(1) time complexity in searching
- Hash function maps the datasets to specific memory locations in hash table.

components of Hashing :-

1) key :- A key can be anything string or integer which is fed as input in the hash function the technique that determines an index or location for storage of an item in a data structure.

2) Hash Function :- The hash function receives the input key and returns the index of an element in an array called a hash table. The index is known as the hash index.

3) Hash Table :- Hash table is a data structure that maps keys to values using a special function called a hash function.

- Hash stores the data in an associative manner in an array where each data value has its own unique index.

- Significance/Advantages:-
  1) Fastest Data Retrival
  2) Space efficiency
  3) Efficient insertion and deletion

D D M M Y Y Y Y

Types of Hash functions:-
1) Division method
2) Mid square method
3) Folding method

1) Division method:-
- This is the most simple and easiest method to generate a hash value.
- The hash function divides the value k by m and then uses the remainder obtained.

Formula:-

$$h(k) = k \bmod m$$

Here, k is the key value
m is the size of the hash table

- The hash function is dependent upon the remainder of a division.

2) mid-square method:-
- The mid-square method is a very good hashing method.
- It involves two steps to compute the hash value.
  1) Square the value of the key k i.e. k^2
  2) Extract the middle r digits as the hash value.

Formula:-

$$h(k) = h(k \times k)$$

Here,
k is the key value.

- The value of r can be decided based on the size of the table.

Time complexity: O(n)
Space complexity: O(1)

DD MM YYYY

3] Folding Method:-
This method involves two steps:-
1] Divide the key-value k into a number of parts i.e. $k_1, k_2, k_3, ---, k_n$, where each part has the same number of digits except for the last part that can have lesser digits than the other parts.
2] Add the individual parts. The hash value is obtained by ignoring the last carry if any.
Formula:-

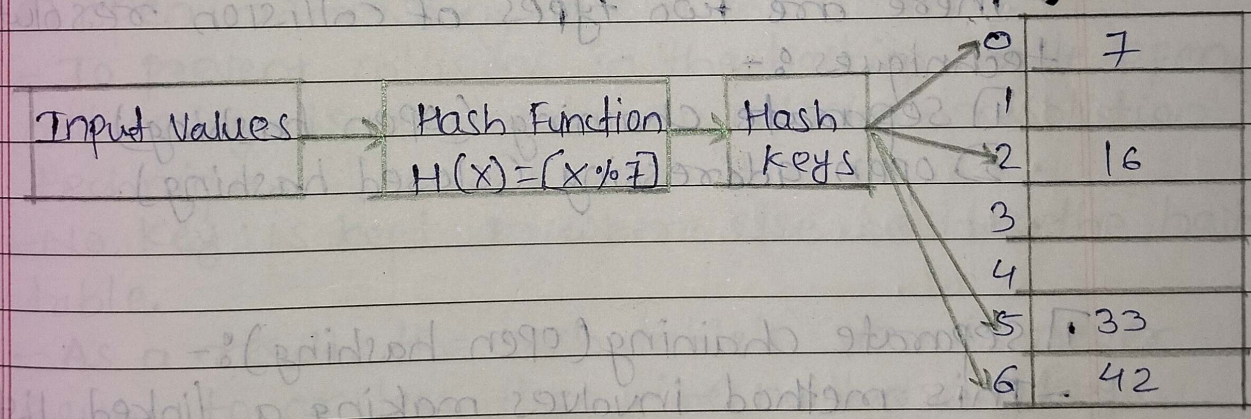
$k = k_1, k_2, k_3, k_4, ---, k_n$

$s = k_1 + k_2 + k_3 + k_4 + --- + k_n$

$h(k) = s$

Here,
s is obtained by adding the parts of the key k.

Example:-



Application:-
1] Hash is used for cache mapping for fast access to the data.
2] Hash can be used for Password verification
3] Hash is used in cryptography as a message digest.
4] Rabin-karp algorithm for pattern matching in string.
5] calculating the number of different substrings of a string

Que Explain collision resolving solution using Hashing.

⇒

Collision Resolution Strategies:-
- If two keys map on the same hash table index then we have a collision
- collisions may still happen, so we need a collision resolution strategy.
- The hash value is used to create an index for the key in the hash table.
- The hash function may return the same hash value for two or more keys.
- When two or more keys have the same hash value, a collision happens.

- There are two types of collision resolution techniques :-
  1) separate chaining (open hashing)
  2) open addressing (closed hashing).

1) Separate chaining (open hashing):-
- This method involves making a linked list out of the slot where the collision happened, then adding the new key to the list.
- Separate chaining is the term used to describe how this connected list of slots resembles a chain.
- It is more frequently utilized when we are unsure of the number of keys to add or remove.

Time complexity.
- Worst-case complexity for searching is O(n).
- worst-case complexity for deletion is: O(n).

Advantages of separate chaining:-
1) easy to implement
2) The hash table never fills full, so we can add more elements to the chain.
3) It is less sensitive to the function of the hashing.

Disadvantages of separate chaining:-
1) cache performance of chaining is not good
2) memory wastage is too much in this method
3) It requires more space for element links.

2) open addressing:-
- To prevent collisions in the hashing table open addressing is employed as a collision-resolution technique.
- No key is kept anywhere else besides the hash table.
- As a result, the hash table's size is never equal to or less than the number of keys.
- Additionally known as closed hashing.

- Probing:- If the table position given by the hashed key is already occupied, increase the position by some amount, until an empty position is found.

- The following techniques are used in open addressing:-
1) Linear Probing
2) Quadratic Probing
3) Double hashing.

1) Linear Probing:-
- This involves doing a linear Probe for the following slot when a collision occurs and continuing to do so until an empty slot is discovered.
- The worst time to search for an element in linear probing is: O.
- The cache performs best with linear Probing, but clustering is a concern.
- This method's key benefit is that it is simple to calculate.

Disadvantages of linear probing:-
- The main problem is clustering
- It takes too much time to find an empty slot.

2) Quadratic Probing:-
- When a collision happens in this, we Probe for the i2-nd slot in the ith iteration, continuing to do so until an empty slot is discovered.
- In comparison to linear probing, quadratic probing has a worse cache performance.
- Additionally, clustering is less of a concern with quadratic probing.

3) Double hashing :-

- In this, you employ a different hashing algorithm, and in the ith iteration, you look for (i * hash 2(x)).
- The determination of two hash functions requires more time.
- Although there is no clustering issue, the performance of the cache is relatively poor when using double probing.

Time Complexity :-

- Insertion : O(n)
- Search : O(n)
- Deletion : O(n)

Auxiliary space : O(size of the hash table).

1) Linear Probing :- Example :- Hash function as "key mod 7" and sequence of key as 50, 700, 76, 85, 92, 73, 101

0		0		0	700		0	700		0	700		0	700	
1		1	50	1	50		1	50		1	50		1	50	
2		2		2			2	85		2	85		2	85	
3		3		3			3			3			3	92	
4		4		4			4			4			4		
5		5		5			5			5			5		
6		6		6	76		6	76		6	76		6	76	

Initial Empty table

Insert 50

Insert 700 and 76

Insert 85 : collision occurs insert 85 at next free slot at index 1

Insert 92, collision occurs as 50 is there Insert at next free slot

0	700
1	50
2	85
3	92
4	73
5	101
6	76

Insert 73 and 101

2) Quadratic Probing :- Example :-

A simple hash function as "key mod 7" and sequence of keys as 50, 700, 76, 85, 92, 73, 101

0		0		0	700	0	700	0	700	0	700	0	700	0	700
1		1	50	1	50	1	50	1	50	1	50	1	50	1	50
2		2		2		2	85	2	85	2	85	2	85	2	85
3		3		3		3		3		3		3		3	
4		4		4		4		4		4		4		4	
5		5		5		5		5		5		5	92	5	92
6		6		6	76	6	76	6	76	6	76	6	76	6	76

Initial empty table

Insert 50

Insert 700 and 76

Insert 85, collision occurs Insert at 1+1*1 Position

Insert 92, collision occurs at 1. collision occurs at 1+1*1 Position Insert at 1+2*2 position

0	700
1	50
2	85
3	73
4	101
5	92
6	76

Insert 73 and 101

3) Double Hashing :- Example :-

Let, Hash1 (key) = key % 13

Hash 2(key) = 7 - (key % 7)

Hash 1 (19) = 19 % 13 = 6

Hash1 (27) = 27 % 13 = 1

Hash1 (36) = 36 % 13 = 10

Hash 1 (10) = 10 % 13 = 10

Hash 2 (10) = 7 - (10 % 7) = 4

(Has1(10) + 1 * Hash2(10)) % 13 = 1

(Hash1(10) + 2 * Hash2 (10)) % 13 = 5

Collision