Midterm Exam of Data Structures (CSE, NTOU)

Student ID: _____

09:20 – 12:05, 30 October 2024; Room INS105

Name: _____

Note: Cell phones and any calculator are forbidden.

1. (20%) Convert the following infix expressions into postfix forms and prefix forms.

   (a). `a / b - c + d * e - a * c` (b). `(a + b) * (c - d)`

2. (15%) Given the following recursive program (pseudo-code) of the Fibonacci number, please convert it into a non-recursive code using the **stack** data structure.

```
int Fibo(int n) {
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else return Fibo(n-1) + Fibo(n-2);
}
```

3. (10%) Ackerman's function $A(m, n)$ is defined as:

$$A(m,n) = \begin{cases} n+1, & \text{if } m = 0 \\ A(m-1,1), & \text{if } n = 0 \\ A(m-1, A(m,n-1)), & \text{otherwise.} \end{cases}$$

   This function grows very quickly for small values of m and n. Please compute the values $A(0,0)$ and $A(2,3)$.

4. (10%) Consider the following code segments and answer the questions.

```
count = 0;
for (int i=0; i<n; i++)
    for (int j=i+1; j<n; j++)
      count++;
printf("%d", count);
```

   (a). `count` = _____

   (b). Θ(`count`) = _____

5. (10%) Analyze the lower bound on time complexity (i.e., Ω) of the following function:

```
int func(int n) {
    if (n == 1) return 0;
    else if (n % 2 == 0) return func((int)(n/2));
    else return func(n+1);
}
```

6. (10%) Let `A[1..j, 1..k]` be a two-dimensional array. Each element of `A` requires two bytes. Suppose `A[2,2]` is stored at `898`, and `A[3,4]` is stored at `918`. Assuming that memory is addressed in bytes, and the array is stored sequentially in memory. This array is **row-major** order.

   (a). What is the value of `k`?

   (b). What is the address of `A[5,7]`?

7. (10%) Given the following matrix int *A*[3][4] with integer entries. suppose we use the sparse matrix ADT to store *A*. How many bytes are required to store *A* using the sparse matrix ADT? (*Hint*: Consider the following structure on the right side for the sparse matrix ADT.)

$$A = \begin{bmatrix} 12 & 0 & 0 & 21 \\ 0 & 1 & -3 & 0 \\ 0 & 0 & 0 & 9 \end{bmatrix}$$

```
typedef struct {
    int col;
    int row;
    int value;
} term;
```

8. (10%) Consider the following function (program), where matrix `a` has the size of $m \times n$ and t nonzero entries. Suppose that it uses the sparse matrix ADT for representing matrix `a`. Suppose that $t \ll mn$. What is its time complexity in big-*O (in terms of m, n and t)*? Please give your answer as tight as possible.

```
void fast_transpose(term a[], term aT[]) {// aT: the transpose of a
    int row_terms[1001], starting_pos[M1001];
    int i, j, num_cols = a[0]->col, num_terms = a[0]->value;
    aT[0]->row = num_cols;
    aT[0]->col = a[0]->row;
    aT[0]->value = num_terms;
    if (num_terms > 0) { // nonzero matrix
        for (i = 0; i < num_cols; i++)
            row_terms[i] = 0; // initialization for the counting
        for (i = 1; i <= num_terms; i++)
            row_terms[a[i]->col]++; // counting the row items
        starting_pos[0] = 1;
        for (i = 1; i < num_cols; i++) // calculate the starting positions
            starting_pos[i] = starting_pos[i-1] + row_terms[i-1];
        for (i = 1; i <= num_terms; i++) {
            j = starting_pos[a[i]->col]++;
            aT[j]->row = a[i]->col;
            aT[j]->col = a[i]->row;
            aT[j]->value = a[i]->value;
        }
    }
}
```

9. (10%) Order the following function by growth rate in increasing order (**Note:** $\log n = \log_2 n = \lg n$):

   (a). $(\log n)!$ (b). $\log(n!)$ (c). $2^{\sqrt{n}}$ (d). $4^{\log n}$

10. (10%) Assume that we are using a linked list to implement a stack of integers as below. Please fill-in the incomplete parts in the `push()` and `pop()` function (see the codes below).

```
struct Node {
    int data;
    Node *link;
};
typedef struct Node NODE;

typedef struct {
    Node *top;
} stack;

Node stackEmpty() {
   printf("STACK EMPTY!\n");
   Node t;
   t.data = 0;
   t.link = NULL;
   return t;
}
```

```
void push(Stack *s, int item) {
    Node *temp = malloc(sizeof(Node));
    temp->data = ... // please complete it
    temp->link = ... // please complete it
    s->top = ... // please complete it
}

Node pop(Stack *s) {
   Node *temp = s->top;
   Node item;
   if (!temp) return stackEmpty();
   item.data = ... // please complete it
   s->top = ... // please complete it
   free(temp);
   return item;
}
```