National University of Computer and Emerging Sciences
Islamabad Campus

FAST School of Computing **Fall-2023**

Question 1 [3+3.5+3.5=10 Marks]

Provide the time complexity of the following code snippet. Also provide justification for your answer. An answer without justification will get a zero score.

a.

```
void Exam1(int n)
{
int sum=0;
for (int i = 1; i <= n; i = i*2)
{
    for (int j=1;j<n/2;j+=2)
            sum++;
}
}
```

3

Time Complexity: $O(n \log_2 n)$

Justification:
inner loop runs $\boxed{n/4}$ times from $1 \rightarrow n/2$ incrementing ~~increasing~~ 2 in each iteration.
outer loop
at k steps $i = 2^{k-1}$, $i <= n \rightarrow 2^{k-1} <= n \Rightarrow \boxed{k = \log_2 n + 1}$
$O(\text{inner} \times \text{outer}) = O(n \log_2 n)$

b.

```
void Exam2(int n)
{
for (int i = 0; i <= n; i = i+4) {
    for (int j = n; j >= i; j--) {
        for (k = 100; k >= 1; k = k/2)
            cout << k;
        }
}
}
```

3.5

Time Complexity: $O(n^2 \log_2 100) \rightarrow O(n^2)$

Justification:
$O(1)$ → for k loop
$O(n)$ → for j loop
$O(n)$ → for i loop

c.

```
Void Exam3(int n)
{
int score=0;
for (int i = 1; i <= n*n; i++) {
            score++;
              for (int j = 0; j <= i; j++) {
                    if(j%2==0)   {
                     cout << score;
                     break;
                    }

              }
}
}
```

Time Complexity: ~~$O(n^4)$~~ $O(n^4)$ 2

Justification:

~~outer loop $O(n^2)$, inner loop $O(n)$~~ $O(1)$ or ~~$O(n)$~~

inner loop runs for ~~1, 2, 3, 4, 5,~~ let $k = n^2$
step different value of i

i steps (inner loop)
1 1 2
1 2
2 3
3 ⋮
4
~~k~~ ~~k~~

$$1 + 2 + 3 + 4 + \dots + k$$

$$\frac{k(k+1)}{2} = \frac{n^2(n^2+1)}{2}$$

Question 2 [15 Marks]

In the recent assignment, you were given a MAXSORT problem which could be solved using two approaches, using *ShiftRight()* method and *Swap()* method. Given an array below apply both methods on it. Show a simulation of each method by showing array at each iteration. If you need to use some variables for correct working of your approach, show the values of those variables at each iteration as well. Your task is to use both methods to sort the array in ascending order. Indicate the complexity of your approach in terms of Big-Oh and indicate which method will be better and why?

6	1	8	2	9	4

```
template <class T)
void    sortShiftRight(~~int a~~ T* arr , int size){

     int i=1 , j=0;
     for( ; i<size ; i++ ){
         T temp = arr[i];
       for( j = i ; j>0 && arr[j-1]>temp ; j-- ){
           arr[j] = arr[j-1];        // shifting element to right
       } ~~it not~~ //we can also use function ShiftRight(arr, j, i)
       arr[j]= temp;                          // here if not shifting elements
                                              //inside the loop
     }
}
```

Time complexity: $O(n^2)$

```
template <class T>
void sortSwap(T arr, int size){
   ~~bool swaps = false;~~ sorted
    bool sorted = ~~true~~ false;
    for(int i=~~0~~ 1; i< size && !sorted; i++){
       sorted = true;
       for(int j=0; j<size-i ; j++){
           if(arr[j]< arr[j+1]){
               swap(arr[j], arr[j+1]);
               sorted = false;
           }
       }
    }
}
```

Time complanity : $O(n^2)$ ✓

Question 3 [07 Marks]

Consider a C++ implementation of a singly linked list called SinglyList with the following structure:

```
#include <iostream>
using namespace std;

template <typename T>
class SinglyList {
private:
    T* data;
    int* next;
    int head_start;
    int free_start;
    int size;
    int capacity;

public:
    SinglyList(int max_size);
    ~SinglyList();
    void insert(T value);
    void remove(T value);
    void resize(int new_capacity);

    void display() {
        int current = head_start;
        cout << "Linked List: ";
        while (current != -1) {
            cout << data[current] << " -> ";
            current = next[current];
        }
        cout << "NULL\n";
    }
};
```

This implementation strategy uses two arrays, **data** and **next**, to simulate a dynamic list of elements in memory. The data array holds values like items in a list, while the **next** array tracks the relationships between these values, indicating which item comes after another. The **head_start** index points to the first item in the list, and the **free_start** index points to the first available space in the arrays to add added items. It is like having a collection of boxes (the data array), each containing an item, and a set of arrows (the next array) that tell us which box to look at next. This design lets you manage a list of items efficiently.

Your task is to complete the implementation of this **SinglyList** class by filling the blanks in the following code snippets:

0181

a. Constructor [2 marks]

Complete the constructtor for the SinglyList class. The constructor should initialize the class members and set up the linked list.

```
template <typename T>
SinglyList::SinglyList(int max_size) {
    capacity = max_size;
    size = 0;

    data = new T[max_size];
    next = new int[max_size];

    for (int i = 0; i < max_size - 1; ++i) {

        next[i] = data[i+1]______________________________; //Complete the line
    }
    next[max_size - 1] = -1;

    head_start = data[0]_______________________________; //Complete the line
    free_start = 0;
}

template <typename T>
SinglyList::~SinglyList() {
    delete[] data;
    delete[] next;
}
```

02

b. Insert an item [3 marks]

Complete the insert function for the SinglyList class. The insert function takes a value as input and insert it at the start of the list (i.e., at the head):

```
template <typename T>
void SinglyList<T>::insert(T value) {
    if (free_start == -1 || size == capacity)
    {
        cout << "Linked list is full. Cannot insert more nodes.\n";
        return;
    }

int new_node_index = __0_________________________________; //Complete the line
    free_start = next[free_start];
    data[new_node_index] = value;

     if (head_start == -1) {
         head_start = new_node_index;
         next[new_node_index] = -1;
     }
     else {
       //Add appropriate code lines here
        head_start = new_node index;
        next[new_node_index] = head_start.
     }

     size++;
}
```

c. Remove an item [2 marks]

Complete the remove function for the SinglyList class. The remove function should allow you to remove a specific value from the linked list by traversing the list to find the node with the specified value, if found, remove it:

```
template <typename T>
void SinglyList<T>::remove(T value) {
    int prev = -1;
    int current = head_start;

    while (current != -1) {
        if (data[current] == value) {
            if (prev != -1) {
                next[prev] = next[current];
            }
            else {
                head_start = next[current];
            }

            next[current] = ____-1___________________; //Complete the line

            free_start = ____________________________; //Complete the line
            size--;
            return;
        }
        prev = current;
        current = next[current];
    }
    cout << "Value not found in the linked list.\n";
}
```

0181

Question 4 [6+14=20 Marks]

a. Which sorting algorithm will have the best time complexity for sorting this array? Select one of them and provide a justification (not more than one line). Note: all sorting algorithms are optimized as per discussed in class. [6 Marks]

1.

1	2	3	4	5	6	7	8	-3	-4

a) Bubble sort
b) Selection sort
[✓ c) Insertion sort] ✓

2

Justification: Most of the part of array is already sorted, we just need to insert -3 and -4 on their right place.

2.

1	15	3	4	5	8	10	12	13	2

a) Bubble sort
[✓ b) Selection sort] ✓
c) Insertion sort

2

Justification: Only one swapping is required of 2 and 15. if we use bubble or insertion sort it will do step that aren't required. selection swaps max'= 15 with last element 2, it will be done in one iteration.

3.

32	27	25	5	9	10	15	18	22	24

[✓ a) Bubble sort] ✓
b) Selection sort
c) Insertion sort

2

Justification: Array is in random order

b. Given a linked list (`train`) and a ` Mutate ` function, answer the part (i) and (ii). [9+5 Marks]

Note: There are no syntax errors in the code; if you encounter any, please disregard them. The head pointer is pointing to the first node of linked list.

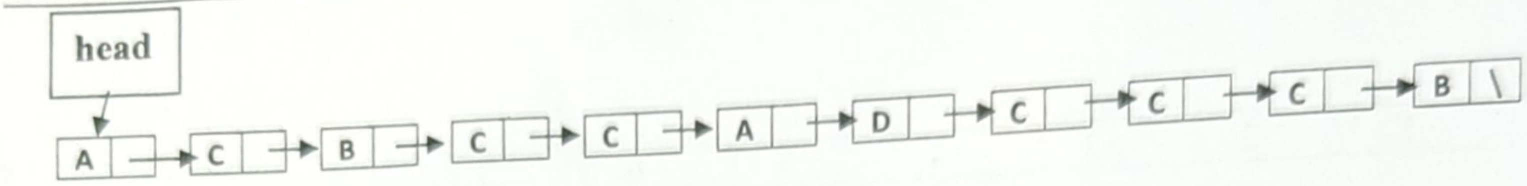
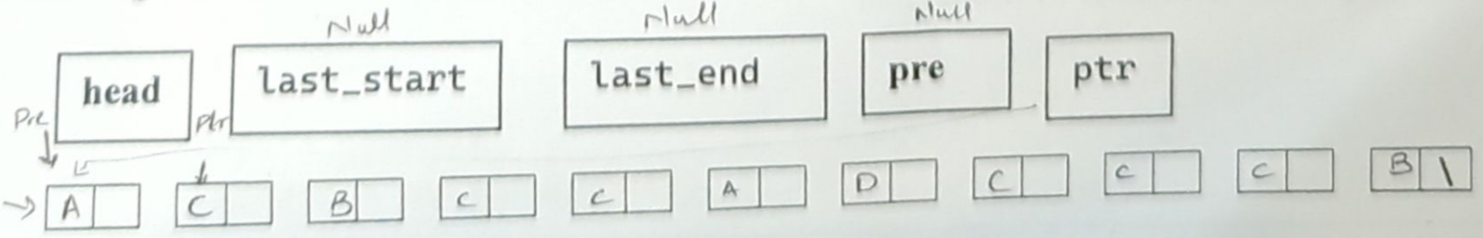


Figure 1:Train

```
int List:: Mutate (char d)
{
    Node* ptr = head;
    bool chk = false;
    Node* last_start=NULL, *last_end=NULL, *pre=NULL;
    while (ptr != NULL)
    {
                                    C
        if (ptr->data == d)
        {
            if (last_start != NULL)
            {
                Node* temp = last_start->next;
                last_start->next = last_end->next;
                pre->next = temp;
                last_end->next = ptr;
                last_start = pre;
            }
            last_start = pre;
            last_end = ptr;
            while (ptr->data == d)
            {
                last_end = ptr;
                ptr = ptr->next;
            }
        }
        pre = ptr;
        ptr = ptr->next;
    }
    Node* temp = last_start->next;
    last_start->next = last_end->next;
    pre->next = temp;
    last_end->next = ptr;
    last_start = pre;
    return 0;
}
```

For Rough Work:



i. What will be the resultant linked list if we call the ` Mutate ` function with the given linked list (`train`) with char='C'?

Mutate (‘C’);

ii. What will be the time complexity of the function in terms of Big-O notation? Please provide a justification for your provided time complexity (not more than two lines).

Time Complexity: $O(n^{\bullet})$ 5

Justification:

Inner and outer loop combinely run for n iterations. (~~num~~ ~~of time inner loop runs~~ (num of time outer runs = n- num of time inner runs)

National University of Computer and Emerging Sciences

FAST School of Computing Fall-2023 Islamabad Campus

Question 5 [10 Marks]

1. Given a linked list with the following items: 9 → 8 → 7 → 1 → 5→ NULL with head pointing to element 9. What will be the output of the following code snippet? Marks will not be awarded without complete dry run[10 marks]

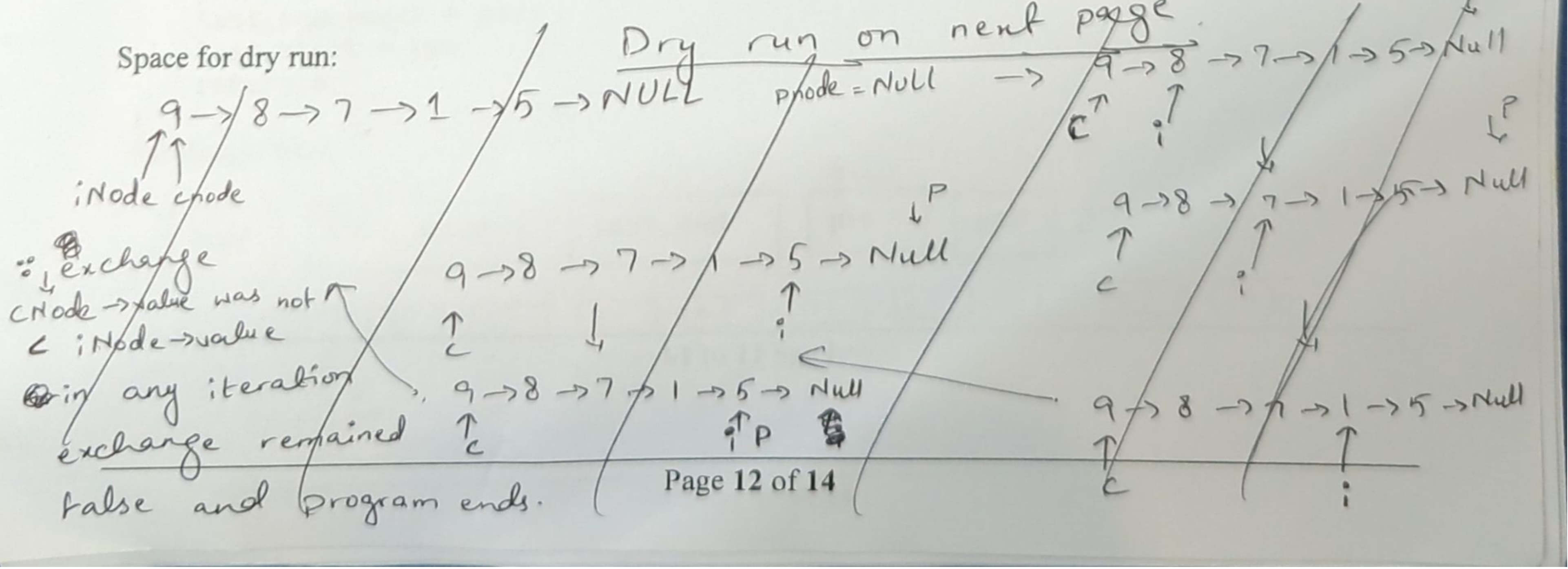
```
void Function( Node* head) {
    boolean exchanges;
    Node* iNode = Head;
Node* cNode = Null;
Node* pNode = Null;
    do {
        exchanges = false;
        while ( cNode = iNode && cNode->next != pNode ){
            iNode = iNode->next;
            if ( cNode->value < iNode->value  ) {
                int tmp = cNode->value;
                cNode->value = iNode->value;
                iNode->value = tmp;
                exchanges = true;
            } // end if statement
        } // end while statement
        pNode = iNode;
        iNode = head;
    } while (exchanges); // end do while statement
}
```

a) 7 → 8 → 9 → 1 → 5→ NULL
b) 9 → 8 → 7 → 1 → 5→ NULL
c) 8 → 7 → 1 → 5 → 9→ NULL
d) 1 → 5 → 7 → 8 → 9→ NULL
[✓ e) 9 → 8 → 7 → 5 → 1→ NULL]
f) None of above

10

Space for dry run:

Dry run on next page



∴ exchange ↓ cNode->value was not < iNode->value in any iteration exchange remained false and program ends.