Problem 8:

```
void initMatrix ( int ____*A[]____ , int r, int c )
{
   int i, j;
   for (i = 0; i < r; ++i) {  /* i is the row index */
      /* Allocate the exact amount of memory for allocating each row */

      A[i] = ____(int *)malloc(c * sizeof(int));____ ;
      for (j=0; j<c; ++j)   /* j is the column index */

         scanf("%d, ____&A[i][j] or A[i] + j or *(A + i) + j____ );   /* Read the (i,j)-th element of A */
   }
}



void printDiagonals ( int ____*A[]____ , int r, int c )
{
   int d, i, j, row_start, col_start;

   /* Initialize the start indices for printing Diagonal 1 */

   row_start = ____0____ ; col_start = ____c - 1____ ;

   for ( d = 1; d <= ____r + c - 1____ ; ++d ) {   /* d is the diagonal number */

      printf("Diagonal %d:", d);

      i = row_start; j = col_start;  /* Indices of the first position on the diagonal */

      while ( ____(i < r) && (j < c)____ ) {

         printf(" %d", A[i][j]);

         /* Update both i and j to the next position on the diagonal */

         ____++i; ++j;____
      }
      printf("\n");  /* Printing the d-th diagonal is complete */

      /* Prepare for printing the next diagonal */

      ____if (col_start > 0) --col_start; else ++row_start;____
   }
}
```

Problem 9:

```
int jumpsearch ( int A[], int n, int key )
{
   int L, R, jump;
   /* Initialize the search interval I = [L, R] */

   L = ____0____ ; R = ____n - 1____ ;
   /* Repeat until I shrinks to a single-element interval */

   while ( ____L < R____ ) {
      jump = 1;      /* jump is the next search index relative to L */
      while (1) {    /* Repeat until the correct sub-interval is located */

         if ( ____L + jump > R (L + jump >= R is also OK)____ ) break; /* Next search index is beyond R */

         /* Compare key with the array element indicated by jump */

         if ( ____key < A[L + jump]____ ) {      /* R is determined now */

            R = ____L + jump - 1____ ; break;
         } else {                                /* Continue the search */

            L = ____L + jump____ ;

            jump = ____jump * 2____ ;
         }
      }
   }
   /* Make a final comparison, and return L or -1 */

   ____return (key == A[L]) ? L : -1;____

}
```

Problem 10:

```
void digisort ( int A[], int n )
{
   int max, maxdigitcnt, digitpos, tenpower, d, i, j;
   int B[10][MAX_SIZE], C[10];
   max = findmax(A,n);             /* Find the maximum element in A[] */
   maxdigitcnt = digitcount(max);  /* Find the number of digits in max */
   /* The following loop runs on digitpos from least to most significant positions */
   tenpower = 1; /* This will always store 10-to-the-power digitpos */

   for ( digitpos = 0; digitpos < maxdigitcnt; ++digitpos ) {

      for (d = 0; d < 10; ++d) C[d] = 0;   /* Initialize all counts to 0 */
      for (i = 0; i < n; ++i) {

         d = (A[i] / tenpower) % 10 ;  /* Compute the digitpos-th digit of A[i] */
         /* Append A[i] to the d-th list B[d] */

         B[d][C[d]] = A[i];

         ++C[d];
      }
      /* Copy back the digit-wise sub-lists from B to A. Index j is for writing to A. */
      j = 0;
      for ( d = 0; d < 10; ++d ) { /* Loop on d */

         for ( i = 0; i < C[d]; ++i ) { /* Loop on i */

            A[j] = B[d][i]; ++j;
      }
      tenpower = tenpower * 10 ;  /* Update tenpower for next iteration */
   }
}
```