|  |
| --- |
| **Data Structures and Algorithms**  ***Section*: BSCE2021 Assignment # 8 *Total marks*: 100**  ***Name*** : ***\_\_NIMRA MAQBOOl\_\_ Roll number* : \_bsce21012\_\_\_** |

***Submission:***

• *Email instructor or TA if there are any questions. You cannot look at others’ solutions or use others’ solutions, however, you can discuss it with each other. Plagiarism will be dealt with according to the course policy.*

*• Submission after due time will not be accepted.*

**There should be a Report explaining your code and highlighting the results. Follow this naming convention for your report RollNumber\_Assignment#.pdf e.g BSCE21000\_Assignment7.pdf.**

**TASK:**

Implement the following Sorts using Linked list

Insertion sort

Merge sort

**Do not swap data. swap nodes**

Program should be menu driven. the program should continue to run until the exit option is selected. Handle all corner cases.

|  |
| --- |
| **FUNCTION.H:**  #include <iostream>  using namespace std;  class Node { //made a class of node public:  int data; //declared data in public  Node \*nextPtr;   Node(int val) {  data = val; //made a constructor to set values  nextPtr = NULL;  }   void setNextPtr(Node \*n) {  nextPtr = n; //set the next ptr  }   Node \*getNextPtr() {  return nextPtr; //get the value of next ptr  } };  class linkList { //making 1 other class of linklist public:  Node \*tail;  Node \*head; //made some pointers  linkList() {  head = nullptr; //declared them to zero  tail = nullptr;  }   void takingInput(int value) {  cout << "IF YOU WANT TO STOP ADDING VALUES THEN ENTER 0." << endl;  do {  cout << "ENTER VALUE = ";  cin >> value;  if(value<0){  cout<<"ENTER POSITIVE NUMBER."<<endl;  break;  }  if (value == 0) {  cout << "YOU CHOOSE TO BREAK." << endl;  break;  } else {  Node \*temp = new Node(value); //declaring  temp->data = value;  temp->nextPtr = nullptr; //initializing the next ptr in the next of the new initialized node  if (head == nullptr) {  head = temp; //if the head is null then store the temp in head  } else {  Node \*temp1 = head; //else make a new node  while (temp1->nextPtr != nullptr) //iterate it till the node is not null  temp1 = temp1->nextPtr; //store the temp to next ptr address  temp1->nextPtr = temp; //store pointer to the last one  }  }  } while (value != 0);   }   void display() {  Node \*temp = head; //declaring  while (temp != nullptr) {  cout << temp->data << "\t"; //displaying the data  temp = temp->nextPtr; //storing the next address  cout << " "; //displaying space  }  }   void insertionSort(Node \*temp1) {  Node \*temp = temp1; //making a new temp and storing the temp1  while (temp != nullptr) { //transversing the temp  Node \*next = temp->nextPtr; //storing the node to temp of next  helperFunction(temp); //calling the helper function in which nodes are swapped  temp = next;  }  head = tail; //making the head, tail  }   void helperFunction(Node \*newNode) {  if (tail == nullptr || tail->data >= newNode->data) { //checking if the tail is null and then checking the data is greater than node  newNode->nextPtr = tail; //making the new node tail  tail = newNode; //making the tail new node  } else {  Node \*temp = tail; //new node temp as tail  while (temp->nextPtr != nullptr //checking if the next ptr is not null and next of temp data is less than data in new node  && temp->nextPtr->data < newNode->data) {  temp = temp->nextPtr; //storing the next temp address  }  newNode->nextPtr = temp->nextPtr; //making the new node temp of next  temp->nextPtr = newNode; //and temp of next as new node  }  }  Node \* midPoint(Node \*head)  {  Node\* low = head; //making the head as low  Node\* high = head->nextPtr; //making high as next of head  while (high != nullptr && high->nextPtr != nullptr) //checking that if high and next of high is null  {  low = low->nextPtr; //making low as low of next  high = (high->nextPtr)->nextPtr; //and high as next of next  }  return low; //returning low  }  Node \* merge(Node \*low, Node \*high) //making a merge function  {  Node \*temp = new Node(0); //making a dummy node  Node \*temp1 = temp;  while (low != nullptr && high != nullptr)  {  if (low->data <= high->data)  { //checking if low is less than swapping  temp1-> nextPtr = low;  low = low->nextPtr;  temp1 = temp1-> nextPtr;  }  else if (high->data < low->data)  { //checking if high is less than low then swap  temp1->nextPtr = high;  high = high->nextPtr;  temp1 = temp1->nextPtr;  }  }  while (low != nullptr)  {  temp1->nextPtr = low;  low = low->nextPtr; //storing the rest of the list  temp1 = temp1->nextPtr;  }  while (high != nullptr)  {  temp1->nextPtr = high;  high = high->nextPtr; //store the rest of the list  temp1 = temp1->nextPtr;  }  return temp->nextPtr;  }   Node \* mergeSort(Node \*temp)  {  if (temp -> nextPtr == nullptr)  {  return temp; //if list have only 1 element then return  }  Node \*mid = midPoint(temp); //finding mid point  Node \*start\_of\_right = mid->nextPtr;  mid->nextPtr = nullptr;  Node \*left = mergeSort(temp); //merging  Node \*right = mergeSort(start\_of\_right);  Node \*new\_head = merge(left, right); //merging 2 list   return new\_head;  } };  **MAIN.CPP:**  #include <iostream> #include "Functions.h"  using namespace std;  // int main() {  Node n(5); //making object  n.setNextPtr(0);  n.getNextPtr();  linkList l; //making object  int opt;  cout<<endl;  cout << "\nENTER OPTION." << endl;  cout << "1.INSERTION SORT." << endl;  cout << "2.MERGE SORT." << endl;  cout << "3.EXIT." << endl; //taking input  cin >> opt;  if(opt==1){  int value;  l.takingInput(value);  l.display();  cout << endl;  l.insertionSort(l.head);  l.display();  cout<<endl;  }  if(opt==2){  int value;  l.takingInput(value);  l.display();  cout << endl;  l.mergeSort(l.head);  l.display();  cout<<endl;  }  if(opt==3){  cout<<"YOU CHOOSE TO END."<<endl;  exit(3);  }  return 0; } |