Week 4 Assessment

Question 1 :

Q : Consider the following code segment. Assume n to be a positive integer.

```
for(i=1; i<=n; i++)
{
   for(j=1; j<=n; j=j+i)
  {
      printf("Hi");
   }
}
```

Let T(n) represent the total running time of the above code segment. Specify which of the following statements are correct.
T(n)=O(n^2)
T(n)=O(n*log n)
T(n)≠O(n^4)
T(n)=O(log n)

A : T(n) = O(nlogn) and T(n) = O(n^2). In the i-th iteration, the inner loop runs O(n/i) times, and therefore, the overall complexity is O(n+n/1+n/2+...+1), and this is O(nlogn).

Question 2 :

Q : Let P(x)=3x^2 + 4x + 2 be a polynomial in the variable x. If Horner's rule is used, what is the number of multiplications to compute P(10)?

A : 2 multiplications. By Horner's rule $P_n(x) = a_n$ and $P_i(x) = a_i + x * P_{i+1}(x)$. To find P(k), we need to calculate $P_0(k)$. $P_0(10) = 2 + 10 * (4 + 10 * 3) = 342$.

Question 3 :

Q : To multiply two six digit numbers what is the number of single digit multiplication operations performed by Karatsuba's Algorithm?

A : 12 multiplications. By applying karatsuba's multiplication, we find the product of two 3 digit numbers recursively, ie 4 + 4 = 8 multiplications. Then we combine the results using 4 more multiplications, hence 8 + 4 = 12 multiplications in total.

Question 4 :

Q : What is the degree of the polynomial defined by R(x) = P(x) * Q(x) where P(x) = x^3+x^2+1 and Q(x) = x^2+ x + 1?

A : The degree of a polynomial is the highest degree of its terms. In R(x), the term with the highest degree will we obtained by multiplying x^3 of P(x) and x ^ 2 of Q(x). So, the degree is 5.

Question 5 :

Q : For the following array and the specified search key, what will be the number of comparisons performed by unordered linear search?

array = { 10, 7, 3, 9, 11, 18, 15, -6, 25, 9, 16, 19}
search key = 15

A : The element '15' is 7th position. So, an unordered linear search will take 7 comparisons.

Question 6 :

Q : For the following array and the specified search key, what will be the number of iterations performed by binary search?

array = {-6, 3, 4, 7, 9, 10, 11, 15, 16, 18, 19, 22, 25 , 33, 47}
search key = 12

A :

1st comparison : {-6, 3, 4, 7, 9, 10, 11, **15**, 16, 18, 19, 22, 25, 33, 47}
2nd comparison : {-6, 3, 4, **7**, 9, 10, 11}
3rd comparison : {9, **10**, 11}
4th comparison : {**11**}

Binary search would exit failing to find '12' as there are no more elements to search. The answer therefore is 4.

Question 7 :

Q : Can linear search take lesser number of comparisons than binary search to find an element in an array?

A : Yes. A linear search takes only one comparison to search for the **first** element whereas with binary search it is at least 1 (1 when the size of an array is 1, > 1 otherwise).

Question 8 :

Q : Given the two arrays below which are to be merged into a single array sorted in ascending order, total number of comparisons required to merge these arrays will be:

A1 = (10, 20, 30, 40, 50, 60)
A2 = (15, 25, 35, 45)

A : 8 comparisons are needed to merge the first 8 elements. After merging the first 8 elements into an array M :

M = {10, 15, 20, 25, 30, 35, 40, 45}
A1 = {50, 60}
A2 = {}

At this point, the elements in A1 can be added into M without any comparisons. Therefore the total number of comparisons required are 8.

Question 9 :

Q : You are given a sorted array A of size 200 elements. Five values are appended at the end of the array. Which sorting algorithm would be a better candidate for sorting the whole array?

A : Let the size of the array initially be n. k elements are added at the end. When using insertion sort, only the last 5 elements have to be inserted into appropriate locations. Doing so, would result in atmost n + (n + 1) + (n + 2) .. + (n + k -1) swaps, which is in O(kn). But, merge sort still takes O((n + k) log (n + k)) steps to sort. When k is significantly smaller than n (k << n) as is the case here (5 and 200), insertions sort takes O(n) steps, whereas merge sort takes O(n log n) step. Hence, insertion sort is preferred.

Question 10 :

Q : Suppose that the range of values k in counting sort is not of O(n). Is counting sort still guaranteed to sort in linear time?

A : No. In counting sort, a collection of objects are sorted according to keys that are small integers. Counting sort uses these integers (keys, in general) as indices into an array. The size of the array could only defined when the range of the values is in O(n).

Question 11 :

Q : Does counting sort use comparisons for sorting?

A : There are no comparisons involved. It uses integers as indices into an array.

Question 12 :

Q : If radix sort uses an unstable sorting algorithm to sort the digits in radix sort, is it still guaranteed to work correctly?

A : No. For example, consider sorting 13, 12.
Sorting the numbers based on the least significant digit :
1<u>2</u>, 1<u>3</u>.

Sorting the numbers based on the next significant digit :
<u>1</u>2, <u>1</u>3

In this step, the sorting algorithm will just see 1 and 1. If the sorting algorithm is not stable, it could output 13, 12 which is wrong. So, unless it is stable, the order that is created by the previous steps is lost.

Question 13 :

Q : Let the numbers 794, 332, 561, 342, 200, 607, and 893 be sorted using radix sort. What will be the sixth number in the sequence of numbers after sorting the second digit?

A : The numbers to be sorted are 794, 332, 561, 342, 200, 607, and 893.
Sorting based on the least significant digit :
20<u>0</u>, 56<u>1</u>, 33<u>2</u>, 34<u>2</u>, 89<u>3</u>, 79<u>4</u>, 60<u>7</u>

Sorting the numbers based on the next significant digit :
2<u>0</u>0, 6<u>0</u>7, 3<u>3</u>2, 3<u>4</u>2, 5<u>6</u>1, **8<u>9</u>3**, 7<u>9</u>4

The answer therefore is 893.

Question 14 :

Q : Find median (arithmetic mean if more than one median) for the given array. A = { 10, 30, 20, 17, 25, 28, 13, 27}

A : Number of elements in the array are even. Upon ordering the array {10, 13, 17, **20, 25**, 27, 28, 30}, the median is the mean of 20, 25 which is 22.5.

Question 15 :

Q : Find median (arithmetic mean if more than one median) for the given array. A = { 10, 30, 20, 25, 28, 13, 27}

A : Number of elements in the array are odd. The middlemost element in the ordered array {10, 13, 20, **25**, 27, 28, 30} is the median which is 25.

Week 4 Programming Assignments

Question 1:

You are given two words (not necessarily meaningful words) over the lower case English alphabet. They are to be merged into a single new word in which all the letters (including repetitions) in the given two words occur in increasing order (‘a’ is the least and ‘z’ is the largest) from left to right.

Example: If the two words are “ehllo” and “wrold”, the output is “dehllloorw”.
Explanation: d, e, h , l, o, r, w is the increasing order among the distinct letters. Since all the letters must occur in the output word, the output is “dehllloorw”.

Input: Two words, one on each line.
Output: The merged word as described in the problem statement.

Constraints: 0 < length of each word <= 10.

Hint: Study the sorting algorithms mentioned in the lectures and pick the one which best suits this problem.

Test Cases:

Public Test Cases:

Input	Output
hello world	dehllloorw
test case	aceesstt
is it	iist

Private Test Cases:

Input	Output
abc abcd	aabbccd
abc abc	aabbcc

ace bdf	abcdef
xyzxyzxyzx abcabcabca	aaaabbbcccxxxxyyyzzz

Solution:

```
include <stdio.h>

#define MAXLEN 10

void printChar(char c, int times) {
        int i;
        for (i = 0; i < times; i++)
        printf("%c", c);
}

int main() {
        /* Buffers to hold 2 words */
        char word1[MAXLEN + 1], word2[MAXLEN + 1];

        /* The array used for counting sort */
    /* Chars can be treated as integers */
        int hash[26] = {0};

        int i = 0; // Iterator

        /* Read both the lines */
        scanf("%s", word1);
        scanf("%s", word2);

        for (i = 0; i < MAXLEN + 1 && word1[i] != '\0'; i++)
           hash[word1[i] - 'a']++;

        for (i = 0; i < MAXLEN + 1 && word2[i] != '\0'; i++)
           hash[word2[i] - 'a']++;

        for (i = 0; i < 26; i++)
           printChar(i + 'a', hash[i]);

        printf("\n");
```

```
        return 0;
}
```

Question 2:

Note: A video solution to this problem may be found here: https://www.youtube.com/watch?v=dE9F0CAeYgY

You are given a set of words over the lowercase English alphabet. They are to be sorted in alphabetical order.

Alphabetical order: It is the order in which words are organized in a dictionary or in an telephone directory.

For example,

- *abc* comes before *xab*. **Reason :** The first letters differ. So, they are compared. **a** comes before **x** in English alphabet.
- *aab* comes before *abc*. **Reason :** The first letters are same (**a**ab **a**bc). So, the next 2 letters are compared (a**a**b a**b**c).
- *aab* comes before *aaba*. **Reason :** The first 3 letters in ***aab*** are same as the first 3 letters in ***aab****a* and there are no more letters in *aab* to compare against *aaba*. So, the word with the smaller length comes first (aab).

Input: The first line has 'n', the number of words, followed by n lines each of which contains a word.

Output: Words in alphabetical order, one one each line.

Constraints: 0 < n <=100, 0 < length of each word <= 10

Hint: See how two-dimensional arrays can help.

Test Cases:

Public Test Cases:

Input	**Output**
2 hello world	hello world
2 antelope ant	ant antelope

5 which was when while were	was were when which while

Private Test Cases:

Input	**Output**
1 a	a
2 aabbccddee aabbccddef	aabbccddee aabbccddef
4 d ce bfg ahij	ahij bfg ce d
3 abc ab a	a ab abc
6 bca cab bac acb abc cba	abc aca bac bca cab cba

Solution:

```
#include <stdio.h>

#define MAXWORDS 100
#define WORDLENGTH 10

/* Function to swap two integers */
void swap(int *a, int *b) {
```

```
        int t = *a;
        *a = *b;
        *b = t;
}

/* Function to compare two strings */
int compare(char *s1, char *s2) {
        while (*s1 && (*s1 == *s2)) {
           s1++;
           s2++;
        }
        return *s1 - *s2;
}

int main()
{
        int n, i = 0, j = 0;
        char words[MAXWORDS][WORDLENGTH + 1];

    /* We'll be manipulating 'order' rather than words */
    /* It is easier this way, we need not move words */
        int order[MAXWORDS] = {0};

        scanf("%d", &n);

        for (i = 0; i < n; i++) {
                scanf("%s", words[i]);
                order[i] = i;
        }

        /* Standard insertion sort */
        for (i = 1 ; i < n; i++) {
                j = i;

                while (j > 0 && compare(words[order[j-1]], words[order[j]]) > 0) {
                swap (&order[j], &order[j-1]);
                j--;
                }
        }

        for (i = 0; i < n - 1; i++)
```

```
            printf("%s\n", words[order[i]]);
        printf("%s", words[order[i]]);

        return 0;
}
```

Question 3 :

A polynomial of degree n is of the form $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$. Given two polynomials f(x) and g(x) of degrees n and m respectively, write a program to find the polynomial h(x) given by,

h(x) = f(x) * g(x)

Input: Line 1 contains n and m separated by space.
Line 2 contains the coefficients a_n, a_{n-1}…, a_0 of f(x) separated by space.
Line 3 contains the coefficients b_m, b_{m-1}…, b_0 of g(x) separated by space.

Output: The degree of h(x) followed by the coefficients c_k, c_{k-1}…, c_0 of h(x) in next line separated by space.

Sample Input:

2 2
1 2 3
3 2 1

Sample Output:

4
3 8 14 8 3

Constraints:

1 <= n <= 10
1 <= a_i <= 100

Test Cases:

Public Test Cases:

Input	Output
2 2 1 2 3 3 2 1	4 3 8 14 8 3
2 3 1 1 1 1 1 1 1	5 1 2 3 3 2 1

3 3 1 0 1 0 1 1 0 1	6 1 1 1 2 0 1 0

Private Test Cases:

Input	**Output**
4 4 10 20 30 40 50 50 40 30 20 10	8 500 1400 2600 4000 5500 4000 2600 1400 500
10 10 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0	20 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5 6 2 3 4 5 6 7 9 8 7 6 5 4 3	11 18 43 74 110 150 193 166 135 104 74 46 21
6 5 71 72 73 74 75 76 77 90 91 92 93 94 95	11 6390 12941 19654 26530 33570 40775 41330 34865 28232 21430 14458 7315
1 1 5 0 5 0	2 25 0 0

Solution:

```
// Program - Polynomial Multiplication.

#include <stdio.h>

// Polynomial Multiplication.
void polynomialMultiply(int n, int m, int *a, int *b)
{
  int c[30] = {0}; // Initialize.

  for(int i = 0; i <= n; ++i)
   for(int j = 0; j <= m; ++j)
    c[i + j] += a[i] * b[j];
```

```
  // Print.
  printf("%d\n", n + m);
  for(int i = 0; i <= n + m; ++i)
  {
    printf("%d", c[i]);
    if(i < n + m) printf(" ");
  }
}
//

int main()
{
  int n, m;
  int a[15], b[15];

  scanf("%d %d", &n, &m);
  for(int i = 0; i <= n; ++i)
   scanf("%d", &a[i]);
  for(int i = 0; i <= m; ++i)
   scanf("%d", &b[i]);

  polynomialMultiply(n, m, a, b);
  return 0;
}
```

Question 4:

You are given a polynomial of degree n. The polynomial is of the form $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$. For given values k and m, You are required to find P(k) at the end of the m^{th} iteration of Horner's rule. The steps involved in the Horner's rule are given below,

$P_n(x) = a_n$

$P_{n-1}(x) = a_{n-1} + x * P_n(x)$ — 1^{st} iteration.

$P_{n-2}(x) = a_{n-2} + x * P_{n-1}(x)$ — 2^{nd} iteration.

.

.

$P_0(x) = a_0 + x * P_1(x)$ — n^{th} iteration.

In general, $P_i(x) = a_i + x * P_{i+1}(x)$ and $P_0(x)$ is the final result. The input to your program is as follows,

Line 1 contains the integers n, m and k separated by space.
Line 2 contains the coefficients $a_n, a_{n-1} \ldots, a_0$ separated by space.

Input: Integers n, m, k and the coefficients as described above.
Output: P(k) value at the end of the m^{th} iteration.

Sample Input:

2 2 5
3 2 1

Sample Output:

86

Constraints:

1 <= n, k, m <= 10
0 <= a_i <=10

Test Cases:

Public test cases:

Input	Output
2 2 5 3 2 1	86

3 2 2 4 3 2 1	24
3 1 5 3 1 0 1	16

Private test cases:

Input	**Output**
2 1 1 1 0 5	1
4 4 2 5 4 3 2 1	129
5 3 10 5 6 7 8 9 10	5678
4 3 10 6 7 8 9 10	6789
4 1 1 1 0 0 1 1	1

Solution:

```
// Program - Horner's Rule.

#include <stdio.h>

int main()
{
  int n, m, k, a[30];
  int ans;

  scanf("%d %d %d", &n, &m, &k);
  ans = 0;
  for(int i = 0; i <= n; ++i)
   scanf("%d", &a[i]);

  // Find the m'th iteration value.
```

```
  ans = a[0];
  for(int i = 1; i <= m; ++i)
   ans = a[i] + k * ans;

  printf("%d", ans);
  return 0;
}
```