Q1. Find time complexity for the following two codes. Specify best case and worst case, if exist (compute time for both of them). You are not required to give asymptotic complexity, just write time expression: **(4+5+3+3=15)**

a. `i = 1;`	**1**
`    j = N;`	**1**
`    while (j > 1){`	**N(N-1) = N² - N**
`        cout << i << ' ';`	**(N-1)(N-1) = N² - 2N + 1**
`        i++;`	**(N-1)(N-1) = N² - 2N + 1**
`        if (i == N){`	**(N-1)(N-1) = N² - 2N + 1**
`            i = 1;`	**N – 1**
`            j = j - 1;`	**N – 1**
`        }`	
`    }`	**T(N) = 4N² - 5N + 3**

Best Case: if condition is always false that is all x[i] are less than equal 50

Worst Case: if condition is always true that is all x[i] are greater than 50

b.	**Best Case**	**Worst Case**
`i = 1`	**1**	**1**
`while i <= N`	**N + 1**	**N + 1**
`   if (x[i] > 50)`	**N**	**N**
`      k = 1`		**N**
`      while k <= N`		**N(lgN + 1)**
`         k = k * 2`		**NlgN**
`      end-while`		
`   }`		
`   i = i + 1`	**N**	**N**
`}`	**3N + 2**	**2NlgN + 5N + 2**

c. Write asymptotic time complexity for each of the following:

i. **2nlog(n) + 10000(log (n))^{10} + 500n + 3.5n^2 + 2n^2**: ______**n^2**______

ii. **2n + 50000logn**: ________**n** _______

iii. **2n + 100900:** ________ **n**________

d. For the code below, you need to identify the exact number of times statements **S1** and **S2** executes: **(3 Marks)**

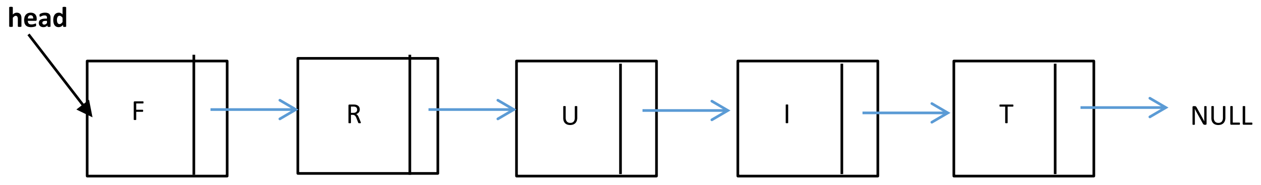
```
sum = 0;
for (i = 1; i <= n-1; i++)          ........... S1
    for (j = 1; j < i ; j += 2)
        sum++                        ........... S2
```

S1: n times

S2: $\frac{1}{4}(\mathbf{n}^2 - 2n)$= $\sum_{i=1}^{n-1} \left\lfloor \frac{i}{2} \right\rfloor = 0 + 1 + 1 + 2 + 2 + 3 + \cdots + \left(\frac{n}{2} - 1\right) = \frac{\left(\frac{n}{2}-1\right)\left(\frac{n}{2}\right)}{2}$ for even values of n

Q.2 Given the following function and a Linked List populated with specific nodes, as shown in the figure. Each node in the Linked List contains a data item of type **char** and has a pointer named **next,** which points to the next node. What would be the output of the function named **recursionP**, if called with head node? **(3 Marks)**

Linked List:



```
void recursionP(Node* head){
      if(!head)    return;
      if (head -> next)
            cout <<  head -> next -> data << ' ';
      else
            cout << head -> data << ' ';
      recursionP (head->next);
      cout <<  head->data << ' ';
}
```

Output: __**R U I T T T I U R F**__

Q3. Consider the following pseudo-code? **(3 Marks)**

```
void f(int n, char s, char d, char t){
      if (n == 1){
            cout << s << ' '  << d << '\n';
            return;
      }
      f (n - 1 , s, t, d);
      cout << n << ' ' << s << ' '  << d << '\n';
      f (n - 1 , t, d, s);
}
f (3, 'A', 'C', 'B');
```

Output:

```
A C
2 A B
C B
3 A C
B A
2 B C
A C
```

Working:

```
f (3, 'A', 'C', 'B')                          External Call
      n = 3, f (2, 'A', 'B', 'C')
            n = 2, f (1, 'A', 'C', 'B')
                  n = 1,                           A C        return to n = 2
            n = 2,                                 2 A B
            n = 2, f (1, 'C', 'B', 'A')
                  n = 1                            C B        return to n = 2
            n = 2,                                            return to n = 3
      n = 3,                                       3 A C
      n = 3, f (2, 'B', 'C', 'A')
            n = 2, f (1, 'B', 'A', 'C')
                  n = 1,                           B A        return to n = 2
            n = 2,                                 2 B C
            n = 2, f (1, 'A', 'C', 'B')
                  n = 1,                           A C        return to n = 2
            n = 2,                                            return to n = 3
      n = 3,                                                  return to main call
```

Q4 Suppose you are given a sorted array A of n distinct numbers that have been rotated k steps, where k is an unknow integer between 1 to n-1. The array is split such that the first k elements (A[0] to A[k-1]) and the remaining elements (A[k] to A[n-1]) are each sorted in increasing order, with the condition that A[n-1] < A[0]. Given this, describe or implement an approach to find the value of k in O(LogN). **(10 Marks)**

The array {9,13,16,18,19,23,28,31,37,42,0,1,2,5,7,8} has 16 elements with k =10.

```
int findK(int Array[], const int SIZE){
      int i = 0, mid;
      int j = SIZE - 1;
      while (i <= j){
            mid = (i + j) / 2;
            if (Array[mid] > Array[mid+1])
                  break;
            else if (Array[mid] > Array[0])
                  i = mid + 1;
            else
                  j = mid - 1;
      }
      return mid + 1;
}
```

Q5. Given two very long positive integers stored as ***Char*** type vectors . Write an ***add*** function to compute the sum of these integers and return the result as a new ***char*** type vector. Assume each integer is stored in either left-to-right(most significant to least significan digit) or right-to-left (least significant to most significant digit) order in the input vectors, and keep the same order in the resultant vector for consistency. **(10)**
Note 1: To secure full marks, write code-avoiding duplication.

```
vector<char> add(vector<char> &a, vector<char>& b){
    vector<char> result;
    int carry = 0;
    int i = N1 - 1, j = N2 - 1;
    int d1, d2;

    while (i >= 0 || j >= 0 || carry) {
        d1 = d2 = 0;
        if (i >= 0)     d1 = a[i];
        if (j >= 0)     d1 = b[j];
        int sum = d1 + d2 + carry;
        carry = sum / 10;
        result.push_back((sum % 10) + '0');
        if (i >= 0) --i;
        if (j >= 0) --j;
    }
    return result;
}
```

Output:
```
  23182536475839542731
 985172263142849321282
----------------------
1008354799618688864013
```

Q6. In a doubly circular header-linked list, a ***swap*** method is used to **interchange** two disconnected nodes. The method does not include any **checks** or special handling for cases where the nodes to be **swapped are adjacent** (i.e., one node is the next or previous node of the other). Analyze if the current implementation of the swap method could cause any **issues** when swapping adjacent nodes, and explain why or why not. **(10)**

Note: Give brief to the point answer, which statement can cause the problem and why?

The issue arises because, in a doubly circular header-linked list, swapping two adjacent nodes requires updating their pointers carefully to maintain the list's structure. Specifically:

The lines of code where the next and prev pointers of the adjacent nodes are directly reassigned without accounting for their adjacency can cause a problem. For example:

```
node1->next = node2->next;
node2->prev = node1->prev;
```

If node1 and node2 are adjacent, node2->next points to node1, and node1->prev points to node2. These updates will break the circular linkage and leave the list in an inconsistent state.

Reason: Adjacent nodes share pointers with each other (node1->next = node2 and node2->prev = node1). Swapping them without special handling will overwrite the shared pointers, resulting in incorrect links.

Q7. Write a brute-force method to evaluate an infix expression directly, without converting it to postfix or prefix notation. You may write either pseudocode, Python code, or C++ code. Also, discuss complexity of your code.**(10)**

```
input: "2+3*4"
output: 14
input: "(2+3)*4"
output: 20
```

```
int n = ex[.size();
int result[n];
char ops[n - 1];
int resIndex = 0, opIndex = 0;
for (int i = 0; i < n; ++i) {    // Step 1: Parse the ex[ into numbers and operators
    if (ex[[i] >= '0' && ex[[i] <= '9')
        result[resIndex++] = ex[[i] - '0';
    else if (ex[[i] == '+' || ex[[i] == '-' || ...)  // operator
        ops[opIndex++] = ex[[i];
}
for (i = 0; i < opIndex; i++) // Step 2: Handle parentheses by evaluating innermost
    if (ops[i] == '(') {
        start = end = i;
        while (end < opIndex && ops[end] != ')') end++;
        for (int j = start + 1; j < end; ++j) // Evaluate the expression within ( )
            if (ops[j] == '*') {
                result[j - 1] *= result[j];
                result[j] = 0;
                ops[j] = '+';
            }
            else if (ops[j] == '/') {
                result[j - 1] /= result[j];
                result[j] = 0;
                ops[j] = '+';
            }
        for (int j = start + 1; j < end; ++j)
            if (ops[j] == '+')
                result[start] += result[j];
            else if (ops[j] == '-')
                result[start] -= result[j];
        result[end] = result[start];
        result[start] = 0;
        ops[start] = '+';
        ops[end] = '+';
    }
for (int i = 0; i < opIndex; ++i)      // Step 3: Process all * and / operators from left to right
    if (ops[i] == '*') {
        result[i] *= result[i + 1];
        result[i + 1] = 0;
        ops[i] = '+';
    } else if (ops[i] == '/') {
        result[i] /= result[i + 1];
        result[i + 1] = 0;
        ops[i] = '+';
    }
int finalResult = result[0];           // Step 4: Process all + and - operators from left to right
for (int i = 0; i < opIndex; ++i)
    if (ops[i] == '+')
        finalResult += result[i + 1];
    else if (ops[i] == '-')
        finalResult -= result[i + 1];
```

Q8 Write an efficient `O(NlogN)` algorithm to generate a vector where each element at index i contains the count of elements in the original vector that are smaller than the element at index i. Specifically, for each element in the input vector, count the number of elements in the entire vector that are smaller, and store this count at the corresponding index in the output vector.

Example:
```
input: {3,7,1,2,4}
output: {2,0,4,3,1}
```

Note: An inefficient method will not be graded.

```
    vector<int> sortedNums = nums;//take new array to keep original
    vector<int> result(n, 0);
    sort(sortedNums.begin(), sortedNums.end()); // sort the temporary array

    // For each element in nums, find the count of smaller elements
    for (i = 0; i < n; i++) {
      // do binary search of element of original array in sorted array
      int pos = binarySearch(sortedNums, nums[i], 0, nums.size() - 1);
      // There are exactly pos (position number of elements before position)
      result[i] = pos;
    }
```

[**Bonus Question**] Consider class NNode with two pointers next1, next2. In NList nodes n1 and n2 may point to NULL or some next node on different paths like:

```
head-> 23 -> 18 -> 25 -> 36 -> 47 -> 58 -> NULL
              |            └─>  39 -> 54 -> NULL
              └> 27 -> 31 -> 43 -> NULL
                         └─>  51 -> 72 -> NULL
```

Traverse the list and give output in the format:

Output:
```
23 18
      25 36
            47  58
            39  54
      27 31
            43
            51 72
```

```
stack<pair<NNode*, int>> st;
int count = 0;
for (NNode *t = head ; t != NULL || !st.empty(); t = t->next1){
  if (t == NULL){
    p = st.top();      st.pop();
    t = p.first;
    count = p.second;
    cout << '\n';
    tabs(count);
  }
  cout << t -> val << ' ';
  count++;
  if (t->next2){
    st.push(pair(t->next2, count);
    cout << '\n';
    tabs(count);
  }
}

void tabs(int count){
  for (int i = 0 ; i < count ; i++)
    cout << '\t';
}
```