Data Structures and Algorithms LAB – Spring 2026

(BS-IT-F24 Morning & Afternoon)

Lab # 4

Instructions:

- You must complete all tasks individually. Absolutely NO collaboration is allowed. Any traces of plagiarism/cheating or usage of AI tools will result in an **"F"** grade in this course and lab.
- Attempt the following tasks exactly **in the given order**.
- **Indent** your code properly.
- Use meaningful variable and function names. Use the **camelCase** notation.
- Use meaningful prompt lines/labels for all input/output.
- Make sure that there are **NO** ***dangling pointers*** or ***memory leaks*** in your programs.

Task # 1

Write a global C++ function to determine and return the **k**th largest element of an **unsorted** array containing **n** integers. The prototype of your function should be:

```
bool findKthLargest (const int* arr, int n, int k, int& val)
```

In the above prototype, **`arr`** is an *unsorted* array containing **`n`** integers from which we want to find the **k**th largest element. The last (reference) parameter (**`val`**) will be used to return the value of the **k**th largest element). Note that:

- Valid range of **k** is $\mathbf{1} \leq \boldsymbol{k} \leq \boldsymbol{n}$. If **k** is invalid, then this function should simply return **false**. Otherwise, it should determine the **k**th largest value from **`arr`**, store it in the reference parameter **`val`**, and return **true**.
- You can assume that all elements of the array **`arr`** are **unique** (i.e. there are no duplicates).
- In your function, you are NOT allowed to modify the contents of the array **`arr`**.
- The **Space complexity** of your function should be $\boldsymbol{O(1)}$ i.e. you are NOT allowed to declare/use any other array in your implementation.

Also determine the **(1)** best-case step count, **(2)** best-case time complexity (in Big Oh notation), **(3)** worst-case step count, and **(4)** worst-case time complexity (in Big Oh notation) of your implemented function.

Task # 2

When a communications site transmits a message through a packet-switching network, it does not send the message as a continuous stream of data. Instead, it divides the message into pieces called *packets*. These packets are sent through the network to a receiving site, which reassembles the message. Packets may be transmitted to the receiving site along different paths. As a result, they are likely to arrive out of sequence. In order for the receiving site to reassemble the message correctly, each packet must include the relative position of the packet within the message. For example, if we break the message "**`COMPUTER PROGRAM`**" into packets which are five characters long and preface each packet with a number indicating the packet's position in the message, the result is the following set of packets:

```
1 COMPU
2 TER P
3 ROGRA
4 M
```

No matter in what order these packets arrive, a receiving site can correctly reassemble the message by placing the packets in ascending order based on their position numbers.

You are required to write a C++ program that reassembles the packets contained in a **text file** and extracts the message stored in these packets. Assume that each packet in the input file contains a **position number** and **five characters** from the message. Your program should be based upon the following declarations:

```
const int PACKET_SIZE = 5;  // Number of characters in a packet

struct Packet {
   int position;               // Packet's position within the message
   char body[PACKET_SIZE+1];   // Characters contained in the packet including NULL terminator
};
```

The format of the input file is as follows. The first line indicates the number of packets present in the file. After that those many packets are present in the file (one on each line), in a random order:

```
8
4 ear W
1 Eleme
2 ntary
7 menta
8 ry.
5 atson
3  my d
6 ! ele
```

Your class declaration should look like this:

```
class PacketManager {
  private:
      Packet* packets;     // Array which will be used to store the packets read from the file
      int numPackets;      // Number of packets in the above array
      char* message;       // The message that was extracted from the packets
      int msgLength;       // Length of the message
};
```

In the main function, your program should take the name of the input file from the user. If the input file is not found/opened, your program should display an appropriate error message, and ask the user to re-enter the file name. This process should be repeated until the user enters a valid file name. After that, your program should create a `PacketManager` object by passing the input file handle into it.

The `PacketManager` class should have the following member functions:

2.1 `PacketManager (ifstream& fin);`

Overloaded constructor which receives the file handle (**`fin`**) of the input file which was opened in the **main** function. It will read the number of packets from the input file, and use it to initialize the member variable **`numPackets`** and dynamically allocate an array through the pointer **`packets`**. After that, it will read the packets from the file and store them in the array **`packets`**. The constructor should initialize the pointer **`message`** to NULL, and the member variable **`msgLength`** to 0.

2.2 `~PacketManager ();`

Destructor will deallocate all the dynamically allocated memory (if any).

2.3 `void displayPackets () const;`

This function should display all the packets present in the array **`packets`** on screen.

2.4 `void sortPacketsBubbleSort ();`

This **private** member function should **sort** the array **`packets`** in increasing order (according to their **`position`**) using **Bubble Sort**. This function should NOT display anything on screen.

2.5 `void sortPacketsSelectionSort ()`

This **private** member function should **sort** the array **`packets`** in increasing order (according to their **`position`**) using **Selection Sort**. This function should NOT display anything on screen.

2.6 `void extractMessage ();`

This function extracts the message from the packets and stores the message in the char array **`message`** (see the data members of class **`PacketManager`**). This function will work as follows:

- First of all, this function will ask the user to for the choice of Sorting algorithm to be used. The user should be asked to enter **`'B'`** or **`'b'`** for **Bubble sort**, and **`'S'`** or **`'s'`** for **Selection sort** (see the Sample run shown below). Then, this function should call the appropriate function to sort the array of packets.
- Then, this function should determine the exact message length (**`msgLength`**) and dynamically allocate the array **`message`** accordingly.
- After that, this function should store the extracted message in the array **`message`**.

2.7 `void displayMessage () const;`

If the message had already been extracted from the packets (using the **`extractMessage()`** function), then this function would display the contents of array **`message`** on screen. Otherwise, this function should display an appropriate error message indicating that the message has not yet been extracted.

Also write a driver program which illustrates the usage and working of all the member functions of the **`PacketManager`** class. A sample run of your program might look like as shown on the next page (Note: Text shown in **red** is entered by the user).

```
Enter the name of the input file: input1.txt
ERROR: The file "input1.txt" does not exist!!

Re-enter the input file name: input2.txt

Packets originally read from the file are:
4 ear W
1 Eleme
2 ntary
7 menta
8 ry.
5 atson
3  my d
6 ! ele


Which sorting algorithm do you want to use?
      Enter 'B' or 'b' for Bubble sort
      Enter 'S' or 's' for Selection sort
Enter your choice: B



The order of packets after extracting the message:
1 Eleme
2 ntary
3  my d
4 ear W
5 atson
6 ! ele
7 menta
8 ry.

The extracted message is:
Elementary my dear Watson! elementary.
```

Output generated by the **displayPackets** function

Output generated by the **extractMessage** function

Output generated by the **displayPackets** function

Output generated by the **displayMessage** function

Task # 3

Modify the program, that you implemented above in **Task # 2**, to include the functionality of creating packets from a message entered by the user and store these packets in a text file specified by the user.

VERY IMPORTANT

In the next Lab, you will need some or all of the functions from Today's Lab. So, make sure that you have the working implementation of **ALL** the functions of Today's Lab, when you come to the next Lab.