ASSIGNMENT-10 SOLUTION

1. Consider a situation where swap operation is very costly. Which of the following sorting algorithms should be preferred so that the number of swap operations are minimized in general?
   a) Selection sort
   b) Insertion sort
   c) Bubble sort
   d) None of the above

Solution: (a) Selection sort
Selection sort makes O(n) swaps which is minimum among all sorting algorithms mentioned above.

2. Select the appropriate code snippet that performs bubble sort.

a)
```
for(int j=arr.length-1; j>=0; j--)
{
        for(int k=0; k<j; k++)
        {
                if(arr[k] > arr[k+1])
                {
                        int temp = arr[k];
                        arr[k] = arr[k+1];
                        arr[k+1] = temp;
                }
        }
}
```
b)
```
for(int j=arr.length-1; j>=0; j--)
{
        for(int k=0; k<j; k++)
        {
                if(arr[k] < arr[k+1])
                {
                        int temp = arr[k];
                        arr[k] = arr[k+1];
                        arr[k+1] = temp;
                }
        }
}
```
c)
```
for(int j=arr.length; j>=0; j--)
{
        for(int k=0; k<j; k++)
        {
                if(arr[k] > arr[k+1])
                {
                        int temp = arr[k];
                        arr[k] = arr[k+1];
                        arr[k+1] = temp;
                }
        }
}
```

d) None of the above

Solution: (a)
The outer loop keeps count of number of iterations, and the inner loop checks to see if swapping is necessary.

3. Which is the appropriate code to improve the best-case efficiency in bubble sort? (The input is already sorted)

a)
```
boolean swapped = false;
for(int j=arr.length-1; j>=0 && swapped; j--)
{
        swapped = true;
        for(int k=0; k<j; k++)
        {
                if(arr[k] > arr[k+1])
                {
                        int temp = arr[k];
                        arr[k] = arr[k+1];
                        arr[k+1] = temp;
                        swapped = false;
                }
        }
}
```

b)
```
boolean swapped = true;
for(int j=arr.length-1; j>=0 && swapped; j--)
{
        swapped = false;
        for(int k=0; k<j; k++)
        {
                if(arr[k] > arr[k+1])
                {
                        int temp = arr[k];
                        arr[k] = arr[k+1];
                        arr[k+1] = temp;
                }
        }
}
```

c)
```
boolean swapped = true;
for(int j=arr.length-1; j>=0 && swapped; j--)
{
        swapped = false;
        for(int k=0; k<j; k++)
        {
                if(arr[k] > arr[k+1])
                {
                        int temp = arr[k];
                        arr[k] = arr[k+1];
                        arr[k+1] = temp;
                        swapped = true;
                }
            }
        }
```

d)
```
boolean swapped = true;
for(int j=arr.length-1; j>=0 && swapped; j--)
{
        for(int k=0; k<j; k++)
        {
                if(arr[k] > arr[k+1])
                {
                        int temp = arr[k];
                        arr[k] = arr[k+1];
                        arr[k+1] = temp;
                        swapped = true;
                }
        }
}
```

Solution: (c)
A boolean variable 'swapped' determines whether any swapping has happened in a particular iteration, if no swapping has occurred, then the given array is sorted and no more iterations are required.

Sorted: 5   unsorted: 4 3 2 1

Insert all elements less than 5 on the left (Considering 5 as the key)

Now key value is 4 and array will look like this

Sorted: 4 5   unsorted: 3 2 1

Similarly for all the cases the key will always be the newly inserted value and all the values will be compared to that key and inserted in to proper position.

4. In ……………, search start at the beginning of the list and check every element in the list.
    a) Linear search
    b) Binary search
    c) Hash search
    d) Binary tree search

Solution: (a) linear search

5. The Worst case occur in linear search algorithm when
    a) Item is somewhere in the middle of the array
    b) Item is not in the array at all
    c) Item is the last element in the array
    d) Item is the last element in the array or is not there at all

Solution: (d) Item is the last element in the array or is not there at all

6. What is the output of following program?

```
# include <stdio.h>
void func(int x)
{
   x = 40;
}

int main()
{
  int y = 30;
  func(y);
  printf("%d", y);
  return 0;
}
```

Solution: 30
Parameters are always passed by value in C. Therefore, in the above code, value of y is not modified using the function func().
Note that everything is passed by value in C. We only get the effect of pass by reference using pointers.

7. What would be the equivalent pointer expression for referring the array element a[i][j][k][l]?
    a) (((*(a+i)+j)+k)+l)
    b) *(*(*(*(a+i)+j)+k)+l)
    c) (*(*(a+i)+j)+k+l)
    d) *((a+i)+j+k+l)

Solution: (b)

8. Bisection method is used to find
    a) Derivative of a function at a given point
    b) Numerical integration of a function within a range
    c) Root of the function
    d) None of the above

Solution: (c) Root of the function

9. What is the output?

```
#include <stdio.h>
   int main()
   {
      char *s = "hello";
      char *p = s;
      printf("%c\t%c", *(p + 1), s[1]);
      return 0;
   }
```

    a) h   e
    b) e   l
    c) h   h
    d) e   e

Solution: (d) e   e

p points to the base address of ‘hello’ i.e. h. so *(p+1)= the next character i.e. e. Similarly s[1] is also e. This is a simple example of pointer arithmetic on strings.

10. What will be output when you will execute following c code?

```
#include<stdio.h>
int main()
{
   short num[3][2]={2,7,10,12,15,18};
   printf("%d  %d",*(num+1)[1],**(num+2));
   return 0;
}
```

a) 12 18
b) 18 18
c) 15 15
d) 12 15

Solution: (c) 15 15

(num+1)[1]=(*((num+1)+1))=*(*(num+2))=*(num[2])=num[2][0]=15
And **(num+2)=*(num[2]+0)=num[2][0]=15
This is example of pointer arithmetic on array.

11. What will be the output?

```
#include<stdio.h>
#define X 3
int main()
{
#if !X
   printf("hello");
#else
   printf("world");

#endif
   return 0;
}
```

a) hello
b) world
c) compiler error
d) run time error

Solution: (b) world

12. Find the output of the following program

```
#include <stdio.h>
int main()
{
   int *ptr, a = 10;
   ptr = &a;
   *ptr += 5;
   printf("%d,%d ", *ptr, a);
```

```
    return 0;
}
```

a) 15,10
b) 15,15
c) 10,15
d) 10,10

Solution: (b) The pointer variable ptr contains the address of the variable a. Incrementing the value at address also modifies the content of a. Thus, both will be containing 15.

13. Find the output of the C code given below

```
#include <stdio.h>
int main()
{
    int ary[4] = {1, 2, 3, 4};
    int *p;
    p = ary + 3;
    *p = 5;
    printf("%d\n", ary[3]);
    return 0;
}
```

Solution: The pointer p contains the address of the last element of the array. Assigning the value at that address to 5 changes the content of the array ary. The last element is replaced with 5. Thus, ary[3] is 5.

14. Find the output of the following program

```
#include <stdio.h>
int main()
{
    int *ptr, a = 5;
    ptr = &a;
    *ptr += 2;
    printf("%d,%d", *ptr, a);
    return 0;
}
```

Solution: 7,7 (short answer type) [Hints: Write exactly what will be printed including “,”]
The pointer variable ptr contains the address of the variable a. Incrementing the value at address also modifies the content of a. Thus, both will be containing 7.

15. What is the solution of the equation given below using Bisection Method upto three decimal places? (Consider the root lying on positive quadrant only)

$$f(x) = xe^x - 3x - 2$$

Solution: 1.472
The root lies between 1 and 2. Using bisection method, we can find the root as 1.472 (upto three decimal accuracy)