Task 2

Final Submission Date - 16.12.2020

Caution: Do not copy the program output messages from the job's speech. In in these cases non-ASCII characters are entered, causing the script running in autolab to terminate unexpectedly and without sufficient justification and not output the actual differences between your output and the desired one.

It is prohibited the use of global variables and the use of auxiliary arrays to perform the classification algorithms.

The code you submit must be original and not bear significant similarity to code from your fellow students.

Demanding Data Structures and Work Algorithms

In order to carry out this work, it is necessary to implement the data structures described below. These structures store integers of type int (or long int) or whatever data type you think suits you. Each structure is implemented by two files, one with a .h extension and one with a .c extension. The files with the ending .h contain the structs of each data structure and the prototypes of the functions that implement it, and in .c only the implementations of the corresponding functions. You are provided with a suitable Makefile which is used to compile the code in autolab.

The doubly linked list

In the files**dlist.c, dlist.h** implement a doubly linked list of your choice.

The binary search tree

In the files**tree.c, tree.h** implement a binary search tree.

Sorting algorithms

The implementations of the classification algorithms requested in this paper should be found in the file **sort.c** and theprototypes of these algorithms in the file **sort.h** .

Classification

Write the programsort1.c which takes one or two arguments from the command line, reads one sequence of positive integers from standard input (*stdin*) and stores it in a doubly linked list in the order it was read. Then do the following:

1. Prints the contents of the list. A blank character is printed after each integer, and a newline character is printed after the last blank.
2. Reprints newline character.
3. Sort the integers using the following algorithms. Algorithms must be applied on a list. The sorting algorithm is selected based on the first command line argument, which is an integer in the range 1 to 6 as follows:
    - 1 -> insertion sort
    - 2 -> selection sort
    - 3 -> quick sort, the rightmost element is always selected as the separation point and the left one is always sorted first of the two parts.
    - 4 -> merge sort
    - 5~~6~~ -> radix sort lsd
    - 6~~5~~ -> radix sort msd

    If the first argument is not an integer in the range 1 to 6 it prints the message " Invalid argument " followed by a newline character and the program terminates.

    If the option is 5 or 6, the program must take an extra argument specifying how many bits make up each word that makes up the integer (see note below). Allowed values are 1,2,4,8 and 16 bits. If no second argument is given or the value which given does not belong to the allowed values the program terminates by printing the message " Invalid argument " followed by a newline character.

4. For each of the options 1 to 6, the program prints information found in the subsection intermediate printouts of algorithms .
5. Prints a newline character and the contents of the sorted list. A blank character is printed after each integer, and a newline character is printed after the last blank.

Merge Sort Note: For the algorithmmerge sort you can apply the following variant for lists:

1. We recursively split the list in the middle into two distinct lists, the left and the right, and recursively apply the merge sort algorithm to these lists. If one of the two lists has one element then it is automatically sorted, while if it has two elements we sort it by comparing the two elements with each other and in case they are in reverse order we transpose them.

2. After calling the merge sort algorithm on the two lists, they are now recursively sorted.

3. Finally, we merge the two recursively sorted lists, so that the final list that results from the two has the elements of both lists correctly sorted. We choose to put the elements in one of the two lists (we don't create a new list) and destroy the other.

Radix Sort Note: For the algorithmsRadix Sort LSD and Radix Sort MSD we can think of each integer as a word consisting of letters (digits) of size X bits each. X can take values 1,2,4,8 or 16. Then we can sort any sequence of unsigned integers with the help of the above two sorting algorithms.

For example, the binary representation of the number 123456789 is 0b111010110111100110100010101. THE below table shows the different options we have to split the integer-word into digits. The first line of the table shows the positions of the digits. The integer can be represented as a) a word of 32 digits of size 1 bit each and alphabet of size 2 digits (line a of the table) , b) a word of 16 digits of size 2 bits each and alphabet of size 4 (line b of the table), c) a word of 8 digits of 4 bits each and an alphabet of size 16 digits (line c), d) a word of 4 digits of 8 bits each and an alphabet of size 256 digits (line d) or e) a word of 2 digits of 16 bits each and an alphabet of size 2$_{16}$digits (last line e).

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 a 0 0 0 0 0 1 1 1 0 1 0 1 1 0 1
1 1 1 0 0 1 1 0 1 0 0 0 1 0 1 0 1 b 0 0 0 0 0 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 1 0 1 c 0 0 0 0 0 1 1 1 0 1 0 1 1 0
1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 1 0 1 d 0 0 0 0 0 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 1 0 1 e 0 0 0 0 0 1 1 1 0 1 0 1 1
0 1 1 1 1 0 0 1 1 0 1 0 0 0 1 0 1 0 1

Algorithm intermediate prints

Each algorithm prints a number of messages that capture the algorithm's execution progress. These messages are presented in detail in this subsection.

Insertion Sort & Selection Sort

The program prints at each step (penetration) of the algorithm the alphanumeric " [S]: " , whereS the step of the algorithm with a range of values from 1 to N-1, where N is the number of items to be sorted. It then prints the contents of the list in the current step. When printing the list, a blank character is printed after each integer, and a newline character is printed after the last blank.

Insertion Sort example

To sort the elements **93 13 85 75 71 35 60 44 25 55** the algorithm prints the following:

```
[1]: 13 93 85 75 71 35 60 44 25 55
[2]: 13 85 93 75 71 35 60 44 25 55
[3]: 13 75 85 93 71 35 60 44 25 55
[4]: 13 71 75 85 93 35 60 44 25 55
[5]: 13 35 71 75 85 93 60 44 25 55
[6]: 13 35 60 71 75 85 93 44 25 55
[7]: 13 35 44 60 71 75 85 93 25 55
[8]: 13 25 35 44 60 71 75 85 93 55
[9]: 13 25 35 44 55 60 71 75 85 93
```

Example Selection Sort

To sort the elements **93 13 85 75 71 35 60 44 25 55** the algorithm prints the following:

```
[1]: 13 93 85 75 71 35 60 44 25 55
[2]: 13 25 85 75 71 35 60 44 93 55
[3]: 13 25 35 75 71 85 60 44 93 55
[4]: 13 25 35 44 71 85 60 75 93 55
[5]: 13 25 35 44 55 85 60 75 93 71
[6]: 13 25 35 44 55 60 85 75 93 71
[7]: 13 25 35 44 55 60 71 75 93 85
[8]: 13 25 35 44 55 60 71 75 93 85
[9]: 13 25 35 44 55 60 71 75 85 93
```

Quick Sort

The program prints the following at each recursive call of the algorithm:

- When entering the callback
  - prints the alphanumeric " == " , the number of times the quicksort algorithm has been called recursively. During the initial call of quicksort we assume that we are at level 0, so the specific alphanumeric is not printed.
  - prints the alphanumeric " [D >] " , whereD current level of recursion. The following is the part of the list to be sorted on this call. See print note below.

- After placing the split point in its final position and ONLY if the part to be sorted has a size greater than two elements
  - prints the alphanumeric " == " , the number of times the quicksort algorithm has been called recursively. During the initial call of quicksort we assume that we are at level 0, so the specific alphanumeric is not printed.

  - ○ prints the alphanumeric " [D -] " , whereD the current level of recursion. Next is the part of the list for which the breakpoint took its final position. See print note below.

- ● Before exiting the callback
  - ○ prints the alphanumeric " == " , the number of times the quicksort algorithm has been called recursively. During the initial call of quicksort we assume that we are at level 0, so the specific alphanumeric is not printed.
  - ○ prints the alphanumeric " [D <] " , whereD the current level of recursion. Here is the ranked portion of the list. See print note below.

Print Note: When printing the list or part of the list, after each integer a blank character is printed and after the last blank a newline character is printed.

Example of Quick Sort

To sort the elements **93 13 85 75 71 35 60 44 25 55** the algorithm prints the following:

```
[0 >] 93 13 85 75 71 35 60 44 25 55 [0 -] 25 13
44 35 55 75 60 85 93 71 == [1 >] 25 13 44 35

== [1 -] 25 13 35 44 == == [2 >]
25 13 == == [2 <] 13 25 == [1 <]
13 25 35 44 == [1 >] 75 60 85
93 71 == [1 -] 60 71 85 93 75 ==
== [2 >] 85 93 75 == == [2 -] 75
93 85 == == == [3 >] 93 85 ==
== == [3 <] 85 93 == == [2 <] 75
85 93 == [1 <] 60 71 75 85 93




[0 <] 13 25 35 44 55 60 71 75 85 93
```

Merge Sort

The program prints the following at each recursive call of the algorithm:

- ● When entering the callback
  - ○ prints the alphanumeric " == " , the number of times the mergesort algorithm has been called recursively. During the initial call to mergesort we assume that we are at level 0, so that particular alphanumeric is not printed.
  - ○ prints the alphanumeric " [S >] " , whereS the current level of recursion. Here is the uncategorized part of the list. The message is printed even if the list has one element, so it is automatically sorted. See print note below.
- ● Before exiting the callback
  - ○ prints the alphanumeric " == " , the number of times the mergesort algorithm has been called recursively. As before, during the initial call of mergesort we assume that we are at level 0, so the specific alphanumeric is not printed.
  - ○ prints the alphanumeric " [S <] " , whereS the current level of recursion. Here is the ranked portion of the list. See print note below.

Print Note: When printing the list or part of the list, after each integer a blank character is printed and after the last blank a newline character is printed.

Merge Sort example

To sort the elements **93 13 85 75 71 35 60 44 25 55** the algorithm prints the following:

```
[0 >] 93 13 85 75 71 35 60 44 25 55 == [1 >] 93
13 85 75 71
== == [2 >] 93 13 == == [2 <] 13
93 == == [2 >] 85 75 71 == ==
== [3 >] 85 == == == [3 <] 85 ==
== == [3 >] 75 71 == == == [3 <]
71 75 == == [2 <] 71 75 85 ==
[1 <] 13 71 75 85 93 = = [1 >]
35 60 44 25 55 == == [2 >] 35
60



== == [2 <] 35 60 == == [2 >] 44
25 55 == == == [3 >] 44 == ==
== [3 <] 44 == == == [ 3 >] 25
55 == == == [3 <] 25 55 == ==
[2 <] 25 44 55 == [1 <] 25 35 44
55 60


[0 <] 13 25 35 44 55 60 71 75 85 93
```

Radix Sort MSD

The program after dividing the words into separate lists (buckets) for each recursive call of the algorithm prints the following for each bucket that has at least one (non-empty) element:

- the alphanumeric " == " , the number of times the radix sort MSD algorithm has been called recursively
- the alphanumeric " [D, B] (XXXX) ", whereD the depth of the retracement, B the number of the bucket, in which one or more elements are stored in the form of a list, and XXXX the binary representation of the number B. The bucket number is obtained based on the numbering of the digit in question in the current step with based on the digit dictionary.
- the elements contained in the above bucket within a list. When printing the list, a blank character is printed after each integer, and a newline character is printed after the last blank.

Radix Sort MSD example

To sort the elements **1594525965 602652567 1318344375 805345487 805345485 272766820 1859915130 1025840939 128870542 534180483** with word length4 bits the algorithm prints the following:

```
== [1, 0] (0000) 128870542
== [1, 1] (0001) 272766820 534180483 == == [2,
0] (0000) 272766820 == == [2, 15] (1111)
534180483 == [1, 2] (0010) 602652567

== [1, 3] (0011) 805345487 805345485 1025840939
== == [2, 0] (0000) 805345487 805345485 == == == [3, 0] (0000)
805345487 805345485 == == == == [4, 0] (0000) 805345487 805345485 ==
== == == == [5, 9] (1001) 805345487 805345485 == == == == == == [6, 8]
(1000) 805345487 805345485 == == == == == == == [7, 12] (1100)
805345487 805345485 == == == == == == == == [8, 13] (1101) 805345485
== == == == == = = == == [8, 15] (1111) 805345487 == == [2, 13] (1101)
1025840939


== [1, 4] (0100) 1318344375 == [1, 5]
(0101) 1594525965 == [1, 6] (0110)
1859915130
```

Radix Sort LSD

The program at each step of the algorithm after sorting the words based on the current digit and merging the individual lists into one, prints the contents of this list as follows:

- prints the alphanumeric " [S] " , whereS the current step of the algorithm that corresponds in parallel to the sequence of the digit examined in this step (we start from step number 0).
- prints the contents of the list in the order they appear. Next to each number in parentheses is displayed in binary form the digit of the number sorted in the current step.

Radix Sort MSD example

To sort the elements **1594525965 602652567 1318344375 805345487 805345485 272766820 1859915130 1025840939 128870542 534180483** with word length4 bits the algorithm prints the following:

```
 [0] 534180483(0011) 272766820(0100) 602652567(0111) 1318344375(0111) 1859915130(1010)
1025840939(1011) 1594525965(1101) 80 5345485(1101) 128870542(1110) 805345487(1111)
 [1] 1594525965(0000) 1025840939(0010) 272766820(0110) 1859915130(0111) 534180483(1000) 128870542(1000)
602652567(1001) 131 8344375(1011) 805345485(1100) 805345487(1100)
 [2] 534180483(0010) 1318344375(0110) 1025840939(0111) 272766820(0111) 128870542(1000) 805345485(1000)
805345487(1000) 1594 525965(1001) 1859915130(1101) 602652567(1111)
 [3] 1859915130(0000) 1025840939(0001) 272766820(0001) 1318344375(0101) 128870542(0110)
1594525965(1000) 805345485(1001) 80 5345487(1001) 602652567(1011) 534180483(1111)
 [4] 805345485(0000) 805345487(0000) 272766820(0010) 1318344375(0100) 1025840939(0101) 534180483(0110)
1594525965(1010) 602 652567(1011) 1859915130(1100) 128870542(1110)
 [5] 805345485(0000) 805345487(0000) 1594525965(0000) 1025840939(0010) 272766820(0100)
1318344375(1001) 128870542(1010) 534 180483(1101) 1859915130(1101) 602652567(1110)
 [6] 805345485(0000) 805345487(0000) 272766820(0000) 602652567(0011) 128870542(0111)
1025840939(1101) 1318344375(1110) 1859 915130(1110) 1594525965(1111) 534180483(1111) [7]
 128870542(0000) 272766820(0001) 534180483(0001) 602652567(0010) 805345485(0011) 805345487(0011)
1025840939(0011) 1318344375(0100) 1594525 965(0101) 1859915130(0110)
```

Binary Trees 1

Write the programtree1.c , which reads a sequence of positive integers from standard input (*stdin*), which is the pre-order traversal of a binary search tree. The numbers in the sequence are separated by one or more spaces and the sequence ends with reading a negative number. The program constructs the binary search tree corresponding to the given penetration.

The program then prints to *stdout*the message " Enter integer: " and reads an integer from the keyboard he is looking for in the tree. If the number does not exist it prints the message "W not found! " followed by a newline character and the program terminates, whereW o integer read. Otherwise it prints a newline character and the message " Integers in level P are: " , whereP the level at which the integer given above was found (consider 0 to be the level of the root). It then prints to *stdout*the remaining numbers in the same level with the number read, printed in ascending order. A blank character is printed after each number is printed, and a newline character is printed after the last blank.

Binary Trees 2

Write the programtree2.c , which reads a random sequence of positive integers from standard input (*stdin*). The numbers in the sequence are separated by one or more spaces, and the sequence ends by reading a negative number. The sequence does NOT represent the traversal of any binary tree.

The program constructs from the elements of the sequence a binary search tree that contains all the elements of the sequence and in which the distance of the root from any of its leaves is equal to the integer part of the number $\log_2 N$ (ie $\lfloor \log_2 N \rfloor$ or floor($\log_2 N$)), where N is the number of tree nodes.

The program then prints to *stdout*herpre-order traversal of the tree he constructed. A blank character is printed after each number is printed, and a newline character is printed after the last blank.

Shipping Method

The work will be sent through the platform autolab (no VPN connection required). Follow these steps:

1. Create a folder named **hw2submit**and copy the files in there**dlist.c, dlist.h, tree.c, tree.h, sort.c, sort.h, sort1.c, tree1.c, tree2.c** .
2. Compress as tar.gz. On Linux/KDE go to the folder, right click and select **Compress -> Here (as tar.gz).**The file is created**hw2submit.tar.gz** .
3. You connect to the address **https://autolab.e-ce.uth.gr** and choose the course **ECE215-F20 (f20)** and from this work**HW2** .
4. To submit your work, click on the option **"I affirm that I have compiled with this course academic integrity policy..."**and press**submit** . Then select the file **hw1submit.tar.g** z that you created in step 2.