For problem 1 to 5, it is recommended that you typeset your answers by \LaTeX (https://www.latex-project.org/get/) or Microsoft Word. Handwritten answers will be accepted only if clearly legible.

For problem 6, you should submit your source file (.java or .py). You should write test code in the main function to test all your implemented functions and print the output. I wrote a simple Java file and implemented two methods (getSize and display), you can start from there. If you are using python, it is easy to change the java code to python code, let me know if you have any troubles.

And please hand in the paper copy of the first five problems in class on Monday. Electronic part must be submitted via Blackboard BEFORE CLASS ON MONDAY!

1. An algorithm takes 1 ms for input size 100. How long will it take for input size 500 (assuming that low-order terms are negligible) if the running time is:
   (a) linear
   (b) quadratic
   (c) cubic
2. Use the definition of \(O, \Omega, \Theta\), to
   (a) show that \(3n + 6 = O(n)\)
   (b) show that \(3n + 6 = \Omega(n)\)
   (c) show that \(3n + 6 = \Theta(n)\)
   (d) is \(2^{n+1} = O(2^n)\)? Explain the reason.
3. Use the definition of \(O\) to prove: If \(f = O(n)\) and \(g = O(n)\), then \(f + g = O(n)\).
4. Rank the following functions: 100, \(\log n\), \(\log(n^2)\), \(\log \log n\), and \(\log^2 n\). Explain reasons for your ranking.
5. Find a big-O estimate for the running time (in terms of \(n\)) of the following function (with explanation):
   (a) int mysterySum( int n ) {
      int i, j, s=0;
      for(i=0; i < n; i++) {
         for(j=0; j < i; j++) {
            s += i*i;
         }
      }
   }
(b) Replace the inner loop in mysterySum by an $O(1)$ expression and compute the running time of the new program.

6. Implement Linked List operations. You can not use existing List APIs. Please implement the belowing operations:
   (a) getSize(ListNode head): return the size of the linked list
   (b) isNull(ListNode head): return true if the linked list is NULL; else false
   (c) display(ListNode head): print the value of each node in the linked list in order
   (d) insert(ListNode head, int n, int val): insert a new node to the nth position, the value of the new node equals to val.
   (e) remove(ListNode head, int n): remove the nth node
   (f) removeVal(ListNode head, int val): remove all the nodes that have value equal to val.
   (g) reverse(ListNode head): reverse the linked list
   (h) append(ListNode head, int val): creates a new node with the integer value to the end of the linked list
   (i) appendSortedASC(ListNode head, int val): creates a new node with the integer value and inserts it to the linked list in a location to maintain the ascending order. If the list is not already sorted, new node can be inserted anywhere.
   (j) appendSortedDESC(ListNode head, int val): creates a new node with the integer value and inserts it to the linked list in a location to maintain the descending order. If the list is not already sorted, new node can be inserted anywhere.