Course Outcomes

At the end of the course, the student will be able to:

CO1. Identify the design technique used in an algorithm.
CO2. Analyse algorithms using quantitative evaluation.
CO3. Design and implement efficient algorithms for practical and unseen problems.

Instructions:

1. Language: C
2. Do not modify the driver code

NAME : NAGAVENI L G
SEC : 4F
SRN:PES2UG21CS315

Ticking Session 4

Q1. A Matrimony assembles a group of equal men and women. They are told to make 2 lines parallel. One for men and another for women. The matrimony then decides to have them in the order of their age. To avoid inconvenience and to form the line quickly, the members of the line do a minimum number of exchange of positions to form the line as required. The matrimony then says to marry each other if the corresponding man and woman facing each other are of the same age. Find the number of people who marry each other.

Sample Input 1:

```
4                   // Number of people in a line
30 28 27 24         // Ages of boys in the line
23 27 28 29         // Ages of girls in the line
```

Expected Output 1:

```
2
```

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
4
30 28 27 24
23 27 28 29
2
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

Boilerplate/Skeleton Code:

```
#include <stdio.h>

int marriage(int boys[], int girls[], int n)
{
        // Write your code here
}

// Driver code
int main()
{
    int n;
    scanf("%d", &n);
    int boys[n], girls[n], res;
    for (int i = 0; i < n; i++)
        scanf("%d", &boys[i]);
    for (int i = 0; i < n; i++)
        scanf("%d", &girls[i]);
    res = marriage(boys, girls, n);
    printf("%d",res);
    return 0;
}
```

```
#include <stdio.h>

void sorting(int num[],int n){
    int a,i,j;
for (i = 0; i < n; ++i){
   for (j = i + 1; j < n; ++j){
      if (num[i] > num[j]){
         a = num[i];
         num[i] = num[j];
         num[j] = a;
      }
   }
```

```
}
}
int marriage(int boys[], int girls[], int n)

{
    int count=0;
    sorting(boys,n);
    sorting(girls,n);
    for(int i=0;i<n;i++){
        if(boys[i]==girls[i]){
            count++;
        }


  }
return count;
// Write your code here

}
int main()

{

int n;

scanf("%d", &n);

int boys[n], girls[n], res;

for (int i = 0; i < n; i++)

scanf("%d", &boys[i]);

for (int i = 0; i < n; i++)

scanf("%d", &girls[i]);

res = marriage(boys, girls, n);

printf("%d",res);

return 0;

}
```

Test cases:

1.

2
25 25
25 25

Output -
2

```
PS C:\Users\Praka\OneDrive\Documents\DAA> gcc weeek4_1.c
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
2
25 25
25 25
2
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

2.
1
27
27

Output -
1

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
1
27
27
1
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

3.
3
27 28 29
24 25 26

Output -
0

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
3
27 28 29
24 25 26
0
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

Q2. Sort the roll numbers of n students using bubble sort given by the array rollnumber[] using lexicographic order, the order of duplicates have to be maintained. Print the sorted order.

Sample Input 1:
4
3 3 1 2
Expected Output 1:
1 2 3 3

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
4
3 3 1 2
1 2 3 3
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

Boilerplate/Skeleton Code:

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


void solution (int rollnumber[],int n){
        //Write your solution here

}

int main()
{
    // Driver code
        int n;
        scanf("%d",&n);
        int arr[n];
        for(int i=0;i<n;i++){

                scanf("%d",&arr[i]);

        }

    solution(arr,n);

    return 0;
```

}

```c
#include <stdio.h>

#include <stdlib.h>

void solution(int rollnumber[],int n){
    int a,i,j;
for (i = 0; i < n; ++i){
   for (j = i + 1; j < n; ++j){
      if (rollnumber[i] > rollnumber[j]){
         a = rollnumber[i];
         rollnumber[i] =rollnumber[j];
         rollnumber[j] = a;
      }
   }
}
for(int i=0;i<n;i++){
    printf("%d ",rollnumber[i]);
}
}

int main()

{

// Driver code

int n;

scanf("%d",&n);

int arr[n];

for(int i=0;i<n;i++){

scanf("%d",&arr[i]);

}

solution(arr,n);

return 0;

}
```

Test cases:
1.
2
1 2
Output -
1 2

```
PS C:\Users\Praka\OneDrive\Documents\DAA> gcc week4_2.c
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
2
1 2
1 2
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

2.
3
2 1 3

Output -
1 2 3

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
3
2 1 3
1 2 3
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

3.
4
4 4 2 3
Output -
2 3 4 4

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
4
4 4 2 3
2 3 4 4
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

Q3. Timur initially had a binary string s (possibly of length 0). He performed the following operation several (possibly zero) times:
Add 0 to one end of the string and 1 to the other end of the string. For example, starting from the string 1011, you can obtain either 010111 or

110110. You are given Timur's final string. What is the length of the shortest possible string he could have started with?
Note: A binary string is a string (possibly an empty string) whose characters are either 0 or 1.

Driver Code:

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

int min_binary_string(char *s) {
    // Enter your code here
}

int main() {
    char s[100];
    scanf("%s", s);
    printf("%d", min_binary_string(s));
    return 0;
}
```

```
#include <stdio.h>

#include <string.h>

int min_binary_string(char *s,int n) {
int count=0;
int total;
// Enter your code here
for(int i=0;i<n;i++){
    if((s[i]=='0' && s[n-i-1]=='1') || (s[n-i-1]=='0' && s[i]=='1')){
       count=count+2;
    }

    else{
        break;
    }
}
total=n-count;
return total;

}

int main() {
int n;
char s[100];
scanf("%d",&n);
```

```
scanf("%s", s);

printf("%d", min_binary_string(s,n));

return 0;

}
```

Sample Input 1:
3 // size of input
100 // input string

Expected Output 1:
1

```
PS C:\Users\Praka\OneDrive\Documents\DAA> gcc week4_3.c
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
3
100
1
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

Test cases:

1. 4
   0111
   Output: 2

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
4
0111
2
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

2. 5
   10101
   Output: 5

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
5
10101
5
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

3. 7
   1010110
   Output: 3

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
7
1010110
3
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

4. 10
   1011011010

Output: 4

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
10
1011011010
4
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

Q4. Given a sequence of numbers from 1 to n, each permutation in the sequence that we generate should differ from the previous permutation by swapping just two adjacent elements of the sequence. Implement the same using Johnson Trotter algorithm.

(Consider input for n=3,4 and 5)

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

#define MAX_N 5

int p[MAX_N + 1];  // The permutation
int pi[MAX_N + 1]; // The inverse permutation
int dir[MAX_N + 1]; // The direction of each element

void print_permutation(int n) {
    for (int i = 1; i <= n; i++) {
        printf("%d ", p[i]);
    }
    printf("\n");
}

int get_mobile(int n) {
    int mobile = 0, mobile_pos = 0;
    for (int i = 1; i <= n; i++) {
        int j = pi[i] + dir[i];
```

```
        if (j >= 1 && j <= n && p[j] < p[i]) {
            if (p[i] > mobile) {
                mobile = p[i];
                mobile_pos = i;
            }
        }
    }
    return mobile_pos;
}

void swap(int i, int j) {
    int temp = p[i];
    p[i] = p[j];
    p[j] = temp;

    pi[p[i]] = i;
    pi[p[j]] = j;
}

void reverse_directions(int n, int mobile) {
    for (int i = 1; i <= n; i++) {
        if (p[i] > mobile) {
            dir[i] = -dir[i];
        }
    }
}

void generate_permutations(int n) {
    for (int i = 1; i <= n; i++) {
        p[i] = i;
        pi[i] = i;
        dir[i] = -1;
    }

    print_permutation(n);

    int mobile = get_mobile(n);

    while (mobile != 0) {
        int j = pi[mobile] + dir[mobile];
        swap(pi[mobile], j);
        reverse_directions(n, mobile);

        print_permutation(n);

        mobile = get_mobile(n);
    }
}
```

```c
int main() {
    printf("Permutations of length 3:\n");
    generate_permutations(3);

    printf("Permutations of length 4:\n");
    generate_permutations(4);

    printf("Permutations of length 5:\n");
    generate_permutations(5);

    return 0;
}
```

```
PS C:\Users\Praka\OneDrive\Documents\DAA> gcc week4_4.c
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
Permutations of length 3:
1 2 3
1 3 2
3 1 2
Permutations of length 4:
1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
Permutations of length 5:
1 2 3 4 5
1 2 3 5 4
1 2 5 3 4
1 5 2 3 4
5 1 2 3 4
PS C:\Users\Praka\OneDrive\Documents\DAA>
```