VARUN K

A016 70362019023

Assignment - I

Q1. Sequential search is a very simple method used for searching an array for a particular value. It works by comparing the value to be searched with every element of array one by one in a sequence until a match is found. It is mostly used to search an unordered list of elements.

```
S-I : SET POS = -1
S-II : SET I = 1
S-III : Repeat S-IV while I<=N
S-IV :  IF A[I] = VAL
            SET POS=I
            PRINT POS
            GOTO S-VI
        [END OF IF]
        SET I = I+1
    [END OF LOOP]
S-V : IF POS = -1
        PRINT "The Value Is Not Present"
    [END OF IF]
S-VI : EXIT
```

Linear search is rarely used practically because other search algorithms and hash tables allow significantly faster search comparison to linear search.

Q2. Insertion sort is a one very simple sorting algorithm in which the sorted array is build one after the element stores.

S-I: Repeat S-II to S-V
for K = 1 to N-1

S-II: SET TEMP = ARR[K]

S-III: SET J = K-1

S-IV: Repeat While TEMP <= ARR[J]
SET ARR[J+1] = ARR[J]
SET J = J+1
END OF INNER LOOP

S-V: SET ARR[J+1] = TEMP
[END OF LOOP]

S-VI: EXIT

12	32	9	5	4	28	50

S-I: A[0] is the only element in sorted list

12	32	9	5	4	28	50

S-II: PASS 1

4	12	32	9	5	4	28	50

12	32	9	5	4	28	50

S-III: PASS 2

9	12	32	5	4	28	50

S-IV: PASS 3

5	9	12	32	4	28	50

S-V: PASS 4

4	5	9	12	32	28	50

S-VI: PASS 5

4	5	9	12	28	32	50

S-VII - PASS 6

4	5	9	12	28	32	50

Q3 Selection sort algorithm sorts an array repeatedly finding the minimum element from unsorted part and putting it at the beginning. The algorithm maintains 2 subarrays in a given array

① Subarray which is already sorted

② Subarray which is unsorted

```
SMALLEST (ARR, K, N, POS)
S-I : SET SMALL = ARR[K]
S-II : SET POS = K
S-III : Repeat for J = K+1 to N-1
        IF SMALL > ARR[J]
        SET POS = J
        [END OF IF]
    [END OF LOOP]
S-IV : RETURN POS
```

```
SELECTION SORT (ARR, N)
S-I : Repeat S-II & III for K = 1 to N-1
S-II : CALL SMALLEST (ARR, K, N, POS)
S-III : SWAP A[K] with ARR[POS]
    [END OF LOOP]
S-IV : EXIT
```

arr[] = 12 24 32 5 11

S-I : Find min^n of ele. in arr[0....4] and place it at beginning

5 24 32 12 11

S-II : Find min^n ele in arr[1....4] and place at beginning of arr[1....4]

5 11 ~~24~~ 32 12 24

S-III : Find min^n ele. in arr[2...4] and place it at beginning of arr[2....4]

5 11 12 32 24

S-IV : Find min^m ele. in arr[3...4] and place it at beginning arr[3...4]

5 11 12 24 32

Q4. Shell sort is a sorting algorithm that is a generalisation of insertion sort. It performs in 2 steps:-

1. Arrange the elements of the array in form of table and sort the columns.
2. Repeat 1, each time with smaller number of longer columns in such a way that in the end there is only one column with all data sorted.

Shell_Sort(Arr, n)

S-I: SET FLAG=1, GAP_SIZE = N

S-II: Repeat S-III to VI while FLAG=1 OR GAP_SIZE>1

S-III: SET FLAG=0

S-IV: SET GAP_SIZE = (GAP_SIZE+1)/2

S-V: Repeat S-VI for I=0 to I <=(N-GAP_SIZE)

S-VI: IF ARR [I+ GAP_SIZE] > ARR[I]
SWAP Arr[I+ GAP_SIZE], Arr[I]
SET FLAG=0

S-VII: END

~~7 12~~ 32 63, 19, 7, 90, 81, 36, 54, 45, 72, 27, 22, 9, 41, 59, 33

Arrange the elements in form of table & sort

63 19 7 90 81 36 54 45
72 27 22 9 41 59 33

Result

63 19 7 9 41 36 33 45
72 27 22 90 81 59 54

Array = 63, 19, 7, 9, 41, 36, 33, 45, 72, 27, 22, 90, 81, 59, 54

Repeat S-I with smaller columns

63 19 7 9 41
36 33 45 72 27
22 90 81 59 54

Result: 22 19 7 9 27
36 33 45 59 41
63 90 81 72 54

Arr = 22, 19, 7, 9, 27, 36, 33, 45, 59, 41, 63, 90, 81, 72, 54

Repeat S-I with small no. of long columns

			Result :			
22	19	7		9	19	7
9	27	36		22	27	36
33	45	59		33	45	54
41	63	90		41	63	59
81	72	54		81	72	90

Arr = 9, 19, 7, 22, 27, 36, 33, 45, 54, 41, 63, 59, 81, 22, 90

Finally arrange the elements.

	Result :
9	
19	9
7	19
22	22
27	27
36	33
33	36
45	41
54	45
41	54
63	59
59	63
81	72
72	81
90	90

Q5 Merge Sort is a sorting algorithm that uses devide, conquer and combine paradigm.

```
MERGE (ARR, BEG, MID, END)
S-I: SET I=BEG, J=MID+1, INDEX=0
S-II: REPEAT while (I<=MID) AND (J<=END)
        IF ARR[I]< ARR[J]
            SET TEMP[INDEX]=ARR[I]
            SET I= I+1
        ELSE
            SET TEMP[INDEX]=ARR[J]
            SET J=J+1
      [END OF IF]
      SET INDEX= INDEX+1
   [END OF LOOP]
S-III: IF I>MID
        Repeat while I<=MID J<=END
            SET TEMP[INDEX]= ARR[J]
            SET INDEX=INDEX+1, SET J= J+1
       [END OF LOOP]
      ELSE
        Repeat while I<=MID
            SET TEMP[INDEX] = ARR[I]
            SET INDEX=INDEX+1, J=J+1
       [END OF LOOP]
   [END OF IF]
S-IV: SET K=0
S-V: Repeat while K<INDEX
            SET ARR[K]=TEMP[K]
            SET K=K+1
    [END OF LOOP]
S-VI : EXIT
```

MERGE_SORT (ARR, BEG, END)

S-I: IF BEG < END

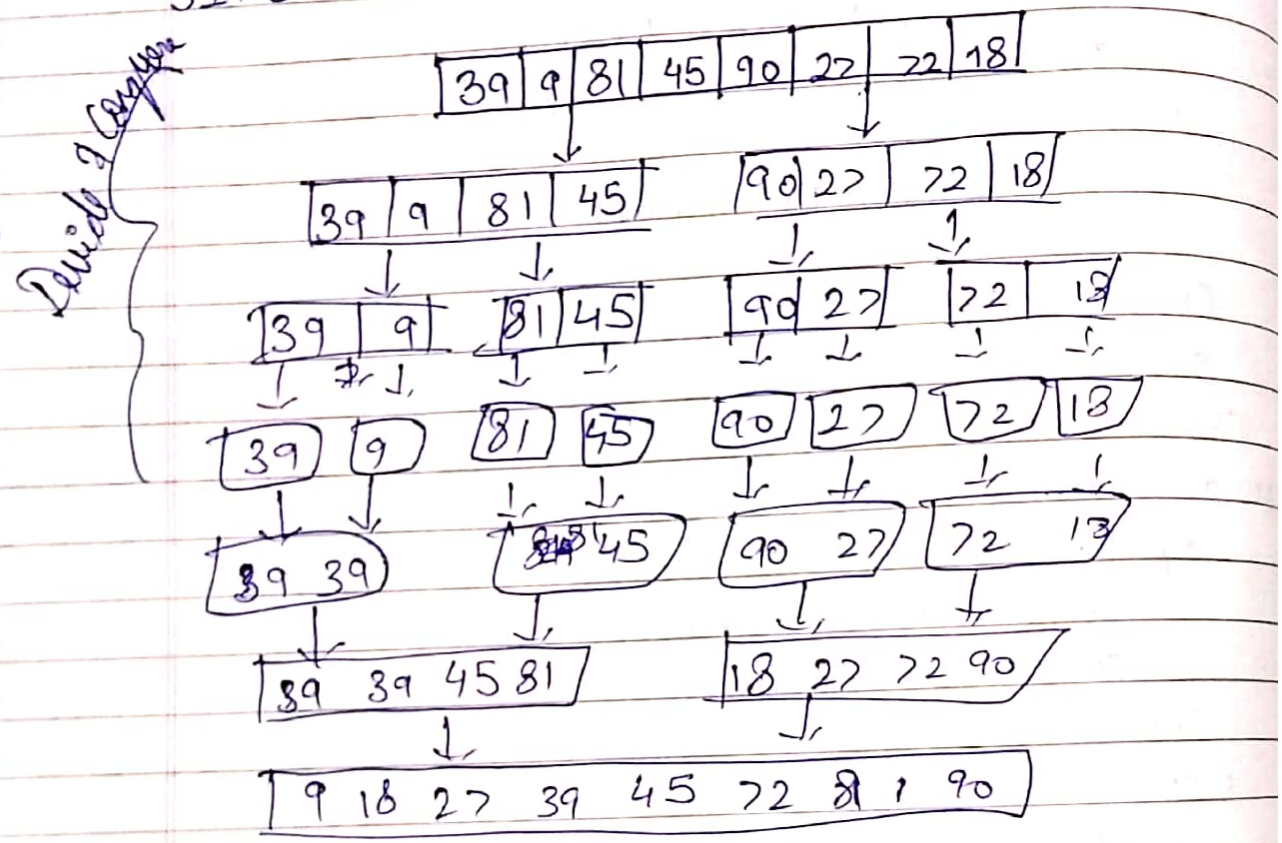
SET MID = (BEG+END)/2

CALL MERGE_SORT (ARR, MID+1, END)

MERGE (ARR, BEG, MID, END)

[END OF IF]

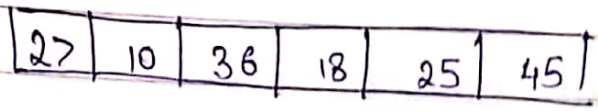
S-II: EXIT



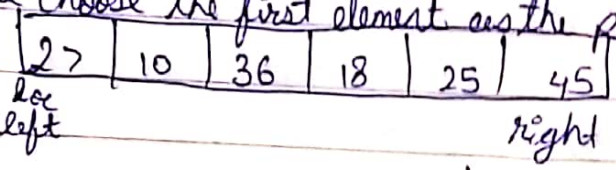
Q6. Quick sort is a widely used sorting paradigm.

```
§ PARTITION (ARR, BEG, END, LOC)
S-I: [INITIALIZE] SET LEFT = BEG, RIGHT = END, LOC = BEG, FLAG = 0
S-II: Repeat S-III to S-VI while FLAG = 0
S-III: Repeat while ARR[LOC] <= ARR[RIGHT] AND LOC != RIGHT
            SET RIGHT = RIGHT - 1
       [END OF LOOP]
S-IV: IF LOC = RIGHT
            SET FLAG = 1
      ELSE IF ARR[LOC] > ARR[RIGHT]
      SWAP  SWAP ARR[LOC] with ARR[RIGHT]
            SET LOC = RIGHT
      [END OF IF]
S-V: IF FLAG = 0
            REPEAT while ARR[LOC] >= ARR[LEFT] AND LOC != LEFT
            SET LEFT = LEFT + 1
           [END OF LOOP]
S-VI:      IF LOC = LEFT
                SET FLAG = 1
           ELSE IF ARR[LOC] < ARR[LEFT]
                SWAP ARR[LOC] with ARR[LEFT]
                SET LOC = LEFT
           [END OF IF]
      [END OF IF]
S-VII: [END OF LOOP]
S-VIII: END
```

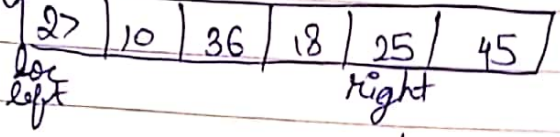
```
QUICK_SORT (ARR, BEG, END)
S-I: IF (BEG < END)
        CALL PARTITION (ARR, BEG, END, LOC)
        CALL QUICK SORT (ARR, BEG, LOC - 1)
        CALL QUICK SORT (ARR, LOC + 1, END)
     [END OF IF]
S-II: END
```



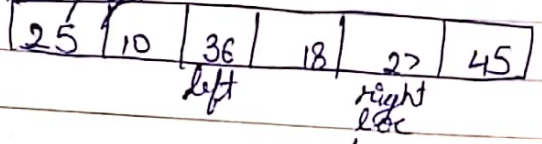
We choose the first element as the pivot. Set loc = 0, left = 0 a



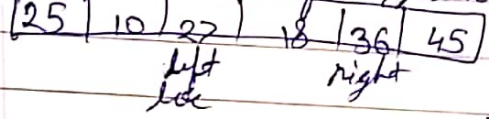
Scan from right to left. Since a[loc] < a[right], decrease the val
right



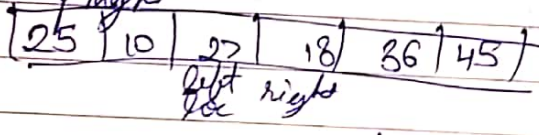
Start scanning from left to right. Since a[loc] > a[left], increased
value of left

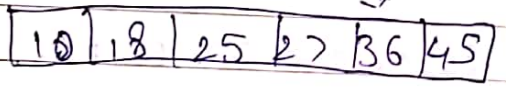


Since a[loc] < a[left], interchange the value and set loc = left



Scan from right to left. Since a[loc] < a[right], decreased the value of right





Q7. Heap_Sort(ARR, N)

S-I: [Build Heap H]

Repeat for I = 0 to N-1

Call Insert_heap (ARR, N, ARR[I])

[END OF LOOP]

S-II: (Repeatedly delete the root element)

Repeat while N > 0

Call Delete_Heap (ARR, N, VAL)

SET N = N+1

[END OF LOOP]

S-III: END

Q8. Hashing is a type of solution which can be used in almost all ~~conditions~~ situations. Hashing is a technique which uses less key comparisons and searches the element in O(n) time in the worst case and in average case it will be done in O(1) time. This method generally used the hash function to ~~keep~~ map the keys into a table, which is called the hash function. Hashing is an important data structure which is designed to use a special function called the hash function which is used to map a given value with a particular key for faster access of elements.