104 - Introduction to Computer Science - Final Exam

1. Why do we need to convert bits into electromagnetic signals at the physical layer? What are the differences between **analog data** and **digital data**? (4%)

   Because bits as the representation of two possible values stored in the memory of a node (host, router, or switch), cannot be sent directly to the transmission medium (wire or air). So the bits need to be changed to signals before transmission.

   Digital data take on discrete values. For example, data are stored in computer memory in the form of 0s and 1s. They can be converted to a digital signal or modulated into an analog signal for transmission across a medium.

2. A mono-programming operating system runs programs that on average need 10 microseconds access to the CPU and 70 microseconds access to the I/O devices. What percentage of time is the CPU idle? (3%)

   70 / (70 + 10) × 100 = 87.5%

3. Three processes (A, B, and C) are running concurrently. Process A has acquired File 1. Process B has acquired File 2, but needs File 1. Process C has acquired File 3, but needs File2. Draw a diagram for these processes. Is this a deadlock situation? If your answer is 'no', show how the processes can eventually finish their tasks. (6%)

   Deadlock happens when processes are all waiting for resources held by other processes: they are all waiting for each other. This happens when the operating system does not put resource restrictions on processes. Starvation happens when the operating system puts too many resource restrictions on a process. If a process must wait until it can get all of the resources that it needs before it starts to execute, it may never start.

4. Using the **selection sort algorithm**, manually sort the following list and show your work in each pass using a table. According to this sorted list, using the **binary search algorithm**, trace the steps followed to find 23. At each step, show the values of *first*, *last*, and *mid*. (6%)

7	23	12	2	9	43	31

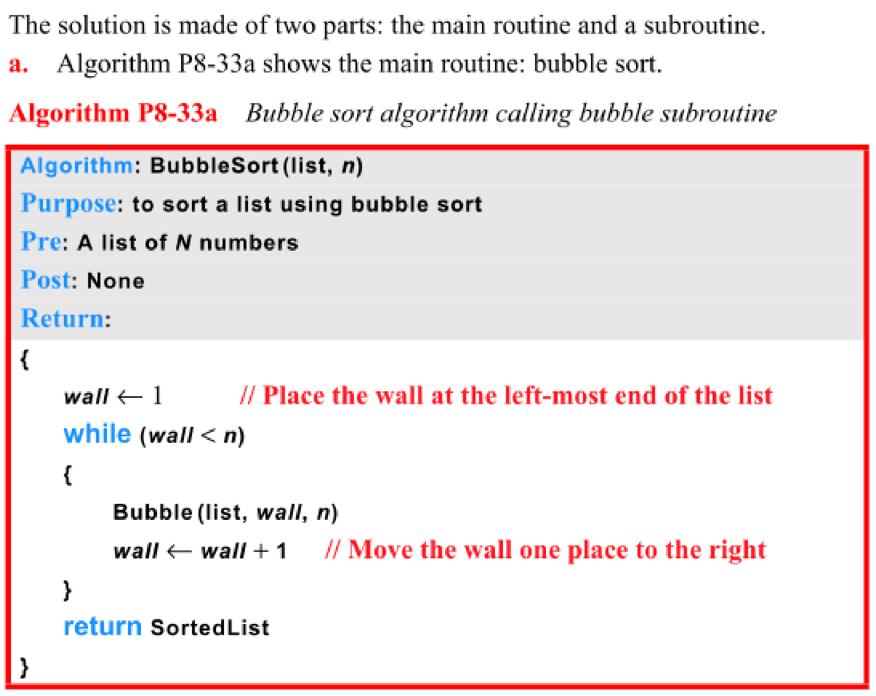
Pass	List						
	7	23	12	2	9	43	31
1	2	23	12	7	9	43	31
2	2	7	23	12	9	43	31
3	2	7	9	23	12	43	31
4	2	7	9	12	23	43	31
5	2	7	9	12	23	43	31
6	2	7	9	12	23	31	43

7	2	7	9	12	23	31	43

first	last	mid
1	7	4
5	7	6
5	5	5

1	2	3	4	5	6	7	
2	7	9	12	23	31	43	Target > 12
				23	31	43	Target < 31
				23			Target = 23

5. Write an algorithm in pseudocode for the **bubble sort** using a *subalgorithm* to do bubbling in the unsorted *sublist*. (6%)



The solution is made of two parts: the main routine and a subroutine.

a. Algorithm P8-33a shows the main routine: bubble sort.

Algorithm P8-33a *Bubble sort algorithm calling bubble subroutine*

```
Algorithm: BubbleSort(list, n)
Purpose: to sort a list using bubble sort
Pre: A list of N numbers
Post: None
Return:
{
    wall ← 1            // Place the wall at the left-most end of the list
    while (wall < n)
    {
        Bubble(list, wall, n)
        wall ← wall + 1     // Move the wall one place to the right
    }
    return SortedList
}
```

b. Algorithm P8-33b shows the subroutine, bubble, called by the main routine.

Algorithm P8-33b *Bubble subroutine*

```
Algorithm: Bubble (list, wall, n)
Purpose: to bubble an unsorted list
Pre: A list, N and location of the wall
Post: None
Return:
{
    $cur \leftarrow n$                         // Start from the end of the list
    while ($cur > wall$)                 // Bubble the smallest to the left of list
    {
        if ($A_{cur} < A_{cur-1}$)            // Bubble one location to the left
        {
            Temp $\leftarrow A_{cur}$
            $A_{cur} \leftarrow A_{cur-1}$
            $A_{cur-1} \leftarrow$ Temp
        }
        $cur \leftarrow cur - 1$
    }
}
```

6. Write a **recursive algorithm** in pseudocode to find the combination of n objects taken k at a time using the definition in following function. Use the definition to find the value of C (10, 3). (6%)

$$C(n,k) = \begin{bmatrix} 1 & \text{if } k = 0 \text{ or } n = k \\ C(n-1,k) + C(n-1,k-1) & \text{if } n > k > 0 \end{bmatrix}$$

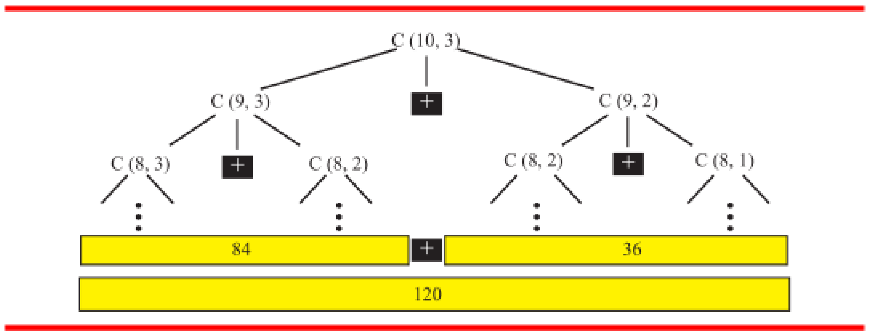
Algorithm P8-19 shows the pseudocode for evaluating combination.

Algorithm P8-19 *Combination*

```
Algorithm: Combination (n, k)
Purpose: Finds the combination of n objects k at a time
Pre: n and k
Post: None
Return: C (n, k)
{
    If ($k = 0$ or $n = k$)
        return 1
    else
        return C ($n - 1, k$) + C ($n - 1, k - 1$)
}
```

a. Figure P8-20 shows the evaluation of C(10, 3).

Figure P8-20 *Evaluation of C(10, 3)*



7. Assume there is a two-dimensional array declared int A[5][20]. Each element occupies 2 bytes using **row-major storage**. The address of first one A[0][0] is 1320 in memory. Write the **equation** that can calculate the address of A[i][j] in memory ($0 \leq i < 5,\ 0 \leq j < 20$). (6%)

1320 + (20 * i + j) * 2

8. We know black-box testing is without knowing what is inside the software and how it works. So there are several methods used in black-box testing. Describe and compare among **exhaustive**, **random** and **boundary-value testing**. (3%)

Exhaustive testing: The best black-box test method is to test the software for all possible values in the input domain. However, in complex software the input domain is so huge that it is often impractical to do so.

Random testing: In random testing, a subset of values in the input domain is selected for testing. It is very important that the subset be chosen in such a way that the values are distributed over the domain input. The use of random number generators can be very helpful in this case.

Boundary-value testing: Errors often happen when boundary values are encountered. For example, if a module defines that one of its inputs must be greater than or equal to 100, it is very important that module be tested for the boundary value 100. If the module fails at this boundary value, it is possible that some condition in the module's code such as $x \geq 100$ is written as $x > 100$.

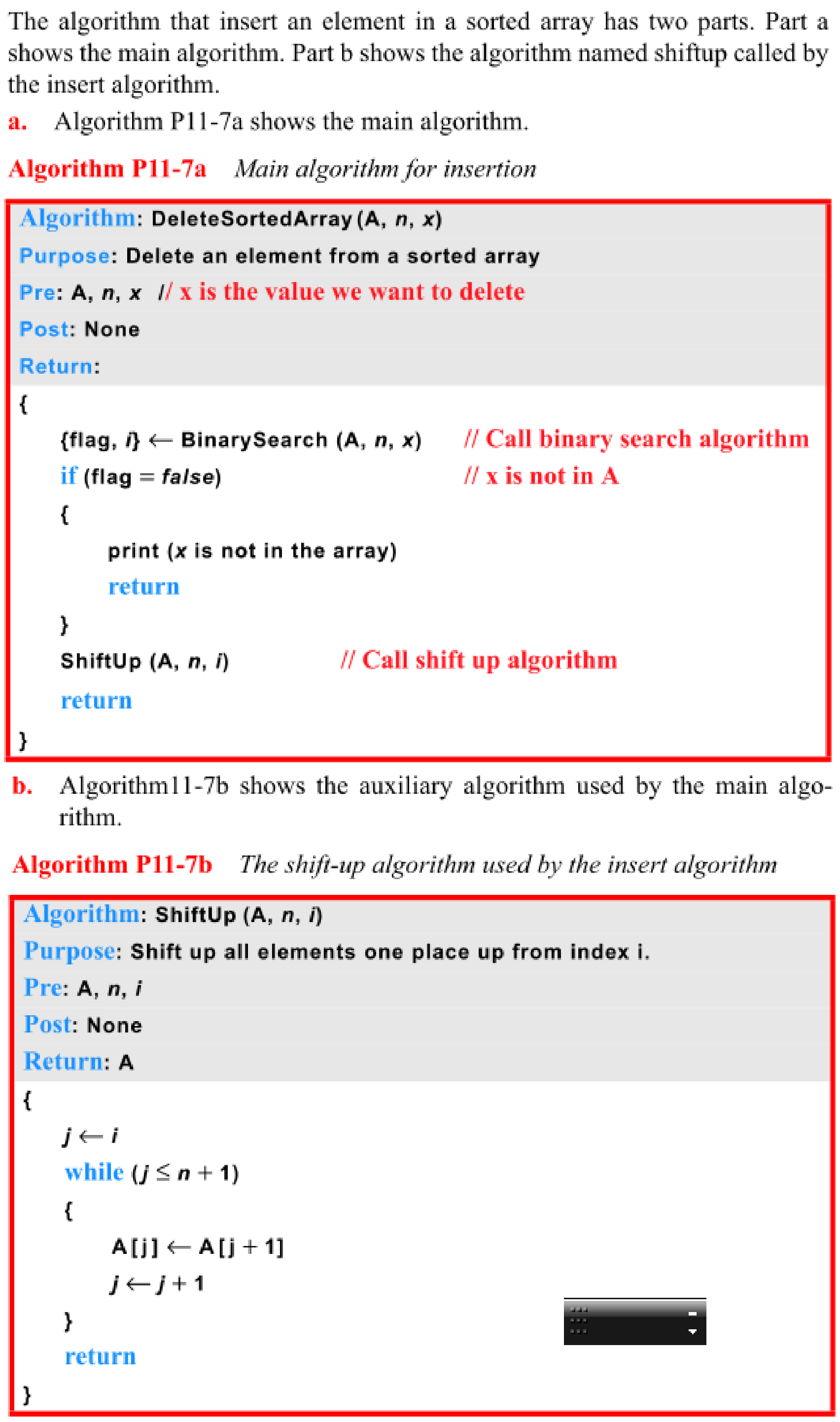
9. Find how many **times** the *statement* in the following code segment in C is executed in the following program and write the code using *for* loop. (6%)

```
A = 10
do {
   if (A % 3 == 0)
      statement;
   A = A + 1;
} while (A ≤ 20)
```

3 (A = 12,15,18)

```
for (var A = 10; A <= 20; A ++) {
   if (A % 3 == 0)
       statement;
}
```

10. Write an algorithm to **delete an element in a sorted array**. The algorithm must call a search algorithm to find the location of deletion. (6%)



The algorithm that insert an element in a sorted array has two parts. Part a shows the main algorithm. Part b shows the algorithm named shiftup called by the insert algorithm.

a. Algorithm P11-7a shows the main algorithm.

Algorithm P11-7a *Main algorithm for insertion*

```
Algorithm: DeleteSortedArray (A, n, x)
Purpose: Delete an element from a sorted array
Pre: A, n, x  // x is the value we want to delete
Post: None
Return:
{
    {flag, i} ← BinarySearch (A, n, x)     // Call binary search algorithm
    if (flag = false)                      // x is not in A
    {
        print (x is not in the array)
        return
    }
    ShiftUp (A, n, i)              // Call shift up algorithm
    return
}
```

b. Algorithm11-7b shows the auxiliary algorithm used by the main algorithm.

Algorithm P11-7b *The shift-up algorithm used by the insert algorithm*

```
Algorithm: ShiftUp (A, n, i)
Purpose: Shift up all elements one place up from index i.
Pre: A, n, i
Post: None
Return: A
{
    j ← i
    while (j ≤ n + 1)
    {
        A[j] ← A[j + 1]
        j ← j + 1
    }
    return
}
```

11. Write an algorithm segment using **while loops** to concatenate the contents of stack S2 with the contents of stack S1. After the concatenation, the elements of stack S2 should be above the elements of stack S1 and stack S2 should be empty. (6%)

The following shows the algorithm segment.

```
stack(Temp)
while (NOT empty (S2))
{
    pop (S2, x)
    push (Temp, x)                    // Temp is a temporary stack
}
while (NOT empty (Temp))
{
    pop (Temp, x)
    push (S2, x)
}
```

12. What does the following code segment do? Show the result. (4%)

```
int x = 0;
if (x = 0 && x == 0)
        printf("%d\n", x);
printf("%d\n", x);
```

0

13. What does the following code segment do? Show the result. (6%)

```
void DesignFun(int *a, int *b) {
    (*a)^=(*b)^=(*a)^=(*b); }
int main(void){
    int x = 3, y = 5;
    DesignFun (&x, &y);
    printf("%d %d\n",x,y);
    return 0; }
```

5 3

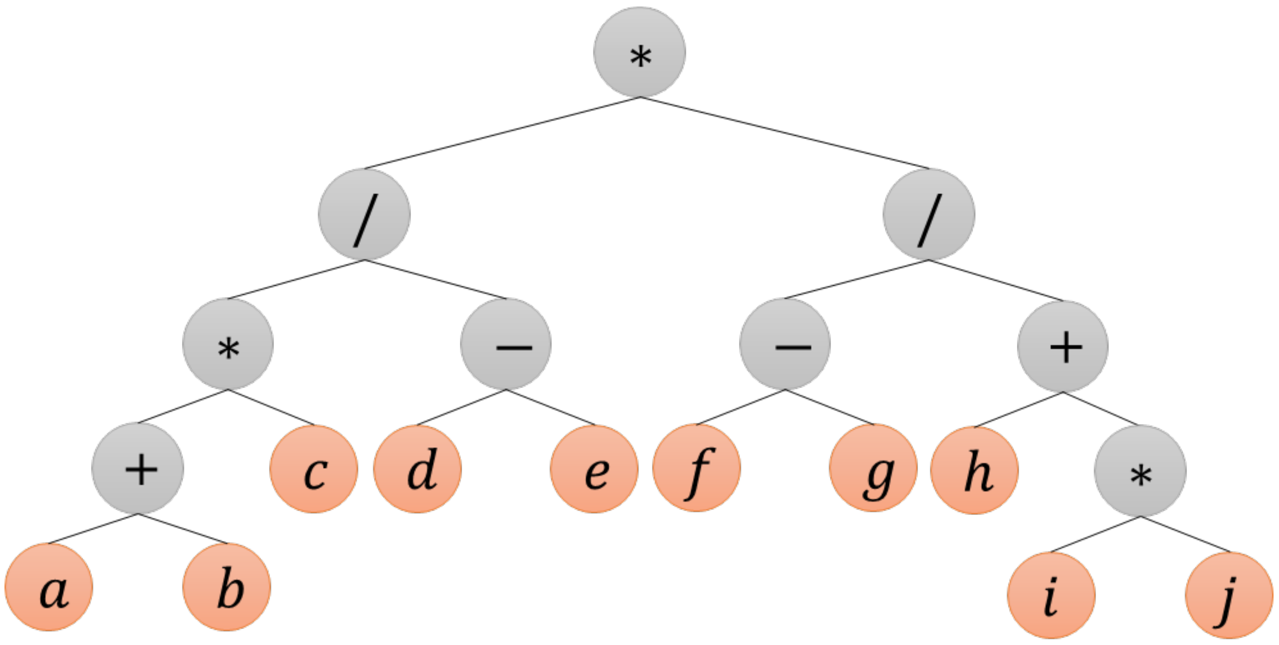
14. What does the following code segment do? Show the result. (6%)

```
int main(void) {
    int a = 5, b, c, d, e, f;
    f = (a+1)/2;
    for (b = 1; b <= f; b++) {
        for (d = f - b ; d > 0; d--) { printf(" "); }
        for (e = 1; e<= 2 * b - 1; e++) { printf("*"); }
        printf("\n"); }
    for(b = f - 1; b > 0; b--) {
        for(d = f - b; d > 0;d--) { printf(" "); }
        for(e = 1; e <= 2 * b - 1; e++) { printf("*"); }
        printf("\n"); }
}
```

```
  *
 ***
*****
 ***
  *
```

15. We know an arithmetic expression can be represented in three different formats: infix, postfix, and prefix. Show the expression [(a + b) * c / (d - e)] * [(f - g) / (h + i * j)] in **prefix notation** and its **expression tree**. (6%)

/+abc-de/-fg+h*ij



16. Write an algorithm to find out the **largest number** among the multiple integer numbers stored in array. Assume **not** use *if* or *switch* and *math* function in C language. (6%)

This is a reference answer, and you can also use the *while* to answer this question.

```
#include <stdio.h>
#include <iostream>

using namespace std;

int maxFun(int a, int b)
{
    int diff = a - b;    //after subtract
    int sign_bit= unsigned(diff) >> (sizeof(int) * 8 - 1); //according to the sign
    return sign_bit; // 0 → a is larger than b, 1 → b is larger than a
}

int main(void)
{
    int num[6] = {1,7,8,15,19,31};    //assume there is a sorted array
    int compare[6] = {};    //store the results of comparison
    int maxNum = 0;

    for (int i = 0;i < 6; i++) {
        int totalCompare = 0;

        for (int j = i + 1;j < 6; j++) {
            totalCompare = totalCompare + maxFun (num[i],num[j]);
        }
      compare[i] = totalCompare;
    }

    for (int i = 0; i < 6; i++) {      //print
      cout << compare[i] << "\n";
    }

    return 0;
}
```

17. Create the ADT package in pseudocode to implement the *insert* and *traverse* operations defined for a **general linear list** using an array as the data structure. (6%)

```
insert (List L, DataRecord x)      // Insert operation
{
    BinarySearchArray (A, n, x.key, flag, i)
    ShiftDown (A, n, i)
    A [i] ← x
    if (empty (L))
        L.first ← 1
    L.count ← L.count + 1
}
```

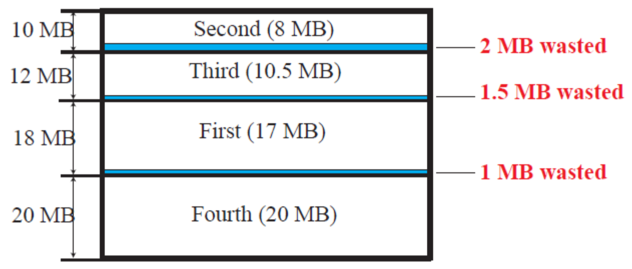
```
traverse (List L, Process)    // Traverse operation
{
    walker ← 1
    while (walker ≤ L.count)
    {
        Process (A [walker]
        walker ← walker + 1
    }
}
```

18. A multiprogramming operating system uses an apportioning scheme and divides the 60 MB of available memory into four partitions of 10MB, 12 MB, 18 MB, and 20 MB. The first program to be run needs 17 MB and occupies the third partition. The second program needs 8 MB and occupies the first partition. The third program needs 10.5 MB and occupies the second partition. Finally, the fourth program needs 20 MB and occupies the fourth partition.

a. What is the total memory wasted? (3%) ➔4.5 MB

b. What percentage of memory is wasted? (3%) ➔ 7.5 %

Figure S7.34 *Exercise 34*



Total memory used = 17 + 8 + 10.5 + 20 = 55.5 MB.
Total memory wasted = 2 + 1.5 +1 = 4.5 MB.
Percent memory wasted = 4.5 / 60 × 100 = 7.5%.

19. According to the following code segment of C language, please show the print results? (6%)

```
#include <stdio.h>
#include <stdlib.h>

int main( )
{
  int i , *ptr;
  int array[3][4]={{1,2,3,4},{4,5,6,7},{8,9,10,11}};
  ptr=(int *)array;
  printf("%d\n",array[1][2]);
  printf("%d\n %d\n",(*(array+1))[1],*((array+1)[1]));
  system("pause");
  return 0;
}
```

```
6
5
8
```

20. Use the following numbers to create **binary trees**. Which one is the **minimum deep** of binary tree? (8%)

a. 4, 1, 5, 6, 2, 3 c. 3, 2, 6, 1, 4, 5

b. 1, 2, 3, 4, 5, 6 d. 3, 2, 5, 1, 4, 6

```
a. 4
   / \
   1 5
   \ \
   2 6
   \
   3
b. 1
   \
   2
   \
   3
   \
   4
   \
   5
   \
   6
c. 3
   / \
   2 6
   / /
   1 4
   \
   5
d. 3
   / \
```

2 5

/ / \

1 4 6

The one d is the **minimum deep** of binary tree.