1)

1. Declare a pointer with enough bits for 300 integer elements.
2. Declare an int vector.
3. Using a doubly linked list class with \*head \*tail
4. int j = 0 //for keeping track of position in pointer
5. For(int i=0; i<=448; i+=2) //Use a loop to get even numbers 150-448.
   1. \*(numbersPointer + j) = numbers[i]; //For pointer: assign each number into the memory pointed to by the pointer while incrementing the memory address
   2. pushback i to vector
   3. if (head == NULL and tail == NULL) //For 1st element in doubly linked list
      1. tail->intValue = i;
      2. tail->prev = head;
      3. tail->next = NULL;
      4. head = tail;
   4. Append newly created node to tail and assign it as new tail. //For all other node
   5. j++ //increment position for pointer
6. For(int i=449; i>=151; i-=2) //Use another loop to append odd numbers 449-151 using similar method to above.
   1. j++ //Keep using j to keep track of position of pointer
   2. All other code should be almost the same

2a) Insertion Sort (ascending)

1. Input: an unsorted pointer. Output: a sorted pointer
2. if (\*(numbersPointer + 0) > \*(numbersPointer + 1)) //compare 1st pair of elements in pointer. Swaps their positions if needed.
   1. int temp; //temp value to store value of 1st element
   2. temp = \*(numbersPointer + 0)
   3. \*(numbersPointer + 0) = \*(numbersPointer + 1)
   4. \*(numbersPointer + 1) = temp
3. int j; //use for position of sorted portion
4. For(int i=2; i<300; i++) //loop through pointer
   1. j = 0; //reset position for each unsorted element
   2. While (\*(numbersPointer + i) > \*(numbersPointer + j)
      1. j++; //find position to insert
   3. Insert unsorted element into position j
   4. Shift other elements to the right

2b) Selection Sort (ascending)

1. Input: an unsorted pointer. Output: a sorted pointer
2. Starting with the first position, assign this element to a variable **X**.
3. Compare the value of **X**, with each element in the list one by one.
4. If **X**’s value is bigger, assign the new element as **X.**
5. Swap element **X** with the initial element. //use a temp integer variable
6. Repeat previous steps with the unsorted elements

3a) Selection Sort (descending)

1. Input: an unsorted vector. Output: a sorted vector
2. Starting with the first index, assign this element to a variable **X**.
3. Compare the value of **X**, with each element in the list one by one.
4. If **X**’s value is smaller, assign the new element as **X.**
5. move element **X** to the front ofthe vector
6. Repeat previous steps with the unsorted elements

3b) Bubble Sort (descending)

1. Input: an unsorted vector. Output: a sorted vector
2. Compare pairs of elements and swap them so that the bigger element comes first, starting from the first element pair.
3. Repeat step 1 for the next pair in the vector.
4. Once one run of comparisons are over, do another run of comparisons.
5. Repeat until no more comparisons are needed

4a) Insertion Sort (ascending)

1. Input: an unsorted doubly linked list. Output: a sorted doubly linked list
2. Compare the first 2 elements and sort them, putting the smaller one as the first element. (sorted)
3. Select the next unsorted element and compare it to the sorted elements.
4. Starting from the first sorted element, check to see if the selected element is less than the sorted element.
   1. If it is, then insert it before the node it was compared to.
   2. If not, then go to the next sorted element and compare them and repeat this step.
5. Repeat steps 2 and 3.

4b) Selection Sort (ascending)

1. Input: an unsorted doubly linked list. Output: a sorted doubly linked list
2. Starting with the first node, assign this element to a variable **X**.
3. Compare the value of **X**, with each element in the list one by one.
4. If **X**’s value is bigger, assign the new element as **X.**
5. Swap element **X** with the initial element.
6. Repeat previous steps with the unsorted elements

7) Linear Search

1. Input: an integer to find, a sorted data structure
2. Output: address of the integer (and index if the data structure is a vector)
3. Iterate through the elements starting from the first element, checking if the element searched for is equal to an element in the list
   1. For pointer: Iterate by incrementing the memory address; print the address
   2. For vector: Iterate by incrementing the index starting from zero; print the address and index
   3. For doubly linked list: Iterate using the next pointer; print the address of the node.

8) Binary Search

1. Input: an integer to find, a sorted data structure
2. Output: address of the integer (and index if the data structure is a vector)
3. Find the middle element
   1. For pointer: The address of the middle element would be (pointerAddress + (numberOfElements/2-1))
   2. For vector: The middle index would be (numberOfElements/2-1)
   3. For doubly linked list: Use 2 pointers to iterate through the nodes with one iterating through twice as fast.
      1. Once the faster one finishes iterating, the slower one would have found the middle
4. Compare the searched for element with the middle element
   1. If equal, print the middle element’s address (and index for vector)
   2. If less than or greater than, then binary search the elements before or after the middle value respectively.
5. Repeat steps 1 and 2 until the searched for element is found.