Solution for 2024-2025 DSA Exam, first sitting, Code 2, solution for max 10 points.

This problem has a requirement for maximum of 6 points and a requirement for maximum 10 points. Only one of these requirements will need to be provided with a solution, the choice is yours which one you wish to attempt.

The first line of your file will contain a one line comment with either 6 or 10 written. This will be the basis on which the code will be tested. Failure to follow this will lead to the impossibility of testing your code which will result in minimum marks.

Write, in C, the necessary data structures and algorithms which do the following:

- Create a dynamically allocated linked list of nodes with 32 bit unsigned integers as data payload based on whitespace separated values from a file. The name of the file, which contains full relevant path, will be the only command line argument for your compiled program. The size of this file is not considered known.
- Sort this list so when traversed from the beginning to end it will generate an ordered sequence of those 32 bit unsigned integers, without touching the same node more than once.
- Traverse this list only one time and print the values of each node to the standard output. Values will be separated by one space. The last value will also have one space after it. This will output an ordered sequences for those values.

Constraints:

- Any form of array is off limits.
- Unnecessary code will result in points deductions.
- No unnecessary memory allocations. After the file is read and the list generated, no more memory allocations should happen.
- Memory leaks will trigger massive points deduction.
- Make sure to free the memory before exit.
- Code should be neatly divided into relevant functions.
- Code should be legible. If you think the grader will need more than a few minutes to understand what you did, please explain what that section of code does using comments
- There is a maximum of 20 seconds execution time when testing. If the code takes longer then it is considered that it does not work at all.

Constraints for the maximum 6 marks solution:

- There are no other constraints

Constraints for the maximum 10 marks solution:

- The list must be built in the order the values are read from the file
- After the list is built, the payload data of a node should not be changed
- After the list is built then its nodes can be sorted (i.e. do not sort the list while building it)

```c
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
-------------------------------------
struct Node{
        int data;
        struct Node* nextNode;
        struct Node* nextOrderedNode;
};

typedef struct Node Node;
-------------------------------------
void printList(Node* head, bool ordered){

        while(head){
                printf("%d ", head->data);
                if(ordered){
                        head=head->nextOrderedNode;
                        continue;
                }
                head=head->nextNode;
        }
}
-------------------------------------
Node* freeList(Node* head){
        Node *tmp = NULL;
        while(head){
                tmp = head;
                head = head->nextNode;
                free(tmp);
        }
return NULL;
}
-------------------------------------
```

Data structure and type definition.
Each node will have two pointers. One for the normal, unsorted list, and the other for the sorted order of node values.
The text specifies unsigned integers, so replace line 6 with [unsigned int data;].
No points were deducted for using just int.

Print list values. The ordered parameter is there to allow us to use this function for quick debug should we want to print the original, unsorted list.

Free the memory allocated for the list to avoid memory leaks. Return NULL not to leave a dangling pointer.

```
34 ----------------------------------------
35 Node* insertIntoList(Node* current, int value){
36         Node* tmp = (Node*) malloc(sizeof(Node));
37         tmp->data=value;
38         tmp->nextNode = NULL;
39         tmp->nextOrderedNode = NULL;
40
41         if(!current) return tmp;
42
43         current->nextNode = tmp;
44
45 return tmp;
46 }
47 ----------------------------------------
48 Node* initListFromFile(char *filePath){
49         FILE *fp = fopen(filePath, "r");
50         Node* first = NULL;
51         Node* current = NULL;
52
53         while(!feof(fp)){
54                 int t;
55                 fscanf(fp, "%d", &t);
56                 if(feof(fp)) break;
57
58                 current = insertIntoList(current, t);
59
60                 if(!first) first = current;
61         }
62
63         fclose(fp);
64 return first;
65 }
   ----------------------------------------
```

Create node and insert the new value into the list.
Please observe all members are initialised.

Open file.
Read from file.
Call function to insert value into list.
Repeat until end of file.
Close file.
Return pointer to the first node.

```
66 ----------------------------------------
67 Node* findMinFromList(Node* first){
68         Node* tmp = first;
69         Node* min = NULL;
70
71         while(tmp){
72                 if(!min && !tmp->nextOrderedNode){
73                         min = tmp;
74                         continue;
75                 }
76
77                 if(min && min->data > tmp->data && !tmp->nextOrderedNode) min = tmp;
78
79                 tmp = tmp->nextNode;
80         }
81 return min;
82 }
83 ----------------------------------------
84 Node* sortList(Node* first){
85         Node* orderedFirst = NULL;
86
87         orderedFirst = findMinFromList(first);
88         orderedFirst->nextOrderedNode = orderedFirst;
89
90         Node* orderedLast = orderedFirst;
91         Node* tmpNode=NULL;
92
93         while( tmpNode = findMinFromList(first)){
94
95                 orderedLast->nextOrderedNode = tmpNode;
96
97                 orderedLast = tmpNode;
98                 orderedLast->nextOrderedNode = orderedLast;
99
100        }
101
102 orderedLast->nextOrderedNode = NULL;
103
104 return orderedFirst;
105 }
    ----------------------------------------
```

Find the node with the minium value in the list.

If min is not set, set it with the current node.

If min is set, value of min node > value of current node, and nextOrderedNode field of the currrent node is not set then the current node becomes the minimum node.
Return the node with the smalles value

Sorting the list. Find the minimum value from the unsorted nodes and add it to the end of the sorted list.

We retain the first node in the ordered list (so we can know which is the first ordered node.

We use an optimisation where the last node added to the ordered list will point to itself, so it is ignored next time we search for a minimum.

We must take care to set the nextOrderedNode to NULL for the last node in the ordered list.

```
106
107 int main(int argc, char **argv){
108
109         char *inputFilePath = argv[1];
110
111         Node* head = NULL;
112         head = initListFromFile(inputFilePath);
113
114         Node* orderedFirst = NULL;
115         orderedFirst = sortList(head);
116
117         printList(orderedFirst, true);
118         head = freeList(head);
119
120 return 0;
121 }
```

Main function.
An assumption is made that the command line arguments are correct in number and values.
Assumption is made that the file exists, is not empty and has the correct format.

This problem could have been solved in multiple ways. This proposed solution is one which has the smallest possibility of messing up pointer operations.

Alternative solution would be to use a double linked list (the text specifies only that whe have to have a “linked list”, without restricting the type of linked list.

We could have also used a single linked list, and then always keep track of the previous node and the current one. This solution has the greates probability of failure due to pointer mistakes.

Anoter, less than optimal, possible solution is to use function which swaps two nodes given as arguments with a modified variant bubble sort.