CSE 3318-003 Lab Assignment 2

Due February 26

Goal:

Implementation of mergesort using linked list manipulation concepts for tables.

Requirements:

1. Write a program to read an integer input sequence and then produce an ***array of links*** giving the values in ascending order. The first line of the input file is the length of the sequence (`n`) and each of the remaining `n` lines is a non-negative integer. The first line of the output indicates the subscript of the smallest input value. Each of the remaining output lines is a triple consisting of a subscript along with the corresponding input sequence and link values.

2. Submit your C program on Canvas by 3:45 p.m. on Wednesday, February 26. One of the comment lines should include the compilation command used on OMEGA (5 point penalty for omitting this).

Getting Started:

1. Your program should read the input files from `stdin` by using Unix shell redirection (e.g. `a.out<lab1.dat`). By using redirection, it is unnecessary to explicitly open and close the input file, nor should your program prompt for a file name. You should dynamically allocate tables for storing the input keys and the table of links. Unlike the mergesort in Notes 1 and CLRS, the table of keys is ***never*** modified. So, there is no `work` array and copying to it.

2. The link values should be initialized to `-1` before the recursive mergesort begins. This corresponds to each sequence value being placed in a single-element list (at the bottom of the recursion's "tree").

3. The input sequence array and the link array may be global. Under this assumption, the following function prototype may be used:

   ```
   int mergeSort(int start,int count)
   ```

   where `start` is the first subscript for a subarray of `count` elements. The returned `int` is the subscript of the first element in the resulting sorted sublist. The last element in the sublist will have a link value of `-1`.

4. The critical part of the code is a linear-time merge of two subarrays that previously had their link values set for ordered sublists. (Be sure to understand the merging concept from pp. 3-4 of Notes 1 before proceeding.) The merge will revise the link values to give a single ordered list.

5. If an input value appears more than once, those elements should be ordered by subscripts in the final list, i.e. your sort code will be stable.

6. Consider the following input file:

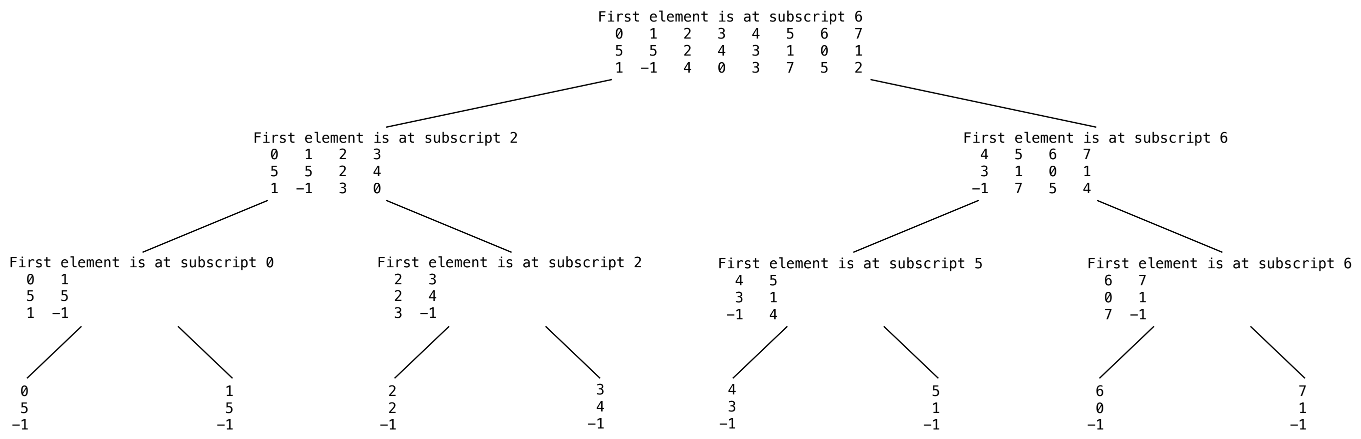
   ```
   8
   5
   5
   2
   4
   3
   1
   0
   1
   ```

   The output (as subscript/value/link triples) will be:

   ```
   First element is at subscript 6
   0 5 1
   1 5 -1
   2 2 4
   3 4 0
   4 3 3
   5 1 7
   6 0 5
   7 1 2
   ```

Notice that the input sequence ordering has not changed.

7. Your code should NEVER scan a subarray to find the minimum key.



```
First element is at subscript 6
 0   1   2   3   4   5   6   7
 5   5   2   4   3   1   0   1
 1  -1   4   0   3   7   5   2
```

```
First element is at subscript 2
 0   1   2   3
 5   5   2   4
 1  -1   3   0
```

```
First element is at subscript 6
 4   5   6   7
 3   1   0   1
-1   7   5   4
```

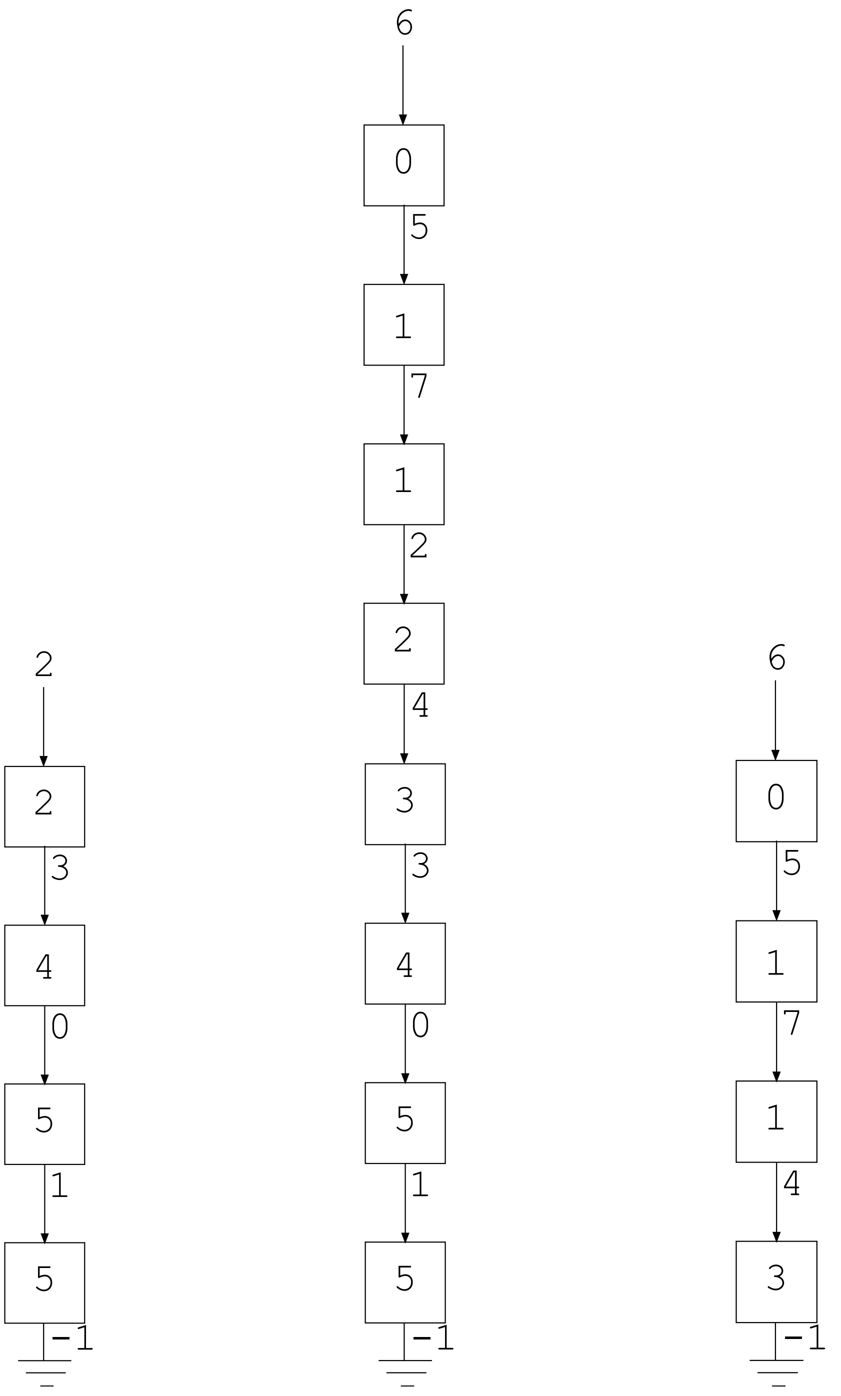
```
First element is at subscript 0
 0   1
 5   5
 1  -1
```

```
First element is at subscript 2
 2   3
 2   4
 3  -1
```

```
First element is at subscript 5
 4   5
 3   1
-1   4
```

```
First element is at subscript 6
 6   7
 0   1
 7  -1
```

```
 0     1     2     3     4     5     6     7
 5     5     2     4     3     1     0     1
-1    -1    -1    -1    -1    -1    -1    -1
```


6
0
5
1
7
1
2
2
4
3
3
4
0
5
1
5
−1
2
2
3
4
0
5
1
5
−1
6
0
5
1
7
1
4
3
−1