MORADABAD INSTITUTE OF TECHNOLOGY

NAME: NAMAN AGARWAL

ROLL No. : 1708210086

BRANCH : C.S.E

BATCH : B2

SESSION: 2019-20

SUBJECT: DESIGN and ANALYSIS of ALGORITHMS LAB

SUBJECT CODE : RCS-552

SUBMITTED TO:

Mrs. NEHA GUPTA MA'AM

INDEX

S. No.	PROGRAM / PRACTICAL	DATE	REMARKS & SIGNATURE
1.	a) WAP to implement Linear Search without recursion	07-08-2019	(10) neha 4/11/19
	b) Binary Search without recursion	07-08-2019	(10) neha 4/11/19
2.	a) Linear Search with recursion.	07-08-2019	(10) neha 4/11/19
	b) Binary Search with recursion.	07-08-2019	(10) neha 4/11/19
3.	Bubble Sort using functions.	14-08-2019	(10)* neha 2/11/19
4.	Selection Sort using functions.	14-08-2019	(10)* neha 2/11/19
5.	Radix Sort using functions.	21-08-2019	(10) neha 4/11/19
6.	Quick Sort using functions.	28-08-2019	(10) neha 18/11/19
7.	Merge Sort using functions.	04-09-2019	(10) neha 18/11/19
8.	Insertion sort using functions.	04-09-2019	(10) neha 18/11/19
9.	Counting Sort.	18-09-2019	(10)* neha 18/11/19
10.	Heap Sort	25-09-2019	(10) neha 9/10/19
11.	Shell Sort	09-10-2019	(10)* neha 9/10/19
12.	Knapsack Problem using Greedy algorithm.	16-10-2019	(10) neha 6/11/19
13.	Implementation of Kruskals algorithm.	06-11-2019	(10) neha 13/11/19
14.	Implementation of n-queen Problem.	13-11-2019	(10) neha 20/11/19

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

07/08/19

Program - 1(a)

Object: WAP to implement linear search without recursion.

Theory: Linear Search is a searching algorithm that compares each element of the sequence by the key element to check it's equality.

Algorithm: Linear Search (Array A, Value x)

1. Set i = 1
2. if i > n then go to step 7
3. if A[i] = x, go to step 6
4. set i = i+1
5. Go to step 2
6. Print Element Found, go to step 8.
7. Print Element Not found
8. Exit

Example: a = [2, 5, 3]
ITEM = 4
Element Not Found.

Source Code:

```
# include <stdio.h>
# define M 10
int main(void) {
    int i, a[M], nf=0;
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
printf("\n\n\t\t LINEAR SEARCH \n\n");
printf("Enter number of elements: \t");
scanf("%d", &n);
printf("Enter the elements: \n");
for (int i=0; i<n; i++)
    scanf("%d", &a[i]);
printf("Enter the key element: \t");
scanf("%d", &e);
for (int i=0; i<n; i++) {
    if (a[i] == e) { f=1;
        printf("Element Found :) \n");
        break;
    }
}
if (f==0) {
    printf("Element Not Found :( \n");
}
return 0
}
```

Output:

```
                LINEAR SEARCH
Enter number of elements: 1
Enter the elements:
5
Enter the key element : 0
Element not found :(
```

(10)

14/Aug/19

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

07/08/19

Program – 1(b)

Object: WAP to implement Binary search without recursion.

Theory: Binary Search is a quicker and efficient way to perform searching operation. Complexity for binary search is O(log n).

Algorithm: Binary Search (A, value)

1. Initialize start = 0 and end = length(A) – 1
2. $mid = \frac{(start + end)}{2}$
3. while (start <= end)
   do
   if value = A[mid]
   return mid
   else if value < A[mid]
   end = mid – 1
   else ~~value~~
   start = mid + 1
4. mid = (start + end)/2
5. return (–1)
6. Exit

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Example: a = [1, 2, 3, 6, 11, 15, 20]

key = 2

Element Found.

Source Code:

```
# include<stdio.h>
# define M 10

int main(void) {
    int l, k, n, t, a[M], init=0, end, mid;
    printf("\n\n\t\t BINARY SEARCH \n\n");
    printf("Enter the number of elements : \t");
    scanf("%d", &n);
    printf("Enter the elements: \n");
    for (int i=0; i<n; i++)
        scanf("%d", &a[i]);
    printf("Enter the key element: \t");
    scanf("%d", &l);
    // sorting steps
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
        }
    }
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
init=0;
end= n;
mid= (init + end)/2;
while (init <= end) {
        if (l== a[mid]) {
            n=-1;
            break;
        } else if (l< a[mid]) {
            end= mid-1;
        } else {
            init= mid+1;
        }
        mid= (init+end)/2;
    }
    if (n==-1) {
        printf ("Element Found :) \n");
    } else {
        printf ("Element not Found :( \n");
    }
    return 0;
}
```

Output:

```
BINARY SEARCH
Enter the number of elements: 3
Enter the elements:
5
1
7
Enter the key element: 0
Element not Found :(
```

10

neha
14/Aug/19

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

07/08/19

Program – 2(a)

Object: WAP to implement linear search using recursion.

Theory: Linear Search is a simple searching technique used to find/search an element in the array of unsort use. Complexity for linear search is of order of n.

Algorithm:

1. Start.
2. Set A[N+1] = ITEM
3. LOC = 1.
4. Repeat while A[LOC] != ITEM
   Set LOC = LOC + 1
5. If LOC = N+1, Set LOC = 0
6. Exit

Example:

a = [2, 1, 5, 6, 3, 9]
ITEM = 5
Element Found.

Source Code:

```
#include<stdio.h>
#define M 10
int linearSearch(int a[], int n, int index, int e) {
    int f=0, o;
    for (int j=index; j<n; j++) {
        if (a[j] == e) {
            f=1;
            o=j;
            break;
        } else {
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
            index += 1;
            linearSearch(a, n, index, l);
        }
    }
    if (l==-1) {
        return 0;
    } else {
        return -1;
    }
}
int main(void) {
    int l, a[M], n, k, index=0;
    printf("\n\n\t\t LINEAR SEARCH\n\n");
    printf("Enter number of elements: \t");
    scanf("%d", &n);
    printf("Enter the elements: \n");
    for (int i=0; i<n; i++)
        scanf("%d", &a[i]);
    printf("Enter the key element: \t");
    scanf("%d", &l);
    k= linearSearch(a, n, index, l);
    if (k!=-1) {
        printf("Element Found :) \n");
    } else {
        printf("Element not Found :( \n");
    }
    return 0;
}
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Output:

LINEAR SEARCH

```
Enter number of elements: 5
Enter the elements:
3
9
6
4
0
Enter the key element: 6
Element Found :)
```

14/Aug/19 (10)

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

07/08/19

Program-2(b)

Object: WAP to implement binary search using recursion.

Theory: Binary Search is relatively a faster algorithm to search a number in any sequence. The pre-condition for the search is, the sequence should be sorted anyways.

Algorithm: Binary Search (A, Value)

1. Initialize start = 0 and end = (length[A]-1)
2. Initialize mid = length[A]/2
3. if value = A[mid]
4. return (mid+1)
5. else if value < A[mid]
6. end = mid-1
7. else
   start = mid+1
8. mid = (start + end)/2
   Binarch Search(A, value)
9. return (-1)
10. Exit

Example: a = [3, 1, 5, 6, 0, 5, 9, 2, 4, 7, 8, 8, 5]

On sorting → a = [0, 1, 2, 3, 4, 5, 5, 5, 6, 7, 8, 8, 9]

Key element = 5

start = 0

end = 12 mid = (0+12)/2 = 6

∴ A[6] = 5 = key element. ∴ Element found.

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Source Code:

```
#include <stdio.h>
#define M 10

int init = 0;
int end;
int mid;
int f=0;
int binarySearch (int a[], int n, int l) {
    if (l == a[mid]) {
        f=1;
        return mid;
    } else if (l < a[mid]) {
        end = mid-1;
    } else {
        init = mid+1;
    }
    mid= (init + end)/2;
    if (init <= end) {
        return binarySearch (a,n,l);
    } else {
        return 0;
    }
}
int main(void) {
    int l,k,n,t,a[M];
    printf ("\n\n\t\t BINARY SEARCH \n\n");
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
printf("Enter the number of elements : \t");
scanf("%d", &n);
printf("Enter the elements : \n");
for (int i=0; i<n; i++)
    scanf("%d", &a[i]);
printf("Enter key element : \t");
scanf("%d", &l);
// sorting starts
for (int i=0; i<n; i++) {
    for (int j=0; j<n-1; j++) {
        if (a[j] > a[j+1]) {
            t = a[j];
            a[j] = a[j+1];
            a[j+1] = t;
        }
    }
}
// sorting ends
end = n;
k = binarySearch(a, n, l);
if (f == 1) {
    printf("Element Found :) \n");
} else {
    printf("Element not Found :( \n");
}
return 0;
}
```

use indentation

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Output:

BINARY SEARCH

```
Enter the number of elements : 6
Enter the elements:
3
5
1
0
7
2
Enter key element: 2
Element Found :)
```

10

14/Aug/19

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

14/oct/19

Program – 3

Object: WAP to implement bubble sort using function.

Theory: Bubble Sort is a sorting algorithm which induces the concept of 'rising of bubbles'. In other words, every element is compared to its adjacent element, and if it is found bigger, than the values are swapped.

Algorithm: BubbleSort (list/array)

1. Initialize i to 0 and j to 0.
2. If array[j] > array[j+1]
3. Swap elements.
4. Increment j, goto 2.
5. Increment i by one.
6. Exit.

Example: array[3] = | 9 | 7 | 3 |

Pass 1:

9	7	3

↓

7	9	3

↓

7	3	9

Pass 2:

7	3	9

↓

3	7	9

Exit

=> | 9 | 7 | 3 | sorted to | 3 | 7 | 9 |

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Source Code :

```
#include<stdio.h>
#define M 50

int a[M];

void bubSort(int n) {
    int t;
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
        }
    }
}

int main(void) {
    int n;
    printf("\n\n\t\t BUBBLE SORT \n\n");
    printf("Enter number of elements : \t");
    scanf("%od", &n);
    printf("Enter assay elements : \n");
    for (int i=0; i<n; i++)
        scanf("%od", &a[i]);
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
printf("Assay before Sorting :\n");
for (int i=0; i<n; i++)
    printf("%d\t", a[i]);
bubSort(n);
printf("\n Assay after Sorting :\n");
for (int=0; i<n; i++)
    printf("%d\t", a[i]);
printf("\n\n");
return 0;
}
```

Output:

```
BUBBLE SORT
Enter the number of elements : 3
Enter the assay elements:
9
0
4
Assay before Sorting:
9    0    4
Assay after Sorting:
0    4    9
```

10

7eh2 14/8/19

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

14/08/19

Program-04

Object: WAP to implement Selection sort using function.

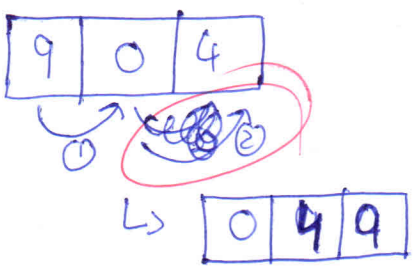
Theory: Selection sort is another type of sorting algorithm with $O(n^2)$. It implements the idea of searching the proper location for an element in the array and then swapping the values.

Algorithm: SELECTION SORT (Array)

1. Set min = 0 (location, i.e. array[0])
2. Search the minimum element in the array.
3. Swap with the min value.
4. Increment min to point to next element.
5. Repeat until list is sorted.

Example: Array[3] = | 9 | 0 | 4 |

Pass 1:



=> | 9 | 0 | 4 | sorted to → | 0 | 4 | 9 |

Source Code:

```
#include <stdio.h>

#define M 50

int a[M];
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
Void selectSort(int n) {
    int t, min;
    for (int i=0; i<n; i++) {
        min=0;
        while (a[i] > a[min])
            min+=1;
        t= a[i];
        a[i] = a[min];
        a[min] = t;
    }
}

int main(void) {
    int n;
    printf("\n\n\t\t SELECTION SORT \n\n");
    printf("Enter the number of elements: \t");
    scanf("%d", &n);
    printf("Enter the assay elements : \n");
    for (int i=0 ; i<n ; i++)
        scanf("%d", &a[i])
    printf("Assay before Sorting: \n");
    for (int i=0; i<n; i++)
        printf("%d", a[i])
    selectSort(n);
    printf("Assay after Sorting: \n");
    for (int i=0; i<n; i++)
        printf("%d\t", a[i]);
    printf("\n\n");
    return 0;
}
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Output :

SELECTION SORT

```
Enter the number of elements: 3
Enter the array elements:
9
0
4
Array Before Sorting:
9  0  4
Array After Sorting:
0  4  9
```

7ek 14/8/19 (10) *

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Program-05

Object: WAP to implement radix sort using functions.

Theory: Radix Sort is a sorting algorithm which involves digit-by-digit sorting methodology, starting with the least significant digit. It relay on other stable sorting algorithm to compare the digits and hence sort them.

Algorithm: RadixSort(A[], n, d) {

1. Initialize i=1.
2. While i<=d repeat steps 3 to 5.
3. Extract i^{th} digit from each number.
4. Sort on the basis of these i^{th} digits.
5. Increment i
6. Exit }

Example: A = [397, 243, 529]

d = 3.

i=1

i ↓

3	9	7
2	4	3
5	2	9

⇒

2	4	3
3	9	7
5	2	9

i++.

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

i=2

	i				
2	4	3		5	2	9
3	9	7	⇒	2	4	3
5	2	9		3	9	7

i++

i=3=d Last iteration.

i					
5	2	9		2	4	3
2	4	3	⇒	3	9	7
3	9	7		5	2	9

∴ Sorted A = [243, 397, 529]

Source Code:

```
#include<stdio.h>
#define M 20

int max_length(int a[], int n) {
    int k=0, count=0;
    for (int i=0; i<n; i++) {
        if (k<a[i]) {
            k=a[i];
        }
    }
    while (k!=0) {
        k=k/10;
        count+=1;
    }
    return count;
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
Void radixSort(int a[], int n, int d) {
    int t=10, i=0, n_val_var, in_val_var, s;
    while (d != 0) {
        for (int i=0; i<n; i++) {
            for (int j=0; j<n-1; j++) {
                n_val_var = a[j] % t;
                in_val_var = a[j+1] % t;
                if (n_val_var > in_val_var) {
                    s = a[j];
                    a[j] = a[j+1];
                    a[j+1] = s;
                }
            }
        }
        t = t*10;
        d -= 1;
    }
    printf("\n Sorted Array: \n");
    for (int j=0; j<n; j++)
        printf("%d \t", a[j]);
    printf("\n\n");
}

int main(void) {
    int a[M], n, d;
    printf("\n\n\t\t RADIX SORT \n\n");
    printf("Enter the number of elements: \t");
    scanf("%d", &n);
    printf("Enter the array: \n");
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
for (int i=0; i<n; i++)
    scanf("%d", &a[i]);
d = max_length(a,n);
radixSort(a,n,d);
return 0;
}
```

Output:

RADIX SORT

Enter the number of elements: 3

Enter the array:

39548

9999

123

Sorted Array:

123 9999 39548

10

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

28/08/19

Program-06

Object: WAP to implement quick sort using functions.

Theory: Quick sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. Best case complexity for quick sort is $\Omega(n \log n)$, while for worst case is $O(n^2)$.

Algorithm: Partition (A, p, r)

1. Initialize $x = A[r]$
2. Initialize $i = p-1$
3. Initialize $j = p$, and repeat steps 4 to 6 until $j < r$
4. if $A[j] \leq x$ do step 5 and 6.
5. Increment i
6. exchange $A[i] \longleftrightarrow A[j]$
7. exchange $A[i+1] \longleftrightarrow A[r]$.
8. return $i+1$
9. Exit

Quick Sort (A, p, r)

1. Initialize q.
2. if $p < r$ do steps 3 to 5, else go to step 6.
3. q = Partition (A, p, r)
4. QuickSort (A, p, q-1)
5. QuickSort (A, q+1, r)
6. Exit

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Example: A → | 5 | 4 | 3 |
1 2 3

QuickSort (A, 1, 3)

↓ 1<3 ⇒ true

q = Partition (A, 1, 3)

↓

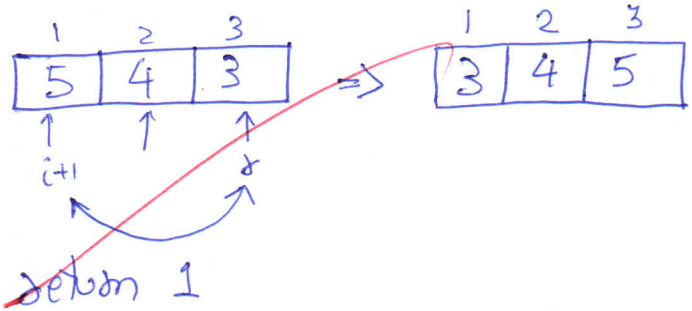
x = 3

i = 0

j = 1 $5 \leq 3$ False

j = 2 $4 \leq 3$ False

j = 3 Loop Exit



Return 1

↓ q = 1

QuickSort (A, 1, 0)

1 < 0

False

∴ Exit

↓

QuickSort (A, 2, 3)

2 < 3

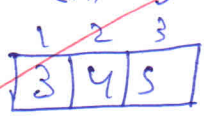
True

↓

q = Partition (A, 2, 3)

⇓

q = 2.



↓

QuickSort (A, 2, 1) → QuickSort (A, 3, 3)

2 < 1 False ∴ Exit 3 < 3 False ∴ Exit.

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Source Code:

```
#include<stdio.h>
#include<stdlib.h>

void swap(int *b, int *c){
    int t;
    t = *b;
    *b = *c;
    *c = t;
}

int partition(int a[], int p, int r){
    int x, i;
    x = a[r];
    i = p-1;
    for(int j=p; j<r; j++){
        if(a[j] <= x){
            i += 1;
            swap(a+i, a+j);
        }
    }
    swap(a+i+1, a+r);
    return i+1;
}

void quickSort(int a[], int p, int r){
    int q;
    if(p<r){
        q = partition(a,p,r);
        quickSort(a,p,q-1);
        quickSort(a,q+1,r);
    }
}
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
int main (void) {
    int r, p=1;
    int *a;
    printf ("\n\n\t\t QUICK SORT \n\n");
    printf ("Enter the number of elements : \t");
    scanf ("%d", &r);
    a= (int *) malloc (r* sizeof (int));
    printf ("Enter the array elements : \n");
    for (int i= 1; i<= r; i++)
        scanf ("%d", &a[i]);
    quickSort (a, p, r);
    printf ("Sorted Array : \n");
    for (int i= 1; i<= r; i++)
        printf ("%d\t", a[i]);
    free (a);
    return 0;
}
```

Output:

QUICK SORT

Enter the number of elements: 3

Enter the array elements:

5 3 2 4

3 9

1 3 4 5

Sorted Array:

3 9 5 3 2 4 1 3 4 5

18/9/19 (10)

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

04/09/19

Program-07

Object: WAP to implement Merge Sort using functions.

Theory: Merge Sort is another sorting technique based on the strategy of divide-and-conquer. The technique involves recursively splitting the array into two equal halves until we get a linear array which is already sorted. The merging of the splitted parts take place. Complexity for merge-sort is $O(n \log n)$.

Algo: MergeSort(A, p, r)

1. if p<r perform steps 2-5, else exit.
2. q = (p+r)/2.
3. Recursively call MergeSort(A, p, q).
4. Recursively call MergeSort(A, q+1, r).
5. Call Merge(A, p, q, r) routine.

Merge(A, p, q, r)

1. Initiate n1 = q-p+1.
2. Initiate n2 = r-q.
3. Create two sub-arrays → L[1... n1+1] and R[1... n2+1].
4. For i=1 to n1 perform step 5
5. L[i] = A[p+i-1].
6. For j=1 to n2 perform step 7
7. R[i] = A[q+j].
8. Assign both L[n1+1] and R[n2+1] equal to infinity.
9. Initiate i=1 and j=1
10. For k=p to r perform steps 11 to 16
11. If L[i] ≤ R[j] do steps 12 and 13
12. Assign A[k] = L[i]
13. Increment i by 1.

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

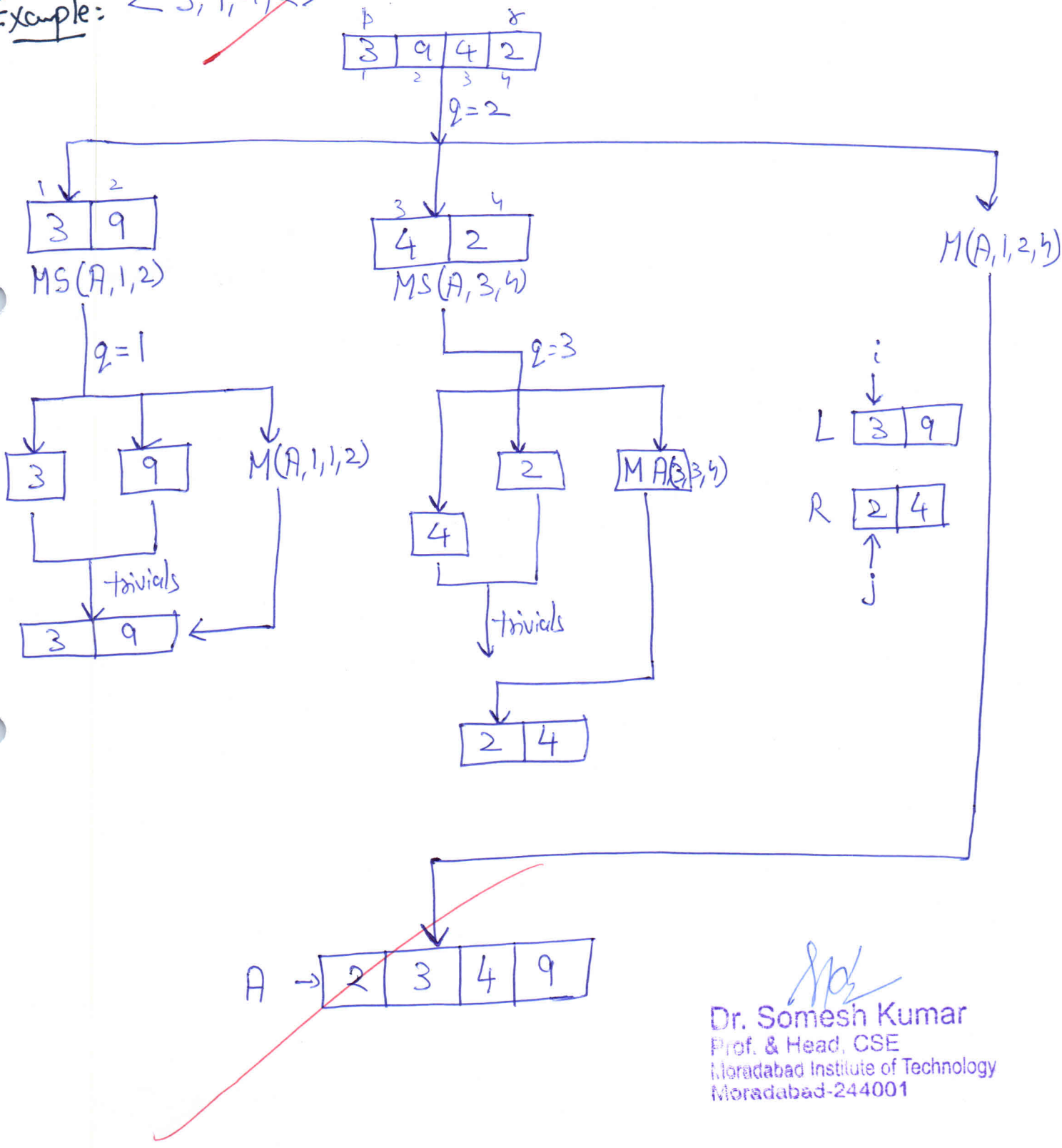
14. Else do steps 15 and 16.

15. Assign A[k] = R[j]

16. Increment j by 1.

17 Exit.

Example: <3,9,4,2>



Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Source Code:

```
#include<stdio.h>
#include<stdlib.h>

void merge(int *a, int p, int q, int r) {
    int n1, n2, s=1, t=1;
    n1 = q-p+1;
    n2 = r-q;
    int L[n1+1], R[n2+1];
    for (int i=1; i<=n1; i+=1)
        L[i] = *(a+(p+i-1));
    for (int i=1; i<=n2; i+=1)
        R[i] = *(a+(q+i));
    L[n1+1] = 10000;
    R[n2+1] = 10000;
    for (int k=p; k<=r; k++) {
        if (L[s] <= R[t]) {
            *(a+k) = L[s];
            s+=1;
        } else {
            *(a+k) = R[t];
            t+=1;
        }
    }
}
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
Void mergeSort (int *a, int p, int r) {
    int q;
    if (p<r) {
        q= (p+r)/2;
        mergeSort (a,p,q);
        mergeSort (a, q+1, r);
        merge (a,p,q,r);
    }
}
int main (void) {
    int r, p=1;
    int *a;
    printf ("\n\n\t\t MERGE SORT \n\n");
    printf ("Enter the number of elements: \t");
    scanf ("%d", &r);
    a= (int *) malloc (r* sizeof (int));
    printf ("Enter the elements: \n");
    for (int i=1; i<=r; i++)
        scanf ("%d", &a[i]);
    mergeSort (a, p, r);
    printf ("Sorted Array is: \n");
    for (int i=1; i<=r; i++)
        printf ("%d\t", *(a+i));
    printf ("\n\n");
    return 0;
}
```

Output: MERGE SORT

Enter the number of elements: 3

Enter the elements:

9

0

7

Sorted Array is:

0 7 9

18/9/19 (10)

Dr. Somesh Kumar
Prof. & Head, CSE
Ghaziabad Institute of Technology
[illegible]-244001

04/09/19

Program-08

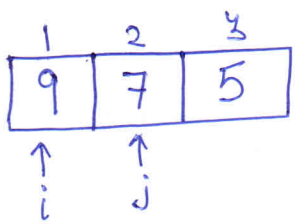
Object: WAP to implement insertion sort using functions.

Theory: Insertion sort is an in-place comparision-based sorting algorithm. It works in the similar fashion as we sort playing cards in our hand. With every iteration, an element is placed (inserted) at it's correct position. $\Omega(n)$ and $O(n^2)$.

Algorithm: Insertion Sort (A)

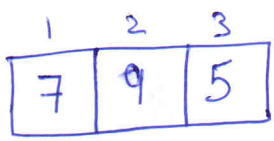
1. For j=2 to length of (A) do steps 2 to 7.
2. Assign key = A[j];
3. Initiate i=j-1.
4. While i>0 and A[i] >key do steps 5 to 6.
5. Assign A[i+1] = A[i].
6. Decrement 'i' by 1.
7. Assign A[i+1] = key.
8. Exit.

Example: j=2

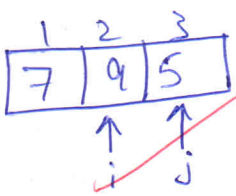


key 7

⇒

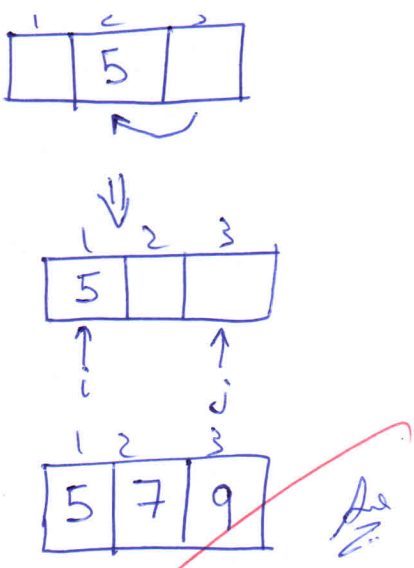


j=3



key 5

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001



Source Code:

```
# include<stdio.h>
# include <stdlib.h>

Void insertionSort(int *a, int n) {
    int key, i;
    for (int j = 2; j<=n; j+=1) {
        key = *(a+j);
        i = j-1;
        while ( (i>0) && (*(a+i) > key) ) {
            *(a+i+1) = *(a+i);
            i-=1;
        }
        *(a+i+1) = key;
    }
}
int main (void) {
    int n;
    int *a;
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
printf("\n\n\t\t INSERTION SORT \n\n");
printf("Enter number of elements : \t");
scanf("%d", &n);
a = (int *) malloc (n * sizeof(int));
printf("Enter the array elements : \n");
for (int i=1; i<=n; i++)
    scanf("%d", a+i);
insertion Sort (a,n);
printf("Sorted Array is: \n");
for (int i=1; i<=n; i++)
    printf("%d \t", *(a+i));
free (a);
return 0;
}
```

Output:

INSERTION SORT

Enter number of elements: 6

Enter the array elements:

6
3
5
7
2
4

Sorted Array is:

2 3 4 5 6 7

18/9/19 (10)

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

18/09/2019

Program-09

Object: WAP to implement Counting Sort.

Theory: Counting Sort is a traversal sorting algorithm which involves usage of three arrays. The final sorted array is made different from the actual array. Counting sort is a stable sorting algorithm with linear complexity, i.e. O(n).

Algorithm: Counting_Sort (A, B, k)

1. Set C[0...k] be a new array.
2. from i=0 to k
3. C[i]=0
4. for j=1 to A_length
5. C[A[j]] = C[A[j]] + 1
6. for i=1 to k
7. C[i] = C[i] + C[i-1]
8. for j= A_length to 1
9. B[C[A[j]]] = A[j]
10. C[A[j]] -= 1.

Example:

A =

5	3	1
1	2	3

C =

0	1	0	1	0	1
0	1	2	3	4	5

C =

0	~~1~~ 0	1	~~2~~ 1	2	~~3~~ 2
0	1	2	3	4	5

⇒ C =

0	0	1	1	2	2
0	1	2	3	4	5

B =

1	3	5
1	2	3

(Sorted)

j	A[j]	C[A[j]]	B[C[A[j]]]	C[A[j]] -1
3	A[3]=1	C[1]=1	B[1]=1	C[1]=0
2	A[2]=3	C[3]=2	B[2]=3	C[3]=1
1	A[1]=5	C[5]=3	B[3]=5	C[5]=2

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Source Code:

```
#include<stdio.h>
#include<stdlib.h>

int main(void) {
    int n, k=0;
    int *a, *b, *c;
    printf("\n\n\t\t COUNTING SORT \n\n");
    printf("Enter number of elements: \t");
    scanf("%d", &n);
    a= (int *) malloc (n* sizeof(int));
    b= (int *) malloc (n* sizeof(int));
    printf("Enter the array elements: \n");
    for (int i=1; i<=n; i++) {
        scanf("%d", a+i);
        if (*(a+i)>k) {
            k= *(a+i)
        }
    }
    c= (int *) malloc (k* sizeof(int));
    for (int i=0; i<=k; i++)
        c[i]=0
    for (int i=1; i<=n; i++)
        c[a[i]]+=1
    for(int i=1; i<=k; i++)
        c[i]+= c[i-1]
    for (int i=n; i>=1; i--) {
        b[c[a[i]]] = a[i];
        c[a[i]]--;
    }
    printf("\n Sorted Array is: \n");
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
for (int i=1; i<=n; i++)
    printf ("%d\t", *(b+i));
free (a);
free (b);
free (c);
return 0;
}
```

Output:

COUNTING SORT

Enter the array elements:

2 5 3 0 2 3 0 3

Sorted Array is:

0 0 2 2 3 3 3 5

7ele 18/9/19 (10) *

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

25/09/2019

Program-10

Object: WAP to implement Heap Sort.

Theory: Heap Sort is a comparision based sorting algorithm based on Binary Heap Data Structure. Heap Sort works for both cases of ascending and descending order of sorting. It utilizes the Max Heap properties for the first, while Min Heap properties for the later. The time complexity for Heap Sort is $O(n \log n)$, where n is the size of the heap.

Algorithm: Heap_Sort (A)

1. Build_Max_Heap (A)
2. for (length (A) to 2) do
3. exchange A[1] ⟷ A[i]
4. heap_size --= 1
5. Max_Heapify (A, 1)

Build_Max_Heap (A)

1. heap_size (A) ← length (A)
2. for $i \leftarrow \left\lfloor \frac{heap\_size(A)}{2} \right\rfloor$ to 1 do
3. Max_Heapify (A, i)

Max_Heapify (A, i) // A → heap ; i → node index

1. $l \leftarrow 2i$
2. $r \leftarrow 2i+1$
3. if $l <= heap\_size(A)$ and $A[l] > A[i]$ do step 4
4. largest ← l
5. else do step 6
6. largest ← i

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

7. if $r \leq heap\_size(A)$ and $A[r] > A[i]$ do step 8

8. $largest \leftarrow r$

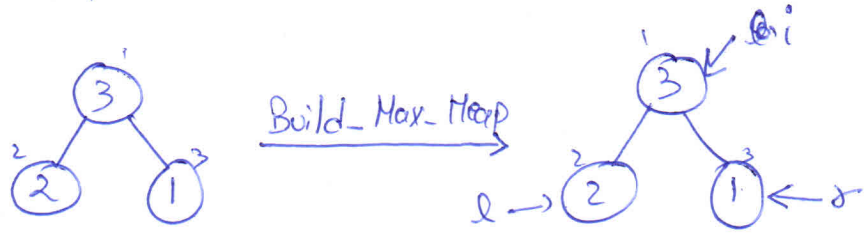
9. if $largest \neq i$ do steps 10 and 11

10. Exchange $A[largest] \longleftrightarrow A[i]$

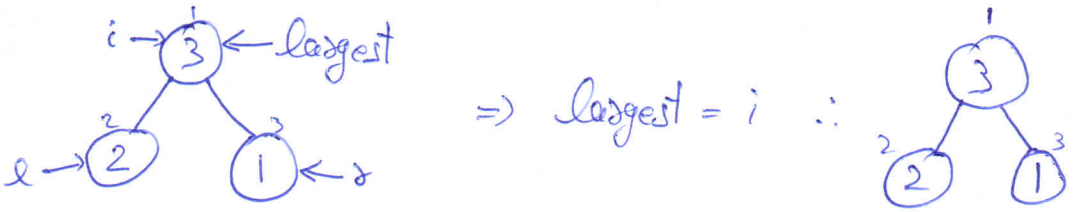
11. Max_Heapify (A, largest)

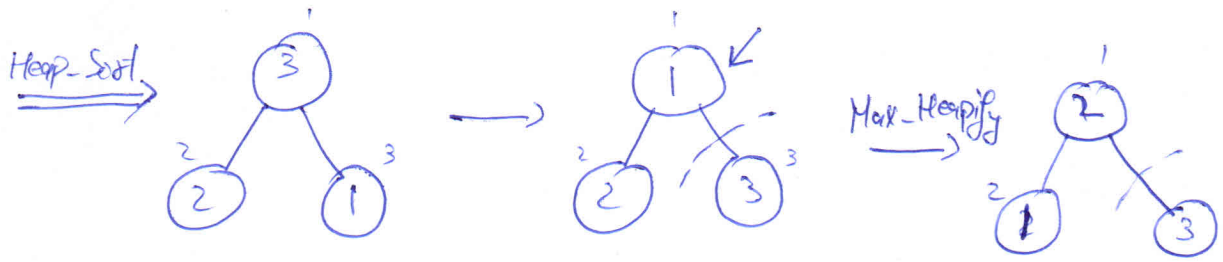
Example:

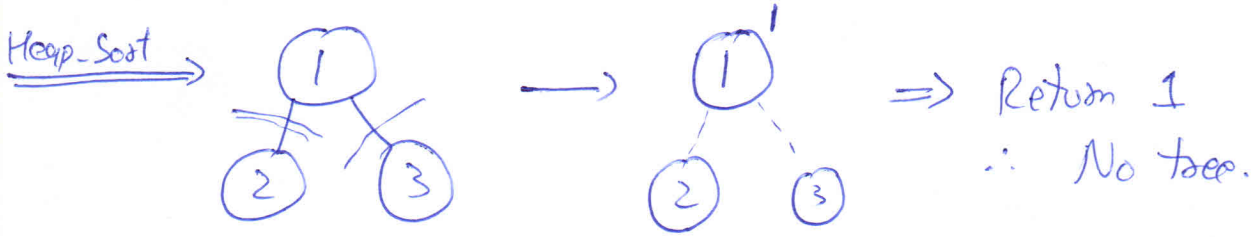
$A = <3, 2, 1>$



$A[i] > A[l] \therefore largest \leftarrow i$







∴ Output → 3 2 1

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

P.T.O

Source Code:

```
# include<stdio.h>
# include<stdlib.h>
int heap_size;
Void swap (int *x, int *y) {
    int t;
    t= *x;
    *x= *y;
    *y= t;
}

Void maxHeapify (int *a, int i) {
    int largest, l, r;
    l= 2*i;
    r= 2*i+1;
    if((l<=heap_size) && (a[l]>=a[i])) {
        largest = l;
    } else {
        largest= i;
    }
    if ((r<=heap_size) && (a[r] > a[largest])) {
        largest = r;
    }
    if (largest != i) {
        swap (a+largest, a+i);
        maxHeapify (a, largest);
    }
    return;
}
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
Void buildMaxHeap (int *a) {
    int k;
    k = heap_size/2;
    for (int i = k; i >= 1; i--) {
        maxHeapify (a,i);
    }
}

/*Void maxHeapify*/
void HeapSort (int *a, int n) {
    buildMaxHeap (a);
    for (int i = n; i >= 2; i--) {
        Swap (a+1, a+i);
        heap_size -= 1;
        maxHeapify (a,1);
    }
}
int main (void) {
    int *a;
    int size;
    printf ("\n\n\t\t HEAP SORT \n\n");
    printf (Enter heap size: \t");
    Scanf ("%d", &size);
    heap_size = size;
    a = (int *) malloc (size * sizeof (int));
    for (int i=1; i<=size; i++) {
        Scanf ("%d", a+i);
    }
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
HeapSort (a, size);
printf ("\n Sorted Array is: \n");
for (int i = 1; i <= size; i++)
    printf ("%d \t", *(a+i));
return 0;
}
```

Output:

HEAP SORT

Enter heap size : 12

7 4 6 5 19 12 13 17 11 10 3 2

Sorted Array is :

2 3 4 5 6 7 10 11 12 13 17 19

19

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

09/10/19

Program-11

Object: WAP to implement Shell Sort.

Theory: Shell sort is a generalized version of insertion sort in which the comparisions are started between elements present at a distance, known as a "gap" in technical terms. However, shell sort does not have a good impact on complexity as of the insertion sort, but improves it a little. The shell sort improves the time complexity from $O(n^2)$ as in insertion sort to $O(n^{1.7})$ in shell sort.

Algorithm: Shell_Sort()

1. Initialize the value of gap = array_size/2
2. Divide the list/array into smaller sub-arrays of size gap.
3. Sort the individual sub-arrays using insertion sort.
4. Repeat until complete array is sorted.

Example:

A

0	1	2
3	2	1

gap = ⌊3/2⌋ = 1.

A

0	1	2
3	2	1

(i, j) → A[i] > A[j] ∴ swap

A

0	1	2
2	3	1

(i, j) → A[i] > A[j] ∴ swap

A

0	1	2
2	1	3

(i, j) → A[i] < A[i-gap] ∴ swap

A

0	1	2
1	2	3

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
Void swap (int *k, int *l) {
    int t;
    t = *k;
    *k = *l;
    *l = t;
}

int main (void) {
    int n, i, j, k, l, gap;
    int *a;
    printf("\n\n\t\t SHELL SORT \n\n");
    printf ("Enter the number of elements: \t");
    scanf ("%d", &n);
    a = (int *) malloc (n * sizeof(int));
    printf ("Enter the elements: \n");
    for (int i=0; i<n; i++)
        scanf ("%d", a+i);
    gap = n/2;
    while (gap >=1) {
        i = 0;
        j = gap;
        while (j<n) {
            if (*(a+i) > *(a+j)) {
                swap (a+i, a+j);
                if (gap <= n/2) {
                    l = i-gap;
                    k = i;
                    while (l>=0) {
                        if (*(a+l) > *(a+k)) {
                            swap(a+l, a+k);
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
            }
            k=l;
            l=l-gap;
        }
    }
    i++;
    j++;
  }
  gap=gap/2;
}
printf("\n Sorted Array is : \n");
for (int z=0; z<n; z++)
    printf("%d\t", *(a+z));
free(a);
return 0;
}
```

Output:

SHELL SORT

Enter the number of elements: 6

Enter the elements:

40 30 20 10 15 7

Sorted Array is:

7 10 15 20 30 40

10

neha 9/10/19

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

16/10/2019

Practical-12

Object: Knapsack problem using greedy algorithm.

Theory: Greedy algorithms are simple and straight forward. They are short sighted in their approach in the sense that they take decisions on the basis of information currently known to them irrespective of the impacts that the decision may have in the future.

Algorithm: Greedy_Fractional_Knapsack(w[0...n-1], v[0...n-1], p[0...n-1], W)

1. Initiate V=0.
2. for i=0 to n-1 do
3. if p[i] < p[i+1] do
4. exchange (p[i] and p[i+1])
   exchange (w[i] and w[i+1])
   exchange (v[i] and v[i+1])
5. while (W != 0) do
6. if w[k] <= W do
7. W -= w[k]
   V+= v[k]
8. else find fractional value upto W and do
   V+= fractional value
9. increment k by 1.
10. Exit.

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Example:

Items	Weight	Value	Profit = Value/Weight
I_1	10	90	9
I_2	20	100	5
I_3	30	170	5.66

after sorting →

Items	Weight	Value	Profit
I_1	10	90	9
I_3	30	170	5.66
I_2	20	100	5

$W = 10$.

for $I_1 \rightarrow 10 = 10$, $\therefore V = 9$, $W = 0$ exit.

$\therefore$ **V = 9** Ans

Source Code:

```
#include<stdio.h>
#include<stdlib.h>

void swap(int *a, int *b) {
    int t;
    t = *a;
   *a = *b;
   *b = t;
}

void main() {
    int n, w, V=0, k=0, p;
    int *w, *v, *p;
    printf("\n\n\t\t KNAPSACK PROBLEM \n\n");
    printf("Enter the number of items: \n");
    scanf("%d", &n);
    w = (int*) malloc(n * sizeof(int));
    v = (int*) malloc(n * sizeof(int));
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
p=(int *) malloc (n*sizeof(int));
printf("Enter each item's weight:");
for (int i=0; i<n; i++)
    scanf ("%d", w+i);
printf("Enter each item's value:");
for (int i=0; i<n; i++)
    scanf("%d", v+i);
for (int i=0; i<n; i++)
    *(p+i) = (*(v+i)) / (*(w+i));
printf("Enter the bag size: \t");
scanf ("%d", &W);
for (int i=0; i<n; i++) {
    for (int j=0; j<n-1; j++) {
        if (*(p+j) < *(p+j+1)) {
            Swap (p+j, p+j+1);
            Swap (w+j, w+j+1);
            Swap (v+j, v+j+1);
        }
    }
}
while (W>=0) {
    if ( W>=(*(w+k)) ) {
        W-=(*(w+k));
        V+=(*(v+k));
    } else {
        q=(*(v+k)) / (*(w+k));
        q*=W;
        V+=q;
        W=0;
    }
    k++;
}
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
printf("\n The Sack Value is: \t%d", V);
free(w);
free(w);
free(p);
}
```

Output:

KNAPSACK PROBLEM

Enter the number of items: 5

Enter each item's weight:

5 10 20 30 40

Enter each item's value:

30 20 100 90 160

Enter the bag size: 60

The Sack value is: 270

10

6/11/19

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

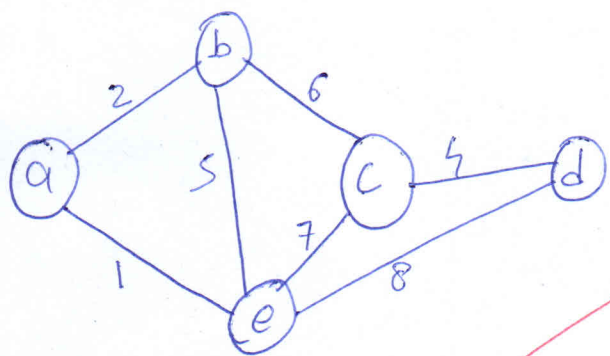
06/4/19

Program-13

Object: Implementation of Kruskal's algorithm.

Theory: Kruskal's algorithm is a minimum-spanning-tree (MST) algorithm which finds the shortest path from a given source vertex to every other node in the graph. Kruskal's algorithm is a type of Greedy approach for a solution to a problem.

Example:



Let source vertex is (a)

→ List of edges:

(a,b) : 2
(a,e) : 1
(b,e) : 5
(b,c) : 6
(e,c) : 7
(c,d) : 4
(e,d) : 8

Sorting in non-decreasing order →

(a,e) : 1 ✓
(a,b) : 2 ✓
(c,d) : 4 ✓
(b,e) : 5
(b,c) : 6
(e,c) : 7
(e,d) : 8

→ Computing set of edges in MST-

Cost = 1+2+4 = 7.

$x = \{a, e, b, c, d\}$

∴ MST

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Algorithm: Kruskal_Algorithm (G, w)

```
1. A ← φ
2. for each vertex v ∈ V[G] do
3.     MAKE_SET(v)
4. Sort the edges of E into non-decreasing order by weight w.
5. for each edge (u,v) ∈ E, taken in non-decreasing order by weight do
6.     if FIND-SET(u) ≠ FIND_SET(v) then
7.         A ← A ∪ {(u,v)}
8.             UNION (u,v)
9. return A
```

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
#define max 50
int x[max];
int pos=0;

Struct node {
    int w;
    int s;
    int d;
};

Void addEdge (int m, int n) {
    pos+=1;
    x[pos]=m;
    pos+=1;
    x[pos]=n;
}

int check (int f) {
    for (int i=0; i<pos; i++) {
        if (x[i]==f) {
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
            return 1;
        }
    }
    return 0;
}

int checkEdge(int u, int v) {
    if((check(u)==1) && (check(v)==1)) {
        return 0; // edge present
    } else {
        return 1; // edge absent
    }
}

int main(void) {
    int n, p, q, r, cost=0;
    struct node *a, t;
    printf("KRUSKAL'S ALGORITHM \n\n");
    printf("Enter the number of edges: \t");
    scanf("%d", &n);
    a=(struct node *) malloc(n * sizeof(struct node));
    printf("Enter the graph (weight, source, destination): \n");
    for(int i=0; i<n; i++) {
        scanf("%d \t%d \t%d", &p, &q, &r);
        (a+i)->w=p;
        (a+i)->s=q;
        (a+i)->d=r;
    }
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
for (int i=0; i<n; i++) {
    for (int j=0; j<n-1; j++) {
        if ( ((a+j)->w) > ((a+j+1)->w) ) {
            t = *(a+j);
            *(a+j) = *(a+j+1);
            *(a+j+1) = t;
        }
    }
}

for (int i=0; i<n; i++) {
    if ( (checkEdge((a+i)->s, (a+i)->d)) == 1) {
        cost += ((a+i)->w);
        addEdge ((a+i)->s, (a+i)->d);
    }
}
printf("Cost of MST : %d", cost);
return 0;
}
```

~~Output:~~

~~KRUSKAL'S ALGORITHM~~

~~Enter the graph (weight, source, destination):~~

~~4 0 1~~

~~8 0 7~~

/

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

Output:

KRUSKAL'S ALGORITHM

```
Enter the number of edges : 14
Enter the graph (weight, source, destination) :
4 0 1
8 0 7
11 1 7
8 1 2
7 3 2
9 3 4
10 4 5
14 3 5
4 2 5
2 5 6
1 6 7
7 7 8
6 6 8
2 2 8

Cost of MST : 35
```

13/11/19 (10)

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

13/4/2019

Program-14

Object: Implementation of n-queen problem.

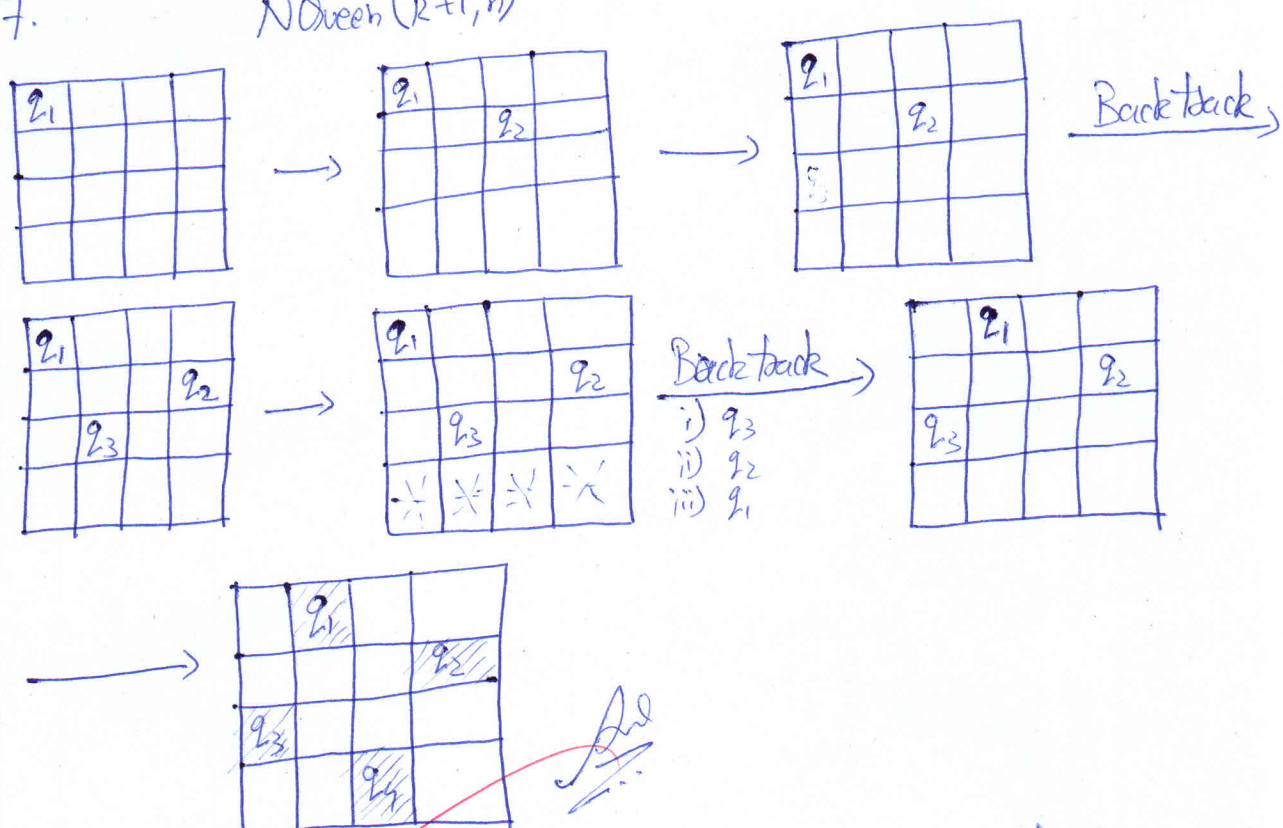
Theory: n-queen problem from Backtracking approach says that we have to place n-queens on a nxn chessboard such that no queen is in attacking position with respect to any queen. We can backtrack atmost one place at a time.

Algorithm:

```
Algorithm_NQueen(k,n)
1. for (i=1 to n) do
2.     if place(k,i) then
3.         x[k]=i
4.         if (k==n) then
5.             write (x[1...n]);
6.     else
7.         NQueen(k+1,n)
```

```
Algorithm_place(k,i)
1. for (j=1 to k-1) do
2.     if((x[j]=i) or (Abs(x[j]-i)=Abs(j-k)))
3.         return false.
4. return true.
```

Example:



∴ One of the solution vector:

$$x[2, 4, 1, 3]$$

Dr. Somesh Kumar
Prof. & Head, CSE
Institute of Technology
244001

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
int *x;
void print(int n) {
    for(int i=1; i<=n; i++)
        printf("%d\t", x[i]);
}
int place(int k, int i) {
    for(int j=1; j<k; j++)
        if((*(x+j)==i) || (abs(*(x+j)-i)==abs(j-k))) {
            return 0;
        }
    return 1;
}
void nQueen(int k, int n) {
    for(int i=1; i<=n; i++) {
        if(place(k,i)==1) {
            *(x+k)=i;
            if(k==n) {
                print(n);
            } else {
                nQueen(k+1,n);
            }
        }
    }
}
```

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001

```
int main(void) {
    int n;
    printf("Enter the number of queens: \t");
    scanf("%d", &n);
    x = (int *) malloc (n * sizeof(int));
    nQueen(1, n);
    free(x);
    return 0;
}
```

Output: Enter the number of queens : 4

2 4 1 3 3 1 4 2

zeke 20111119 (10)

Dr. Somesh Kumar
Prof. & Head, CSE
Moradabad Institute of Technology
Moradabad-244001