CSD 102 Data Structures

Mid-Semester Exam

Date: 01-10-2024 Total Marks = 50 Time: 9:30 - 11:00 AM

Exam Instructions

1. You are **not permitted** to carry any gadget for communication/computation (including **mobile phones, scientific calculator, and smart watches**) inside the examination room.
2. It is **mandatory** for you to carry the university ID card for being permitted to take the examination. In the event of loss of the ID card, you are advised to contact Mr Gaurav Paliwal and obtain a duplicate ID card to enable you to write the examination. The time lost, if any, in the transaction for obtaining a duplicate card **shall not** be compensated for, during the examination.
3. You shall not be allowed entry to the examination room **15** minutes after the scheduled commencement of the examination, and shall not be permitted to leave the room prior to **30** minutes after the commencement of the examination.
4. It is **mandatory** to write your **roll number** on the **question paper** and **answer sheet**, as soon as you receive them.
5. Only **one** student at a time will be permitted to leave the examination hall for using the toilet.

You are advised to strictly comply with the above instructions.

Name: _____

Roll No: _____

Room No: _____

Student Signature: _____

Invigilator Signature: _____

Instructor Signature: _____

Objective (Mark ✓ in front of correct option)

1. What is the time complexity of the below function?

```
void fun(int n, int arr[])
{
  int i = 0, j = 0;
  for (; i < n; ++i)
    while (j < n && arr[i] < arr[j])
      j++;
}
```

a. $O(n^2)$
b. O(n)
c. O(n*log(n))
d. $O(n*log(n)^2)$ **[2 Marks]**

Answer: b

At first look, the time complexity seems to be $O(n^2)$ due to two loops. But, please note that the variable **j** is not initialized for each value of variable **i**. So, the inner loop runs at most **n** times.

2. The output of the following C program is

```
#include < stdio.h >
int main () {
        int arr [] = {1,2,3,4,5,6,7,8,9,0,1,2,5}, *ip = arr+4;
        printf ("%d\n", ip[2]);
        return 0;
}
```

a. 6
b. 5
c. 66
d. 7 **[2 Marks]**

Answer: d

3. The seven elements A, B, C, D, E, F and G are pushed onto a stack in reverse order, i.e., starting from G. The stack is popped five times, and each element is inserted into a queue. Two elements are deleted from the queue and pushed back onto the stack. Now, one element is popped from the stack. The popped item is

(a) A (b) B (c) F (d) G **[2 Marks]**

Answer: b

4. A circularly linked list is used to represent a Queue. A single variable p is used to access the Queue. To which node should p point such that both the operations enQueue and deQueue can be performed in constant time?

a. rear node
b. front node

c. not possible with a single pointer
d. node next to front **[2 Mark]**

Answer: a

5. Consider the following statements:

   i. First-in-first out types of computations are efficiently supported by STACKS.
   ii. Implementing LISTS on linked lists is more efficient than implementing LISTS on an array for almost all the basic LIST operations.
   iii. Implementing QUEUES on a circular array is more efficient than implementing QUEUES on a linear array with two indices.
   iv. Last-in-first-out type of computations are efficiently supported by QUEUES.

   Which of the following is correct? **[2 Marks]**

   a. (ii) is true
   b. (i) and (ii) are true
   c. (ii) and (iv) are true
   d. (iii) is true

Answer: d

6. Consider the function f defined below

```
struct item {
        int data;
        struct item *next;
};

int f (struct item *p){
        return ((p == NULL) || (p->next == NULL) || ((p->data ≥ p->next->data) && f(p->next)));
}
```

for a linked list p the function f returns '1' iff

a) p contains 0 or 1-element
b) p is non-decreasing order
c) p is non-increasing order
d) None of the above. **[3 Marks]**

Answer: c

7. An array A consists of n integers in locations A[0], A[1] ....A[n-1]. It is required to shift the elements of the array cyclically to the left by k places, where 1 <= k <= (n-1). An incomplete algorithm for doing this in linear time, without using another array is given below. Complete the algorithm by filling in the blanks. Assume all the variables are suitably declared.

```
min = n; i = 0;
while (_____) {
     temp = A[i]; j = i;
     while (_____) {
     A[j] = _____
     j= (j + k) mod n ;
     If ( j< min ) then
          min = j;
}
A[(n + i - k) mod n] = _____
i = _____
```

a. i > min; j!= (n+i)mod n; A[j + k]; temp; i + 1 ;
b. i < min; j!= (n+i)mod n; A[j + k]; temp; i + 1;
c. i > min; j!= (n+i+k)mod n; A[(j + k)]; temp; i + 1;
d. i < min; j!= (n+i-k)mod n; A[(j + k)mod n]; temp; i + 1; **[3 Marks]**

Answer: d

Explanation :

In the five blanks given in the question, the last two blanks must be temp and i+1 because all the given options for the fourth and fifth blanks have temp and i+1. Now, for the first blank, it must be I & min then the control goes out of the while loop in the initial case when i=0 and min=n So, the first blank is i < min which implies either option (B) or option (D) is correct. Assume option (B) is correct then in the bracket of while we have j!=(n+i)modn That means whenever j becomes equal to (n+i)modn then control goes out of the while loop. Now (n+i)modn=i and j is always equal to i because in line 3 of the code we are assigning the value of i to j. So, if option (B) is true control never enters the second while loop but it has to enter the second while loop to shift the nos. K places left. Hence, option (D) is correct.

Subjective

8. An array X[-15.....10, 15.....40] require 4 bytes of storage. If the beginning location is 1300 determine the location of X[5][25] by using row-major order and column-major order.

[Row-major formula (1 Mark) + Column-major formula (1 Mark) + 2 Marks]

Answer:

$X[i][j] = BA + [(i-lb1)*n_c + (j-lb2)]*C$; $n_c = ub2 - lb2 + 1 = 40 - 15 + 1 = 26$

$X[5][25] = 1300 + [(5+15)*26 + (25 - 15)]*4$

$= 1300 + [520 + 10]*4$

$= 1300 + 2120$

$= 3420$

$X[i][j] = BA + [(j-lb2)*n_r + (i-lb1)]*C$; $n_r = ub1 - lb1 + 1 = 10 + 15 + 1 = 26$

$X[5][25] = 1300 + [(25-15)*26 + (5 + 15)]*4$

$= 1300 + [260+20]*4$

$= 1300 + 1120$

$= 2420$

9. Consider the following functions

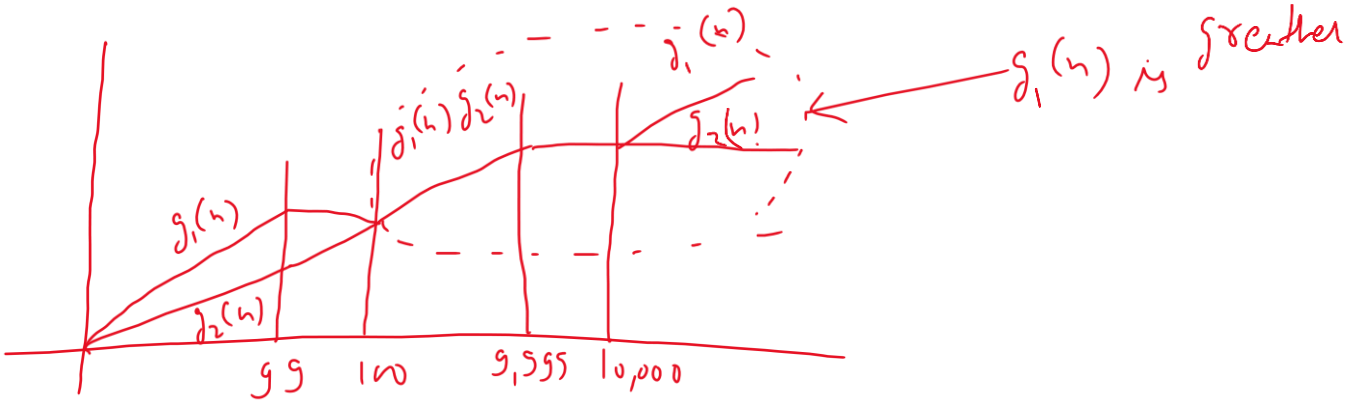
$$g_1(n) = \begin{cases} n^3 & 0 \le n < 10{,}000 \\ n^4 & n \ge 10{,}000 \end{cases}$$

$$g_2(n) = \begin{cases} n & 0 \le n < 100 \\ n^3 & n \ge 100 \end{cases}$$

Is $g_1(n) = O(g_2(n))$ or $g_1(n) = \theta(g_2(n))$ or $g_1(n) = \Omega(g_2(n))$? **[1 Marks]**

Please provide an explanation. **[2 Marks]**

Answer: $g_1(n) = \Omega(g_2(n))$



10. Consider the following sequence of operations on an empty stack.

push(54); push(52); pop(); push(55); push(62); s=pop();

Consider the following sequence of operations on an empty queue.

enqueue(21); enqueue(24); dequeue(); enqueue(28); enqueue(32); q=dequeue();
The value of s+q is _____. **[2 Marks]**

Answer: s + q = 62 + 24 = 86

11. The initial array of elements is:

21, 6, 3, 57, 13, 9, 14, 18, 2

a. Show the results of selection sort for each pass/iteration.
b. Show the results of bubble sort for each pass/iteration. **[4 Marks]**

Answer:

a. Selection sort

Pass 1: [2, 6, 3, 57, 13, 9, 14, 18, 21]

Pass 2: [2, 3, 6, 57, 13, 9, 14, 18, 21]

Pass 3: [2, 3, 6, 57, 13, 9, 14, 18, 21]

Pass 4: [2, 3, 6, 9, 13, 57, 14, 18, 21]

Pass 5: [2, 3, 6, 9, 13, 57, 14, 18, 21]

Pass 6: [2, 3, 6, 9, 13, 14, 57, 18, 21]

Pass 7: [2, 3, 6, 9, 13, 14, 18, 57, 21]

Pass 8: [2, 3, 6, 9, 13, 14, 18, 21, 57]

b. Bubble sort

Pass 1: 6, 3, 21, 13, 9, 14, 18, 2, 57

Pass 2: 3, 6, 13, 9, 14, 18, 2, 21, 57

Pass 3: 3, 6, 9, 13, 14, 2, 18, 21, 57

Pass 4: 3, 6, 9, 13, 2, 14, 18, 21, 57

Pass 5: 3, 6, 9, 2, 13, 14, 18, 21, 57
Pass 6: 3, 6, 2, 9, 13, 14, 18, 21, 57

Pass 7: 3, 2, 6, 9, 13, 14, 18, 21, 57

Pass 8: 2, 3, 6, 9, 13, 14, 18, 21, 57

12. What is the output of following function in which start is pointing to the first node of the following linked list 1->2->3->4->5->6 ?

```
void fun(struct node* start)
{
 if(start == NULL)
  return;
 printf("%d  ", start->data);

 if(start->next != NULL )
  fun(start->next->next);
 printf("%d  ", start->data);
}
```

[2 Marks]

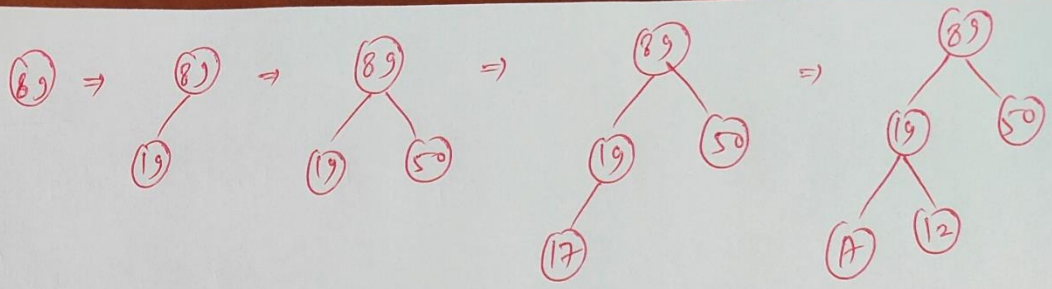
Answer: 1 3 5 5 3 1

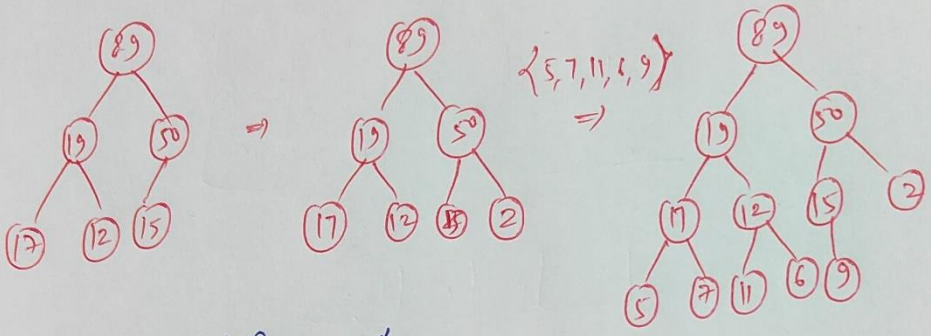
13. Consider the following array of elements.
⟨89, 19, 50, 17, 12, 15, 2, 5, 7, 11, 6, 9, 100⟩.
    a) Draw a max-heap by inserting each element one at a time.
    b) How many minimum interchanges/swaps are needed to convert it into a max-heap?
    c) Draw a min-heap by inserting each element one at a time.
    d) How many minimum interchanges/swaps are needed to convert it into a min-heap?
    e) Draw the final min-heap after deleting the root element.

[10 Marks]

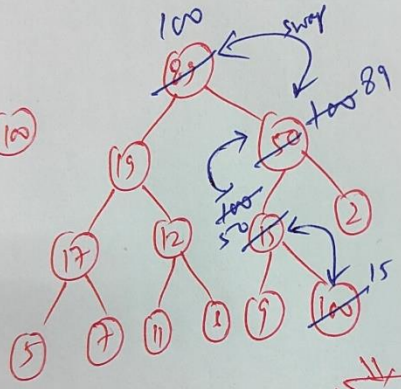
Answer:

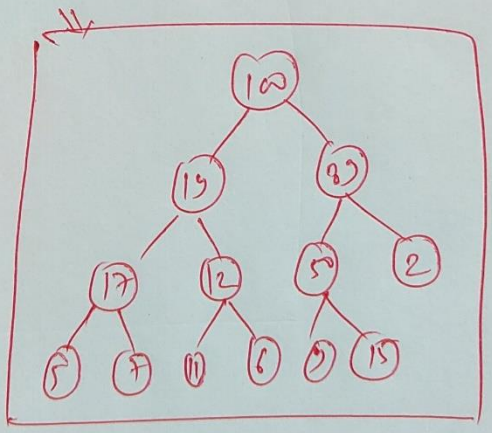
13. (a)





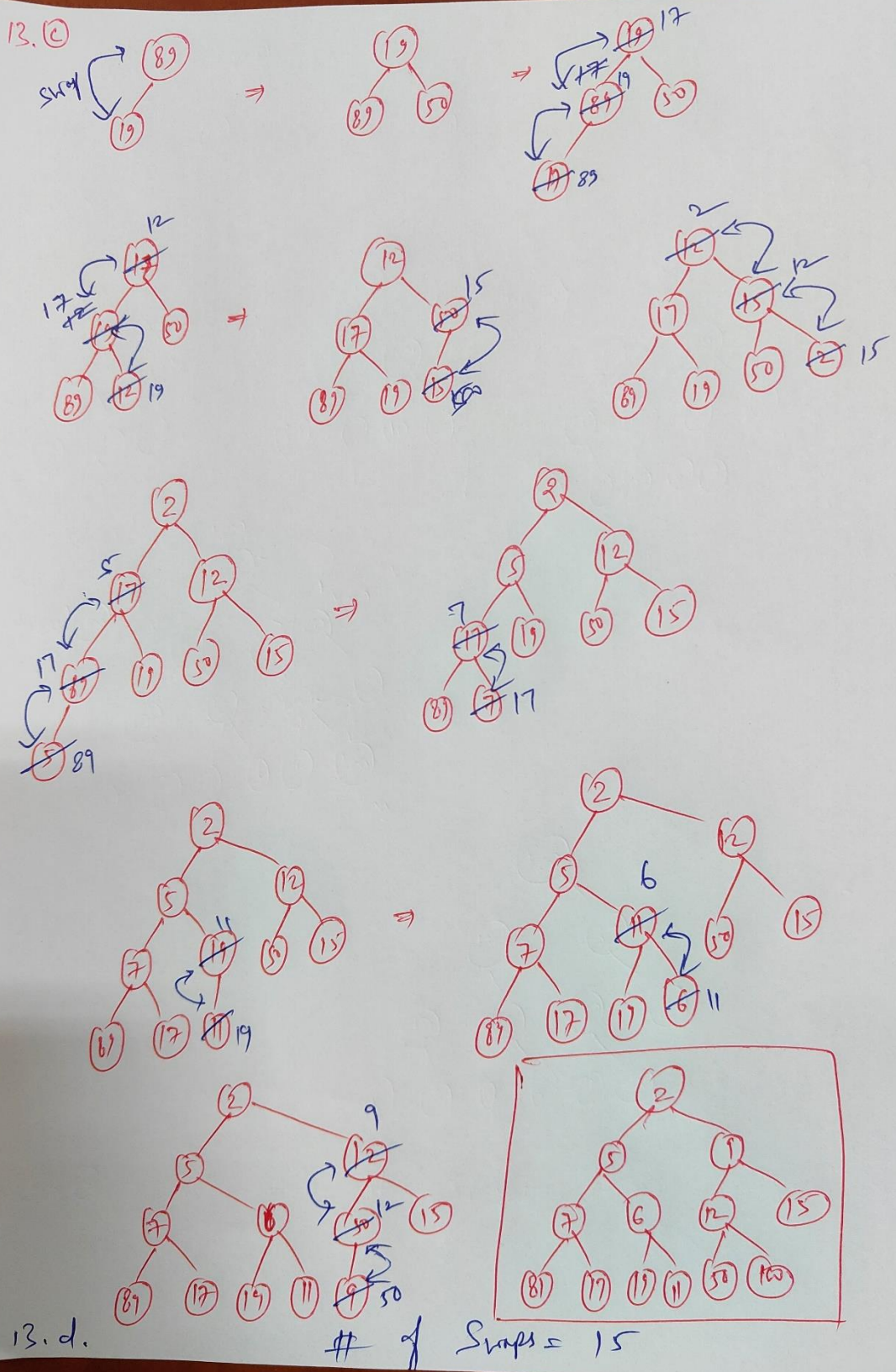
insert (100) ⇒



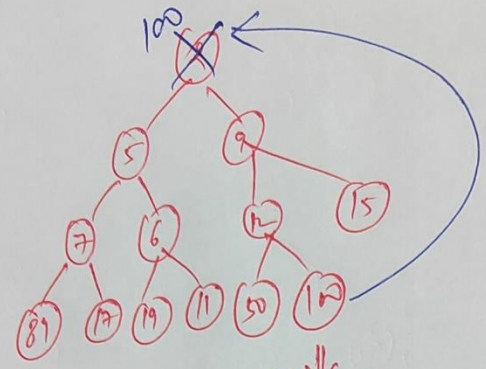


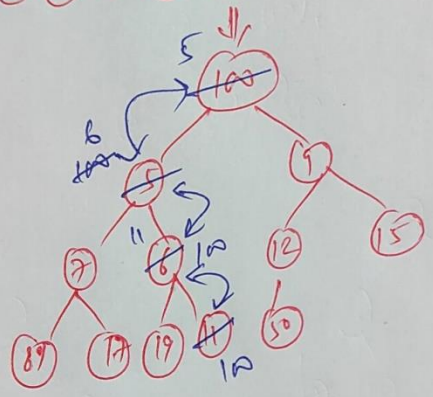
13. (b) # of swaps = 3

13. ⓒ

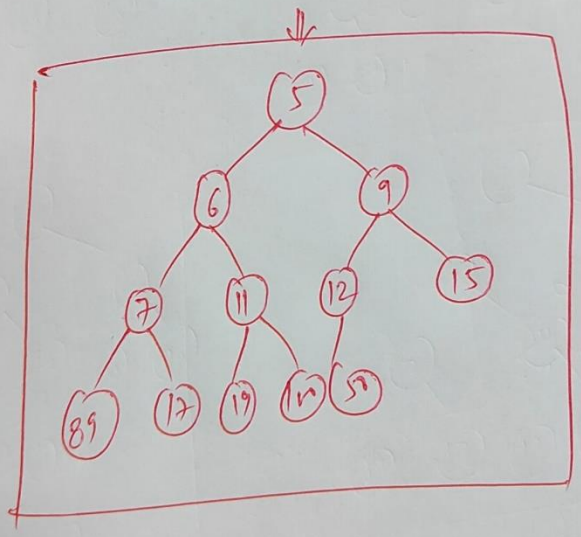


13. e.



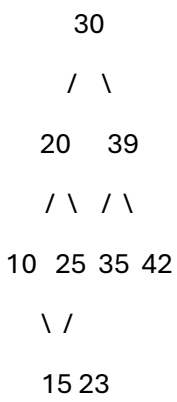


[Not Min heap]



14. Draw the final Binary Search Tree (BST) of the following elements:
    30 , 20 , 10 , 15 , 25 , 23 , 39 , 35 , 42 **[3 Marks]**

Answer:



15. Write a C program to detect the cycle in the single linked list using two pointers. **[6 Marks]**

Answer:

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

// Define the structure for a node in the linked list
struct Node {
   int data;
   struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}
```

```
// Function to detect cycle using Floyd's Cycle Detection Algorithm
int detectCycle(struct Node* head) {
    struct Node* slow = head;     // Slow pointer
    struct Node* fast = head;     // Fast pointer

    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;           // Move slow pointer by 1
        fast = fast->next->next;       // Move fast pointer by 2

        // If slow and fast pointers meet, there is a cycle
        if (slow == fast) {
            return 1;  // Cycle detected
        }
    }
    return 0;  // No cycle
}

// Function to free the linked list (optional)
void freeList(struct Node* head) {
    struct Node* current = head;
    struct Node* nextNode;

    while (current != NULL) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }
}
```

```
// Main function to demonstrate cycle detection
int main() {
    // Create linked list: 1 -> 2 -> 3 -> 4 -> 5
    struct Node* head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(3);
    head->next->next->next = createNode(4);
    head->next->next->next->next = createNode(5);

    // Create a cycle for testing: 5 -> 3
    head->next->next->next->next->next = head->next->next;

    // Detect cycle
    if (detectCycle(head)) {
        printf("Cycle detected in the linked list.\n");
    } else {
        printf("No cycle detected in the linked list.\n");
    }

    // Free the linked list (will not free the cycle)
    freeList(head);

    return 0;
}
```