INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR	
	Stamp / Signature of the Invigilator

EXAMINATION (End Semester)	**SEMESTER (Spring 2023-2024)**

Roll Number									**Section**		**Name**		
Subject Number	C	S	1	0	0	0	3	**Subject Name**	*PROGRAMMING AND DATA STRUCTURES*				
Department / Center of the Student			**Additional sheets**										

Important Instructions and Guidelines for Students

1. You must occupy your seat as per the Examination Schedule/Sitting Plan.
2. Do not keep mobile phones or any similar electronic gadgets with you even in the switched off mode.
3. Loose papers, class notes, books or any such materials must not be in your possession, even if they are irrelevant to the subject you are taking examination.
4. Data book, codes, graph papers, relevant standard tables/charts or any other materials are allowed only when instructed by the paper-setter.
5. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, exchange of these items or any other papers (including question papers) is not permitted.
6. Write on both sides of the answer script and do not tear off any page. **Use last page(s) of the answer script for rough work.** Report to the invigilator if the answer script has torn or distorted page(s).
7. It is your responsibility to ensure that you have signed the Attendance Sheet. Keep your Admit Card/Identity Card on the desk for checking by the invigilator.
8. You may leave the examination hall for wash room or for drinking water for a very short period. Record your absence from the Examination Hall in the register provided. Smoking and the consumption of any kind of beverages are strictly prohibited inside the Examination Hall.
9. Do not leave the Examination Hall without submitting your answer script to the invigilator. **In any case, you are not allowed to take away the answer script with you.** After the completion of the examination, do not leave the seat until the invigilators collect all the answer scripts.
10. During the examination, either inside or outside the Examination Hall, gathering information from any kind of sources or exchanging information with others or any such attempt will be treated as **'unfair means'**. Do not adopt unfair means and do not indulge in unseemly behavior.

Violation of any of the above instructions may lead to severe punishment.

Signature of the Student

To be filled in by the examiner

Question Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
Marks Obtained											

Marks obtained (in words)	**Signature of the Examiner**	**Signature of the Scrutineer**

CS10003 : Programming and Data Structures (Spring 2023-2024)

April 2024 **End-Semester Test** Maximum Marks: 100

Instruction to students

- Write your answers in the question paper itself. Be brief and precise. Answer all questions.
- Write the answers only in the respective spaces provided.
- The questions appear on the next *nine* pages.
- Please ask for supplementary sheets during the test if you need more rough space.
- All the questions use the programming language C.
- Not all blanks carry equal marks (unless otherwise stated). In those cases, evaluation will depend on overall correctness.
- In the questions involving fill in the blanks inside the code snippets, *each blank can be filled with at most one expression or statement as applicable.*

Do not write anything on this page.

Questions start from the next page.

Q1. If the following C programs terminate, then write the output. When no output can be generated, write `NONE`. In case, the program contains an error, write `ERROR` with the `LINE NUMBER`(s). Otherwise, say that the program does not terminate (write `NO TERMINATION`). **(2×10)**

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

// structure definition for queue
typedef struct _queue {
  // array storing queue elements
  int items[SIZE];
  /* index of front and rear
     elements of queue */
  int front, rear;
} Queue;

/* function to append an element
   at the end of a queue:
   returns nothing */
void enqueue(Queue *q, int value);

/* function to remove an element
   from the beginning of queue:
   returns the removed element,
   or -1 if the queue is empty */
int dequeue ( Queue *q );

int main() {
  Queue q;
  q.front = q.rear = 0;
  enqueue(&q,5); enqueue(&q,10);
  printf ("%d ", dequeue (&q));
  enqueue(&q,15); enqueue(&q,20);
  printf ("%d ", dequeue (&q));
  return 0;
}
```

Answer: `5 10`

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 3

// structure definition for stack
typedef struct _stack {
  // array storing stack elements
  int items[SIZE];
  // index of top element of stack
  int top;
} Stack;

/* function to push an element
   onto a stack:
   returns nothing */
void push ( Stack *s, int value );

/* function to pop an element
   from the stack:
   returns the popped element,
   or -1 if the stack is empty */
int pop ( Stack *s );

int main () {
  Stack s;
  s.top = -1;
  push(&s, s.top); push(&s, s.top);
  printf("%d ", pop(&s));
  printf("%d ", pop(&s));
  printf("%d ", pop(&s));
  return 0;
}
```

Answer: `0 -1 -1`

```c
#include <stdio.h>
typedef struct Point2D {
  int x;
  int y;
} POINT;
int main () {
  POINT p = {3,4}, *ptr = &p;
  printf("%d %d",ptr->x,(*ptr).y);
  return 0;
}
```

Answer: `3 4`

```c
#include <stdio.h>
int main () {
  int *px, *py;
  int a = 10, b = 20;
  px = &a;
  *px = b;
  *py = a;
  printf ("%d %d, ", a, b);
  printf ("%d %d", *px, *py);
  return 0;
}
```

Answer: `ERROR in Line 7: *py = a;`
`(Segmentation Fault)`

```
#include <stdio.h>
void foo ( int *p, int *q,
           int x, int y ) {
  p = &y;  *q = 30;
}
int main () {
  int *px, *py;
  int a = 10, b = 20;
  px = &a;  py = &b;
  foo (px, py, a, b);
  printf ("%d,%d", *px, *py);
  return 0;
}
```

Answer: 10,30

```
#include <stdio.h>
int fun ( char *s ) {
  if ( *s == '\0' ) return 0;
  if ( *s != *(s+1) ) {
    printf ("%c", *s);
    return 1 + fun (s+1);
  }
  return fun (s+1);
}
int main () {
  printf(" %d",fun("Yiipppeeee"));
  return 0;
}
```

Answer: Yipe 4

```
#include <stdio.h>
int count = 0;
int pds ( int n ) {
  int m = n - 1;
  ++count;
  while(m>=0) { pds(m); m=m-2; }
  return count;
}
int main () {
  printf("%d", pds(5));
  return 0;
}
```

Answer: 13

```
#include <stdio.h>
int main () {
  typedef struct tag {
    char str[10];
    int a;
  } har;
  har h2 = { "IHelp", 10 };
  har h1 = h2;
  h1.str[1] = 'h';
  printf("%s %d", h1.str, h1.a);
  return 0;
}
```

Answer: Ihelp 10

```
#include<stdio.h>
double recurse (int A[], int N) {
  if (N == 1)
    return((double)A[N-1]);
  else
    return((double)(recurse(A,N-1)
            *(N-1)+A[N-1]) / N);
}
int main () {
  double result = 0;
  int A[] = { 1, 2, 3, 4 };
  int N = sizeof(A)/sizeof(A[0]);
  printf ("%.2lf", recurse(A,N));
  return 0;
}
```

Answer: 2.50

```
#include <stdio.h>
typedef struct _str {
  int mem1, mem2;
  struct _str* next;
} STR;
int main () {
  STR var1 = { 1, 2, NULL };
  STR var2 = { 10, 20, NULL };
  var1.next = &var2;
  STR *ptr1 = &var1;
  printf ("%d %d",
           ptr1->next->mem1,
           ptr1->next->mem2);
  return 0;
}
```

Answer: 10 20

Q2. The following C function takes a string (pointer to character) as argument, removes all duplicate adjacent characters from the string using a *stack* (`st[]` with `top` index) and returns the shortened string in reverse. For example, if the input string is "azxxzy", then the output will be "ya" because the removal of "xx" modifies the string to "azzy", and then the removal of "zz" modifies the string to "ay", and finally "ay" does not containing duplicates, so the output string became "ya". Similarly, if the input string is given as "aaccdd" or "abccba", then the output will be an *empty string* in both cases (and the function returns the phrase "EMPTY"). Fill in the blanks in the following code snippet (*stack operations should be written explicitly inside the code*) to achieve the desired result. **(5)**

```
char *shortenString ( char *str ) {
  char st[100];   int top = -1, i, j;
  for ( i = 0; str[i] != '\0'; ++i ) {
      if ( (top == -1) || ( ____________str[i] != st[top]____________ ) )     (1)
          st[++top] = __________str[i]__________ ; /* push here */           (1)
      else __________--top__________ ; /* pop here */                         (1)
  }    /* check for emptiness below */
  if ( __________top == -1__________ )   return "EMPTY";                      (1)
  else {
      char* shortStr = (char*) malloc ((top+2)*sizeof(char));
      for ( j=0; top != -1; j++ )
          shortStr[j] = __________st[top--]__________ ; /* pop here */        (1)
      shortStr[j] = '\0';
      return shortStr;
  }
}
```

Q3. The following C function takes as its arguments a pointer to the head node of a linked list (the node structure is also defined below) and an integer `n` to indicate the position (node distance) from last node and then finds/prints the data of the node which is n positions (nodes) before the last node ($n = 0$ indicates the last node). Fill in the blanks in the following code snippet to achieve the desired result. **(5)**

```
struct node {
  int num;
  struct node *next;
};
void nthnode ( struct node *head, int n ) {
  struct node *p, *q;   int i;   q = p = head;
  for ( i = 0; (i < n) && ( __________q != NULL__________ ); i++ )          (1)
      q = __________q->next__________ ;                                      (1)
  if (q == NULL)   printf ("List has Less Elements than the Entered n.") ;
  else {
      while ( __________q->next__________ != NULL ) {                        (1)
          __________q__________ = q->next;                                   (1)
          p = __________p->next__________ ;                                  (1)
      }
      printf ("%d is Positioned %d Nodes before Last Node.", p->num, n) ;
  }
}
```

Q4. **(a)** Any rational number (fractions of the form $\frac{x}{y}$, where $y \neq 0$) can be normalized to its lowest form by dividing both its numerator and the denominator with their greatest common divisor (GCD). Complete the following C function, which takes as its argument two integers x and y (assume that, $y \neq 0$) presenting the fraction $\frac{\pm \mathtt{x}}{\pm \mathtt{y}}$, and computes the lowest form of that fraction. The function also normalizes the sign of the overall fraction from the individual signs ($\pm$) of x and y, finally printing it as $\pm\frac{\mathtt{a}}{\mathtt{b}}$ (where, $\frac{\mathtt{a}}{\mathtt{b}}$ becomes the normalized version of $\frac{\mathtt{x}}{\mathtt{y}}$). **(7)**

```
void normRat ( int x, int y ) {
  int sign = 1, gcd, temp, rem;
  if ( x < 0 ) { sign = ________-1 (or, -sign)________ ;   x = - x ; }     (0.5)
  if ( y < 0 ) { sign = ________sign * (-1)________ ;   y = - y ; }       (0.5)
  if ( ____________x >= y____________ ) { gcd = x; temp = y; }            (1)
  else   { gcd = y; temp = x; }
  while ( __________temp > 0 (or, temp != 0)__________ ) {                 (1)
      rem = ____________gcd % temp____________ ;                           (1)
      gcd = ____________temp____________ ;                                 (1)
      temp = ____________rem____________ ;                                 (1)
  }
  ( sign > 0 ) ? printf ("+") : printf ("-") ;
  printf (" %d / %d", ______x / gcd______ , ______y / gcd______ );         (1)
}
```

(b) Every positive rational number (such as $\frac{x}{y}$, where $x, y > 0$) has an essentially unique finite continued fraction representation, which can be expressed mathematically as:

$$\frac{x}{y} = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cfrac{1}{\ddots \, \frac{1}{q_n}}}}} \equiv \left[\, q_0 \; q_1 \; q_2 \; q_3 \; \cdots \; q_n \,\right] \leftarrow (\mathtt{notation})$$

Such a representation for any rational number follows directly the basic division (with quotient and remainder) principle, which is shown below:

$$\frac{x}{y} = q_0 + \frac{r_0}{y} = q_0 + \frac{1}{\frac{y}{r_0}} = q_0 + \cfrac{1}{q_1 + \frac{r_1}{r_0}} = q_0 + \cfrac{1}{q_1 + \cfrac{1}{\frac{r_0}{r_1}}} = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \frac{r_2}{r_1}}} = \cdots = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{\ddots \, \frac{1}{q_n}}}}$$

Note that, each division above of the form $\left[\frac{x}{y}, \frac{y}{r_0}, \frac{r_0}{r_1}, \frac{r_1}{r_2}, \cdots, \frac{r_{n-3}}{r_{n-2}}, \frac{r_{n-2}}{r_{n-1}}\right]$ produces the quotients $\left[q_0, q_1, q_2, q_3, \cdots, q_{n-1}, q_n\right]$, and the remainders $\left[r_0, r_1, r_2, r_3, \cdots, r_{n-1}, 0\right]$, respectively. Here, the final remainder, i.e. $r_n = 0$, since such a representation of any rational number is *finite*.

Complete the following C function, which takes as its argument two positive integers x and y, computes the finite continued fraction representation for $\frac{\mathtt{x}}{\mathtt{y}}$, and prints the representation obtained in the notational form, $[\, \mathtt{q}_0 \; \mathtt{q}_1 \; \mathtt{q}_2 \; \mathtt{q}_3 \; \cdots \; \mathtt{q}_n \,]$, as shown above. **(5)**

```
void cfRat ( unsigned int x, unsigned int y ) {
  unsigned int z;
  printf ("[ ") ;
  do {  printf ("%u ", ____________x / y____________ );                    (1)
        z = y;
        y = __________x - y * ( x / y ) (or, x % y)__________ ;            (2)
        x = ____________________z____________________ ;                    (1)
     } while ( __________y > 0 (or, y != 0)__________ ) ;                   (1)
  printf ("]") ;
}
```

(c) The generalized (and infinite) form for the continued fractions are: $a_0 + \frac{a_1}{a_2 + \frac{a_3}{a_4 + \frac{a_5}{\ddots}}}$

There are interesting algebraic manipulations possible with this. In particular, you can determine square root of any positive number n using this concept. For example, to determine $x = \sqrt{n}$, we solve the equation $x^2 = n$ as:

$$x^2 - 1 = n - 1 \implies (x+1)(x-1) = n-1 \implies x - 1 = \frac{n-1}{1+x} \implies x = 1 + \frac{n-1}{1+x}$$

Now, the x in the RHS denominator can be further decomposed iteratively as continued fractions with the existing x from LHS itself – thereby stepwise expanding to,

$$x = \sqrt{n} = 1 + \frac{n-1}{1+x} = 1 + \frac{n-1}{2 + \frac{n-1}{1+x}} = \cdots = 1 + \frac{n-1}{2 + \frac{n-1}{2 + \frac{n-1}{\ddots}}}$$

Complete the following C function, which takes as its argument a positive integer `n`, and computes/returns its square root (i.e., $\sqrt{\texttt{n}}$) by iterative expansion of the continued function as shown above. Though the continued fraction form may be infinite, but your computation of square root must stop when the difference of the derived roots in successive steps fall below a threshold (say, $|\texttt{next} - \texttt{root}| \leq 10^{-6}$). *You may use functions from* `math.h` *library.* **(4)**

```
double sqRoot ( unsigned int n ) {
  double root, next = 1.0;
  do {
        root = next;
        next = 1.0+(n-1)/(1.0+root) (or, 1.0+(n-1)/(1.0+next)) ;      (2)

     } while (  fabs ( next - root ) > 0.000001  ) ;                   (1)
           (or, ((next-root)>0.000001) || ((root-next)>0.000001) )

  return  next (or, root)  ;                                           (1)
}
```

(d) Generalizing the procedure adopted in Part (b), the real roots of a quadratic equation (of the form $x^2 + bx + c = 0$, where $b^2 \geq 4c$) can also be determined. We may find one root as follows:

$$x^2 + bx + c = 0 \implies x = -b - \frac{c}{x} = -b - \frac{c}{-b - \frac{c}{x}} = \cdots = -b - \frac{c}{-b - \frac{c}{-b - \frac{c}{\ddots}}}$$

However, the other root can then be obtained by the known formula, i.e. $\frac{c}{x}$.

Complete the following C function, which takes as its argument two real values `b` and `c` presenting the quadratic equation of the form $x^2 + bx + c = 0$ (assume that, $b^2 \geq 4c$ and $b \neq 0$), and computes/prints both of its roots by the iterative expansion of the continued function as shown above. Though the continued fraction form may be infinite, but your computation of square root must stop when the difference of the derived roots in successive steps fall below a threshold (say, $|\texttt{next} - \texttt{root}| \leq 10^{-6}$). *You may use functions from* `math.h` *library.* **(4)**

```
void quadRoot ( double b, double c ) {
  double root, next = - b;
  do {
        root = next;
        next =  - b - c/root (or, - b - c/next)  ;                     (1)

     } while (  fabs ( next - root ) > 0.000001  ) ;                   (1)
           (or, ((next-root)>0.000001) || ((root-next)>0.000001) )

  printf ("Roots=(%lf,%lf)", next (or, root) , c/next (or, c/root) );  (2)
}                                    (or, -next-b; or, -root-b)
```

Q5. Fill in the blanks of the following C function which recursively arranges an array of integers such that all negative numbers and zeros precede all positive numbers. For example, given an array of a list of integers $\{1, 14, -3, 4, 5, 6, 0, -2\}$, a possible arrangement would be $\{-2, 0, -3, 4, 5, 6, 14, 1\}$. **(8)**

```
____void____ negpos ( int *A, int n ) {                          (1)
  int i = 0, j = n-1, temp;
  while ( A[j] >  0 ) ____j--____ ;                              (1)
  while ( A[i] <= 0 ) ____i++____ ;                              (1)
  if ( i < j ) {
      temp = A[i];
      A[i]= ____A[j]____ ;                                       (1)
      A[j]= ____temp____ ;                                       (1)
      negpos ( ____A+i+1____ , ____j-i-1____ ) ;                 (3)
  }
}
```

Q6. Consider the following C program and answer the following questions. *Assume that a variable of data type* `int` *required* 4 *bytes of storage space.*

```
#include <stdio.h>
int check ( int *p, int n ) {
  int i, temp, mid;
  for ( i=0; i<n-1; i++ )
      if ( p[i] > p[i+1] ) { temp = p[i]; p[i] = p[i+1]; p[i+1] = temp; }
  for ( i=n-1; i>0; i-- )
      if ( p[i] < p[i-1] ) { temp = p[i]; p[i] = p[i-1]; p[i-1] = temp; }
  mid = (p[0] + p[n-1]) / 2;
  return ( ( mid > p[n/2] ) ?  1 :  0 );
}
int main () {
  int A[10] = { 2, 4, -3, 6, -9, 1, -5, 2, 10, 7 },   val = check(A,10);
  printf ("%lu %d %d %d", sizeof(A)/sizeof(int), A[0], A[9], val);
  return 0;
}
```

(a) What is the output of the above C program? **(4)**

Answer: `10 -9 10 0`

(b) Suppose the address of `A`[3] is 1866954492 (in decimal). What would be the address of `A`[7]? **(2)**

Answer: 1866954508

(c) What value would be passed to the argument variable `p` in the function `check(.)` when it is called in the program from `main()`? **(2)**

Answer: 1866954480

(d) What would be the value of &`A` + 1? **(2)**

Answer: 1866954520

Q7. Consider a 2D grid (nr × nc) of characters and a single word (pattern), Your task is to find all occurrences of the given word in the grid in the diagonal direction North-East (Right and Up) as illustrated in Fig. 1. Fill in the blanks of the following C functions to achieve the desired result. **(9)**

Consider the following example of a 5×5 grid ($nr = nc = 5$),

```
g  a  a  g  g
g  a  g  a  g
g  a  g  a  a
g  a  g  a  g
g  a  g  a  g
```

- If the pattern is "gag", there are 4 matches (refer to Fig. 2).
- If the pattern is "agg", there is 1 match.
- If the pattern is "gaag" there is 0 match.

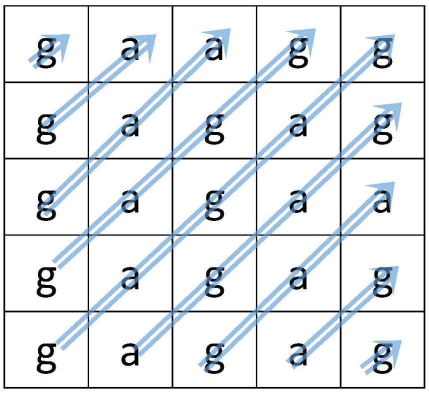
Fig 1: Diagonal directions to be considered

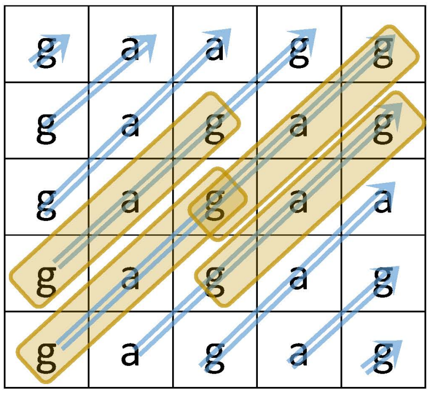
Fig 2:Matches of pattern "gag" in the grid

```
int matchRUDiag ( char cmat[][100], char *pattern,
                  int nr, int nc, int i, int j ) {
  if ( pattern[0] == ________'\0'________ ) /* base condition 1 */      (1)
      return ________1________ ;                                       (0.5)
  if ( (i>=nr) || (j>=nc) )                  /* base condition 2 */
      return ________0________ ;                                       (0.5)
  if ( pattern[0] == ______cmat[i][j]______ ) /* recursive call */     (1)
      return matchRUDiag ( ____cmat, pattern+1, nr, nc, i-1, j+1____ ) ; (2)
  return ________0________ ; /* unmatched condition return */          (1)
}
int checkRUDiag ( char cmat[][100], char *pattern, int nr, int nc ) {
  int count = 0, i, j;
  for ( i = 0; i < ________nr________; i++ )                           (1)
      for ( j = 0; j < ________nc________; j++ )                       (1)
          if ( matchRUDiag(cmat, pattern, nr, nc, i, j) )
              ____________count ++ ;____________                       (1)
  return count;
}
```

Q8. Given a sorted array `A[]` of n natural numbers and a natural number x, find a pair in the array whose sum is closest to x. Assume that the values of the numbers in `A[]` as well as the value of x is at most 100. Fill in the blanks of the following C function to achieve the desired result. Assume that you may use the function having the prototype as `int abs(int x);` from `stdlib.h` library that takes as input an integer and returns its absolute value. **(12)**

```
#include <stdlib.h>

#define MAXVAL 100

void closestPair ( int A[], int n, int x ) {
  /* index variables to hold the pair whose sum is closest to x */
  int idxL, idxR;

  /* variable to store current minimum difference between a pair and x */
  int minDiff = 2 * MAXVAL;

  /* auxiliary variables to be used inside iteration */
  int i, left, right, mid, elm;

  for ( i = 0; i<n; i++ ) {
      elm = A[i];

      /* using binary search technique to find the element d
          in the array A[] such that (elm + d) is closest to x */

      left  = ________________i + 1________________ ;                 (1)

      right = ________________n - 1________________ ;                 (1)

      while ( left <= right ) {
          mid = ______________( left + right ) / 2______________ ;     (1)
                    (or, left + (right - left) / 2)

          if ( A[mid] + elm == x ) {
              idxL = ______________i (or, mid)______________ ;         (1)

              idxR = ______________mid (or, i)______________ ;         (1)

              minDiff = ______________0______________ ;                (1)

              break;
          }

          /* check if this pair is closer than the closest pair so far */
          if ( abs(A[mid] + elm - x) < minDiff ) {
              minDiff = __________abs(A[mid] + elm - x)__________ ;    (1)

              idxL = ______________i (or, mid)______________ ;         (1)

              idxR = ______________mid (or, i)______________ ;         (1)
          }

          /* update binary searching ranges accordingly */
          if ( A[mid] + elm < x )
              left = ______________mid + 1______________ ;             (1)
          else
              right = ______________mid - 1______________ ;            (1)
      }
  }
  printf ("Pair = (%d , %d)", ______A[idxL]______ , ______A[idxR]______);   (1)
}
```

Q9. A k-nearly sorted array is one where each element is at a distance of at most k from its actual sorted position. For example, $\{2,6,3,12,56,8\}$ is a 3-nearly sorted array. The following C function takes as input a k-nearly sorted array and sorts it. Fill in the blanks of the following C function to achieve the desired result. **(6)**

```
void nearlySort ( int A[], int size, int k ) {
  int i, j, key;
  for ( i=1; i<size; i++ ) {
      key = ________________A[i]________________ ;                    (1)
      j = i - 1;
      /* in the following loop, j must have a non-negative value
          as it represents an index of this k-nearly sorted array */
      while ( ( _____________(i-k>=0 && i-k<=j) || (j>=0)_____________ )   (3)
             && (A[j] > key) ) {
          A[j+1] = A[j];
          ____________________j-- ;____________________ /* update j */   (1)
      }
      _________________A[j+1]_________________ = key ;                 (1)
  }
}
```

Q10. Represent the following numbers in the mentioned formats.

(a) The Binary outcome of $(3-6)$ when operated in 1's complement form. Assume, there are 4 bits allocated for storage of these numbers. State whether there is an overflow. **(1.5)**

Answer: `0011 (+3) + 1001 (-6) = 1100 (-3)     NO Overflow!`

(b) The Binary outcome of $(6+7)$ when operated in 2's complement form. Assume, there are 4 bits allocated for storage of these numbers. State whether there is an overflow. **(1.5)**

Answer: `0110 (+6) + 0111 (+7) = 1101 (-3)     Overflow!`

(c) The Binary representation of -119.362 in IEEE754 Single-Precision Floating-Point format. **(2)**

Answer: `1 (sign)   10000101 (exp.)   11011101011100101011000 (mant.)`

The question paper ends here.