שאלה 1

<u>סעיף א'</u>

התוכנית מבצעת חיפוש במערך <u>ממוין</u>.

הסבר יותר מפורט:
החיפוש דומה לחיפוש בינארי, אך שיטת החלוקה של המערך שונה. במקום כל פעם לבצע חלוקה של המערך לשני חלקים שווים, מבצעים pivoting ע"פ $r=|a[0]| \bmod n$.
(שארית החלוקה של הערך המוחלט של a[0] ב- n), במילים אחרות, מחלקים את המערך לשני חלקים בצורה הבאה:

$$a[0], a[1], \dots , a[r] \;\; ; \;\; a[r+1], a[r+2] , \dots , a[n]$$

שימו לב כי r אכן מייצג אינדקס חוקי בערך $(0 \le r \le n-1)$.

הערה:
אם נחליף את שורת הקוד "r=abs(a[0]) % n; " בשורה "r = n / 2;" נקבל חיפוש בינארי רגיל.

<u>סיבוכיות:</u>

נסתכל על קלט x=2 ו $a=\{0,0,0, \dots , 0\}$. במקרה זה r=0 בכל איטרציה מה שאומר שבכל פעם הפונקציה תיקרא רקורסיבית עם פרמטר n-1. ולכן על קלט זה עומק הרקורסיה יהיה n. כיוון שעומק הרקורסיה **אינו** יכול להיות יותר גדול מ-n (בכל איטרציה הn קטן) נקבל שב worst case עומק הרקורסיה הוא בדיוק $\Theta(n)$. בנוסף, שימו לב שהרקורסיה במקרה זה היא ליניארית.

בגוף הפונקציה אין לולאות ו/או הקצאות דינמיות נקבל שסיבוכיות הזמן והמקום הנוסף של **איטרציה בודדת** היא $\Theta(1)$. מזה נובע שהסיבוכיות הזמן והמקום הנוסף הכוללת של הפונקציה היא $\Theta(n)$.

הערה 1:
למרות שהפונקציה מאוד "דומה" באופייה לחיפוש בינארי, סיבוכיות הworst case של הפונקציה שונה מזו של חיפוש בינארי.

הערה 2:
על מנת להראות שהסיבוכיות היא $\Theta(n)$, לא מספיק להגיד "עומק הרקורסיה חסום ע"י n..." כי גם במקרה של חיפוש בינארי רגיל עומק הרקורסים חסום ע"י n, מאידך הסיבוכיות במקרה זה היא $\Theta(\log n)$.

<u>סעיף ב'</u>

סיבוכיות זמן: $\Theta(\log n)$
סיבוכיות מקום נוסף: $\Theta(1)$

<u>מקום נוסף:</u>
אין הקצאות דימניות, מערכי עזר בגדלים לא קבועים, ו/או קריאות רקורסיביות.

<u>זמן:</u>
כל הפקודות מתבצעות ב - $\Theta(1)$ פרט ללולאת הwhile. בכל איטרציה של הלולאה r אומנם יכול לקבל ערך שונה, אך מתקיים $2 \le r \le 11$. כיוון שבכל איטרציה n מחולק ב-r, מספר האיטרציות לולאה הwhile הינו בין $\log_2 n$ ל $\log_{11} n$, כלומר $\Theta(\log n)$. שאר הפקודות בפונקציה מתבצעות ב-$\Theta(1)$ ואין קריאות רקורסיביות, אז סיבוכיות הזמן הכולל של הפונקציה היא $\Theta(\log n)$.

הערה:
בדומה לסעיף א' לא מספיק להגיד " $2 \le r$ אז מספר האיטרציות במקרה הגרוע הוא $\log_2 n$". למעשה מה שטיעון זה מראה הוא "חסם עליון", (כלומר שסיבוכיות הזמן היא $O(\log n)$) אך זה לא מבטיח שהחסם אכן הדוק כפי שזה נדרש בהדרת המדד Θ.

לצורך המחשה נסתכל על הפונקציה f1:

```
int f1(int n)
{

int r=0;

        while (n>=4)
        {
        r = n / 2;
        n /= r;
        }

return r;

}
```

שימו לב שגם במקרה זה $2 \le r$ (כי $4 \le n$) אך סיבוכיות זמן הריצה במקרה זה היא $\Theta(1)$! (לוודא!)

שאלה 2

סעיף א'

```
int is_identical(int a[N][N], int x1, int y1, int x2, int y2, int k){
    int i, j, similar = 0;
    // check if they're the same without rotation
    for (i = 0; i< k; ++i)
        for(j = 0; j < k; ++j)
            if (a[x1+i][y1+j] == a[x2+i][y2+j])
                similar++;
    if (similar == k*k)
       return 0;
    // check if they're the same under 90-degree rotation
    similar = 0;
    for (i = 0; i< k; ++i)
        for(j = 0; j < k; ++j)
            if (a[x1+i][y1+j] == a[x2+j][y2+k-1-i])
                similar++;
    if (similar == k*k)
       return 0;
    // check if they're the same under 180-degree rotation
    similar = 0;
    for (i = 0; i< k; ++i)
        for(j = 0; j < k; ++j)
            if (a[x1+i][y1+j] == a[x2+k-1-i][y2+k-1-j])
                similar++;
    if (similar == k*k)
       return 0;
  // check if they're the same under 270-degree rotation
    similar = 0;
    for (i = 0; i< k; ++i)
        for(j = 0; j < k; ++j)
            if (a[x1+i][y1+j] == a[x2+k-1-j][y2+i])
                similar++;
    if (similar == k*k)
       return 0;
    // no match found
    return 1;
}
```

סעיף ב'

```
int find_identical(int a[N][N], int k){
 int x1, y1, x2, y2;
 if (k<=0) return 1;
 for(x1 = 0; x1 <= N-k; ++x1)
     for(y1 = 0; y1 <= N-k; ++y1)
         for(x2 = 0; x2 <= N-k; ++x2)
             for(y2 = 0; y2 <= N-k; ++y2)
               if (!(x1==x2 && y1 == y2) && !is_identical(a,x1,y1,x2,y2,k)){
                 return 0;
               }
 return 1;
}
```

סעיף ג'

```
int max_identical(int a[N][N]){
 int high = N-1, low = 1, mid, last_good_k = 0;
 while (high >= low){
       mid = (high+low)/2;
       if (find_identical(a,mid)) // no rotation-equal sub-matrices
           high = mid - 1;
       else
           if (mid > last_good_k)
               last_good_k = mid;  // the highest k found so far
           low = mid + 1;
 }
 return last_good_k;
}
```

שאלה 3

סעיף א'

```
int indexPermutation(int a[], int n)
{
        int *p;
        int i;
        int isIndex = 0;

        p = (int *) malloc ( sizeof(int) * n);
        if(p==NULL)
         {
              printf("memory allocation error, exiting...");
              exit(1);
          }

        for (i=0;i<n;i++)
         p[i] = 0;

        for (i=0; i<n; i++)
         {
                if (a[i] < 0 || a[i] > n-1)
                {
                        isIndex = 1;
                        break;
                }
                p[a[i]] ++;
                if (p[a[i]] > 1)
                {
                        isIndex = 1;
                        break;
                }
         }

         free(p);
         return isIndex;
}
```

סעיף ב'

```
void d_swap(int array_1[], int array_2[], int x, int y)
{
     int temp1 = array_1[x];
     int temp2 = array_2[x];
     array_1[x] = array_1[y];
     array_2[x] = array_2[y];
     array_1[y] = temp1;
     array_2[y] = temp2;
 }

int permutation(int a[], int index[], int n)
{

    int count = 0;
    int loc = 0;
    while(count < n)
    {
      d_swap(a,index,loc,index[loc]);
      if (loc == index[loc])
          loc++;
      count++;
    }
  return 1;
}
```

```
void word_permutation (char *s, int index[])
{
    int *word_start=NULL,i,j,word_num=0,idx;

    //counting the number of spaces
    for(i=0;*(s+i);++i) {
        if (*(s+i)==' ') {
                ++word_num;
                }
    }
    ++word_num;
                // the number of words is bigger by one than the number of spaces
    word_start = (int*)malloc(word_num*sizeof(int));
    if(word_start==NULL) {
                    printf("memory allocation error, exiting...");
                    exit(1);
                    }
    //finding the start of each word;
    word_start[0]=0;
    j=1;
    for(i=0;*(s+i);++i) {
        if (*(s+i)==' ') {
                word_start[j]=i+1;
                }
    }

    permutaion(word_start,index,word_num);

    //printing
    for(i=0;i<word_num;++i) {
                    idx=word_start[i];
                    while ((s[idx])&&(s[idx]!=' ')) {
                        printf("%c",s[idx]);
                    }
                    if (i<word_num-1) {
                                printf(" ");
                    }
    }
    free(word_start);
    return;
}
```

שאלה 4

```
int find_path(int a[N][N], int s, int t, int k, int path[]) {
    if (s == t) {
        path[0] = t;
        return 1;
    }
    int i, res;
    for (i=0; i < N; i++) {
        if (a[s][i] > 0 && a[s][i] <= k) {
            res = find_path(a, i, t, k - a[s][i], path+1);
            if (res > 0) {
                path[0] = s;
                return res+1;
            }
        }
    }
    return -1;

}
```