Johnson Trotter

```
int p[MAXN];
int dir[MAXN];

void printPermutation (int n)
    for (int i=0; i<n; i++)
        pf ("%d", p[i]);
    }
    pf("\n");
}

void initialize (int n) {
    for (int i=0; i<n; i++) {
        p[i] = i+1;
        dir[i] = -1;
    }
}

int findLargestMobile (int n)
    int mobileIndex = -1;
    int mobile = 0;
    for (int i=0; i<n; i++) {
        if ((dir[i] == -1 && i>0 && p[i] > p[i-1] && p[i] > mobile) || (dir[i] == 1 && i<n-1
        && p[i] > p[i+1] && p[i] > mobile)) {
            mobile = p[i];
            mobileIndex = i;
        }
    }
    return mobileIndex;
}
```

```
void swapAdjacent(int n, int mobileIndex){
    int temp = p[mobileIndex];
    int tempDir = dir[mobileIndex];
    p[mobileIndex] = p[mobileIndex + tempDir];
    p[mobileIndex + tempDir] = temp;

    dir[mobileIndex] = dir[mobileIndex + tempDir];
    dir[mobileIndex + tempDir] = tempDir;
}

void reverseDirections(int n, int x){
    for(int i=0; i<n; i++){
        if(p[i] > x){
            dir[i] *= -1;
        }
    }
}

void generatePermutations(int n){
    initialize(n);
    printPermutation(n);
    int mobileIndex = findLargestMobile(n);
    while(mobileIndex != -1){
        swapAdjacent(n, mobileIndex);
        reverseDirection(n, n-p[mobileIndex]);
        printPermutation(n);
        mobileIndex = findLargestMobile(n);
    }
}
```

```
int main() {
    int n;
    printf("Enter the value of n (3 or 4): ");
    scanf("%d", &n);
    if (n != 3 && n != 4) {
        printf("Invalid input! Please enter 3 or 4.\n");
        return 1;
    }
    generatePermutations(n);
    return 0;
}
```

Output:

Enter the no (3 or 4): 3

Total permutations =

123	312	231
132	321	213.

13/09/24 Sub String & Pattern Matching.

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

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], patter[100];
    pf("Enter the main text:");
     sf("%s", text);
    pf("Enter the pattern to search:");
    sf("%s", pattern);
    int index = substringMatch(text, pattern);
    if (index != -1)
        pf("substring found at index: %d\n",
          index);

    else
        pf("substring not found.\n");

    return 0;
}
```

Output:

Enter the main text : sujju
Enter the substring to search : ju
Substring found at index : 3

Enter the main text : hello
Enter the substring to search : suj
Substring ~~Pattern~~ not found

Date __/__/__
Page

Leet Code:

```
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];
}
```

Output:

Case 1: nums = [3, 6, 7, 10]
k = 4
o/p = Expected = 3

Case 2: nums = [2, 12, 21, 1]
k = 3
o/p = Expected = 2

Case 3: num = [0, 0]
k = 2
o/p = Eexpected = 0.