13/06/24

LAB-7.

① Johnson Trotter Program.

```
int flag=0;
int swap (int *a,int *b){
    int t =*a;
    *a = *b;
    *b = t;
}
int search (int arr[], int num, int mobile){
    int g;
    for (g=0; g<num; g++){
        if(arr[g] == mobile)
            return g+1;
        else
            flag++;
    }
    return -1;
}
int find_Mobile (int arr[], int d[], int num){
    int mobile = 0;
    int mobile_p=0;
    int i;
    for (i=0;i<num;i++){
        if((d[arr[i]-1] ==0) && i!=0){
            if (arr[i]>arr[i-1] && arr[i] > mobile_p){
                mobile = arr[i];
                mabile_p = mobile;
            }
            else{
                flag++;
            }
        }
        else if((d[arr[i]-1] ==1) & i!=num-1){
```

```
            if (arr[i]>arr[i+1] && arr[i]>mobile_p){
                mobile=arr[i];
                mobile_p=mobile;
            }
            else{
                flag++;
            }
        }
        else{
            flag++;
        }
    }
    if ((mobile_p==0)&&(mobile==0))
        return 0;
    else
        return mobile;
}
void permutations(int arr[], int d[],int num){
    int i;
    int mobile=findMobile(arr,d,num);
    int pos =search(arr,num,mobile);
    if(d[arr[pos-1]-1]==0)
        swap(&arr[pos-1],&arr[pos-2]);
    else
        swap(&arr[pos-1],&arr[pos]);
        for(int i=0;i<num;i++){
            if(arr[i]>mobile){
                if(d[arr[i]-1]==0)
                    d[arr[i]-1]=1;
                else
                    d[arr[i]-1]=0;
            }
        }
        for(i=0;i<num;i++)
            printf("%d",arr[i]);
}
```

```
int factorial (int k){
    int f=1;
    int i=0;
    for (i=1; i<k+1; i++){
        f = f*i;
    }
    return f;
}
```

O/P:-

→ Johnson Trotter Algorithm
Enter the no.
3.
total permutations = 6
All possible permutations are:

1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3

→ Johnson Trotter Algorithm
Enter the no.
4
Total permutations: 24.
All possible permutations are:

1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 1 4 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4
3 4 1 2

② Substring Matching / Pattern Matching.

```
int substringMatch(char *text, char *pattern){
    int textLength = strlen(text);
    int patternLength = strlen(pattern);
    for(int i=0; i<=textLength - patternLength; i++){
        int j;
        for(j=0; j<patternLength; j++){
            if(text[i+j] != pattern[j])
                break;
        }
        if(j == patternLength)
            return i;
    }
    return -1;
}
int main(){
    char text[100], pattern[100];
    printf("Enter the main text:");
    scanf("%s", text);
    printf("Enter the pattern to search:");
    scanf("%s", pattern);
    int index = substringMatch(text, pattern);
    if(index != -1)
        printf("Substring found at index: %d\n", index);
    else
        printf("Substring not found.\n");
    return 0;
}
```

O/P:

```
Enter the main text: hello_brother
Enter the pattern to search: bro
Substring found at index: 6.
```

③ Leetcode Problem -1985
(Find Kth largest integer in array)

```
int cmp(const void *a, const void *b){
    const char* str1 = *(const char**) a;
    const char *str2 = *(const char **) b;
    if (strlen (str1) == strlen (str2)){
        return strcmp(str1, str2);
    }
    return strlen(str1) - strlen(str2);
}
char * kthLargestNumber (char ** nums, int numsSize, int k){
qsort (nums, numsSize, sizeof (char *), cmp);
return nums [numsSize - k];
}.
```

O/P:-

Case 1: nums = ["3","6","7","10"]
k = 4
Output = "3"
Expected = "3".

Case 2: nums = ["2","12","21","1].
k = 3
Output = "2"
Expected = "2"

Case 3: nums = ["0","0"]
k = 2
Output = "0"
Expected = "0".

13/6/24