Data Structures

Bubble Sort

1. Consider an input list of items to be sorted.
2. Repeat the following steps until no more swaps are made in a pass:
    1. Set the "swapped" flag to false.
    2. For each item in the list from the first item to the second-to-last item:
        1. Let the "current item" be the item at the current index.
        2. Let the "next item" be the item immediately following the "current item".
        3. If the "current item" is greater than the "next item", then swap them.
        4. Set the "swapped" flag to true if a swap was performed.
    3. If the "swapped" flag remains false, exit the loop since no more swaps are needed.
3. The list is now sorted.

28/08/2023

Insertion Sort

1. Let the “sorted list” consist of one item – the first item in the input list
2. Let the “unsorted list” consist of all input items except the first one
3. Repeat the following steps until the unsorted list is empty
    1. Let the “insert item” be the first item in the unsorted list
    2. Let the “current item” be the last item in the sorted list
    3. Repeat until the “insert item” has been inserted into the sorted list
        1. If the beginning of the sorted list has been reached then
            1. Place the “insert item” at the beginning of the sorted list
            2. Note that the “insert item” has been inserted
        2. If the “current item” is less than or equal to the “insert item” then
            1. Place the “insert item” immediately after the “current item”
            2. Note that the “insert item” has been inserted
        3. If the “insert item” has not yet been inserted then
            1. If the “current Item” is not the first item in the sorted list
                - then Let the item immediately preceding the present "current item” be the new “current item”
                - else Note that the beginning of the sorted list has been reached
4. Halt (The sorted list now contains all of the input items, in order.)

Merge Sort

1. Divide the N item list into N lists each one item long
2. Repeat until only one list remains
    1. Merge the lists in pairs.
        1. Let the first list in the pair be ListA
        2. Let the second list in the pair be ListB

```
3. Let the first item in ListA be CurrentA
4. Let the first item in ListB be CurrentB
5. Let the output list, ListC, initially be empty
6. Repeat until the end of either ListA or ListB is reached
    1. If CurrentA <= CurrentB then
        1. Append CurrentA to ListC
        2. Update CurrentA to be the item after CurrentA
           Otherwise (CurrentA > CurrentB)
        3. Append CurrentB to ListC
        4. Update CurrentB to be the item after CurrentB
7. If the end of ListA has been reached then
    1. Append the remainder of ListB to the end of ListC
       Otherwise (end of ListB has been reached)
    2. Append the remainder of ListA to the end of ListC
8. Replace ListA and ListB with ListC
```

Linear Search

```
1. Let the current list item be the first item in the list.
2. Repeat the following steps until (a) you have reached the end of the list
   or (b) are explicitly told to halt.
    1. If the current item is the same as the target item then
        1. Print "Yes, found it."
        2. Halt (quit looking)
    2. Let the current item be the next item in the list
3. Print "No, did not find it."
4. Halt
```

30/08/2023

Binary Search

```
1. If the list is empty (has no items) then
    1. Print "No, did not find it"
    2. Halt (quit looking)
2. Let the current item be the item in the middle of the list
3. If the current item is the same as the target item then
    1. Print "Yes, found it"
    2. Halt (quit looking)
4. If the current item is larger than the target item then
   repeat this entire procedure on the first half of the list
    ◦ the items up to, but not including, the middle item
5. If the current item is smaller than the target item then
   repeat this entire procedure on the second half of the list
    ◦ the items following, but not including, the middle item
```

19/09/2023

Doubly Linked List

- A doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes.
- Each node contains two fields, called links, that are references to the previous and to the next node in the sequence of nodes.
- The beginning and ending nodes' previous and next links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list.
- If there is only one sentinel node, then the list is circularly linked via the sentinel node.
- It can be conceptualized as two singly linked lists formed from the same data items, but in opposite sequential orders.

evaluation of infix expression

1. if operator:
    - pop two operands
    - apply operator
    - push result
2. if operand:
    - push operand

For lab:

1. stack using arrays and linked list
2. infix to postfix
3. evaluate postfix

INfix to Postfix:

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, put it in the postfix expression.
3. Otherwise, do the following
    - If the precedence and associativity of the scanned operator are greater than the precedence and associativity of the operator in the stack [or the stack is empty or the stack contains a '('], then push it in the stack. ['^' operator is right associative and other operators like '+','–','*' and '/' are left-associative].
        - Check especially for a condition when the operator at the top of the stack and the scanned operator both are '^'. In this condition, the precedence of the scanned operator is higher due to its right associativity. So it will be pushed into the operator stack.
        - In all the other cases when the top of the operator stack is the same as the scanned operator, then pop the operator from the stack because of left associativity due to which the scanned operator has less precedence.
    - Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator.
        - After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
4. If the scanned character is a '(', push it to the stack.

5. If the scanned character is a ‘)’, pop the stack and output it until a ‘(‘ is encountered, and discard both the parenthesis.
6. Repeat steps 2-5 until the infix expression is scanned.
7. Once the scanning is over, Pop the stack and add the operators in the postfix expression until it is not empty.