BÀI TẬP 3
CHUYÊN ĐỀ TỔ CHỨC DỮ LIỆU

Họ và tên: Bùi phúc kiển
MSSV: 20880034

Câu 1. Giả sử cấu trúc một nút của danh sách liên kết chứa số nguyên được khai báo như sau:

```
struct NODE
{
    int data;
    NODE *next;
};
```

a) Hãy viết hàm void reverse1(NODE* &head) để đảo ngược thứ tự các phần tử trong danh sách liên kết có nút đầu trỏ bởi head bằng kĩ thuật lặp.

b) Hãy viết hàm void reverse2(NODE* &head) để đảo ngược thứ tự các phần tử trong danh sách liên kết có nút đầu trỏ bởi head bằng kĩ thuật đệ qui.

c) So sánh thời gian chạy của hai hàm trên.

Bài làm:

a) Hãy viết hàm void reverse1(NODE* &head) để đảo ngược thứ tự các phần tử trong danh sách liên kết có nút đầu trỏ bởi head bằng kĩ thuật lặp.

```
void reverse1(NODE* &head)
{
    //Khởi tạo các NODE con trỏ current, prev, và next
    NODE* current = head;
    NODE *prev = NULL, *next = NULL;

    while (current != NULL)
    {
        // Lưu lại NODE next
        next = current->next;
        // Đảo pointer của con trỏ hiện tại
        current->next = prev;
        // Di chuyển các con trỏ sang vị trí tiếp theo
        prev = current;
        current = next;
    }
    //Khi current = NULL, gán lại head
    head = prev;
}
```

b) Hãy viết hàm void reverse2(NODE* &head) để đảo ngược thứ tự các phần tử trong danh sách liên kết có nút đầu trỏ bởi head bằng kĩ thuật đệ qui.

```
NODE* reverseUtil(NODE* &head, NODE* node)
{
    //Nếu node bằng NULL thì return NULL
    if (node == NULL)
        return NULL;
    //Nếu node là node cuối của danh sách return node
    if (node->next == NULL) {
        head = node;
        return node;
    }
    //Đệ qui - đảo ngược các phần tử của danh sách từ node->next trở đi.
    //Trả về node cuối danh sách sau khi đảo. Gọi đó là node1.
    NODE* node1 = reverseUtil(head, node->next);
    //Điều chỉnh liên kết của node1 và node.
    node1->next = node;
    node->next = NULL;
```

```
    return node;
}
/* reverse2 */
void reverse2(NODE* &head)
{
    //Gọi hàm reverseUtil với hai tham số truyền vào là head và head
    reverseUtil(head, head);
}
```

c) So sánh thời gian chạy của hai hàm trên.

Gọi n là chiều dài của danh sách.

Với hàm reverse1

`//Khởi tạo các NODE con trỏ current, prev, và next`
`NODE* current = head;`
`NODE *prev = NULL, *next = NULL;`
	Thực hiện 3 phép gán.

`while (current != NULL)`
`{`
`// Lưu lại NODE next`
`next = current->next;`
`// Đảo pointer của con trỏ hiện tại`
`current->next = prev;`
`// Di chuyển các con trỏ sang vị trí tiếp theo`
`prev = current;`
`current = next;`
`}`
	Mỗi bước lặp thực hiện 4 phép gán nên ở bước này có 4n phép gán.

`//Khi current = NULL, gán lại head` `head = prev;`	Một phép gán sau khi thoát khỏi vòng lặp.
Tổng thời gian chạy	4n + 4

Với hàm reverse2

Gọi T(n) là thời gian chạy.

`//Nếu node bằng NULL thì return NULL`
`if (node == NULL)`
`return NULL;`
`//Nếu node là node cuối của danh sách return node`
`if (node->next == NULL) {`
`head = node;`
`return node;`
`}`
	Hai phép so sánh

`NODE* node1 = reverseUtil(head, node->next);`	T(n-1)

`//Điều chỉnh liên kết của node1 và node.`
`node1->next = node;`
`node->next = NULL;`
	Hai phép gán

Nên ta có T(n) = 4 + T(n-1) = 4.2 + T(n-2) + … = 4(n-1) + T(1) = 4n

Hai hàm có time complexity như nhau nên thời gian chạy như nhau.

Câu 2. Giả sử cấu trúc một nút của cây nhị phân chứa số nguyên được khai báo như sau:

```
struct NODE
{
    int data;
    NODE *left, *right;
};
```

Hãy viết hàm NODE* minPositive(NODE *root) để tìm nút mang giá trị nhỏ nhất trong số các nút mang giá trị dương trên cây nhị phân có nút gốc trỏ bởi root. Hàm này trả về NULL nếu trên cây không có nút nào mang giá trị dương.

Bài làm:

```
#include <limits.h>
```

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

class Node {
public:
    int data;
    Node *left, *right;
    //Hàm tạo
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right = NULL;
    }
};
```

```cpp
//Duyệt cây bằng đệ qui để tìm ra node có số dương nhỏ nhất
Node* minPositiveUtil(Node *root)
{
    //Tạo một node có data bằng INT_MAX
    Node *minPos = new Node(INT_MAX);
    //Nếu root bằng NULL thì trả về minPos
    if(root == NULL)
        return minPos;
    //Lấy ra node có số dương nhỏ nhất của các cây con trái và cây con phải
    Node *minPosLeft = minPositiveUtil(root->left);
    Node *minPosRight = minPositiveUtil(root->right);
    //So sánh data của root với data của minPos
    if(root->data > 0 && root->data < minPos->data)
        minPos = root;
    //So sánh data của minPosLeft với data của minPos
    if(minPosLeft->data > 0 && minPosLeft->data < minPos->data)
        minPos = minPosLeft;
    //So sánh data của minPosRight với data của minPos
    if(minPosRight->data > 0 && minPosRight->data < minPos->data)
        minPos = minPosRight;
    return minPos;
}
```

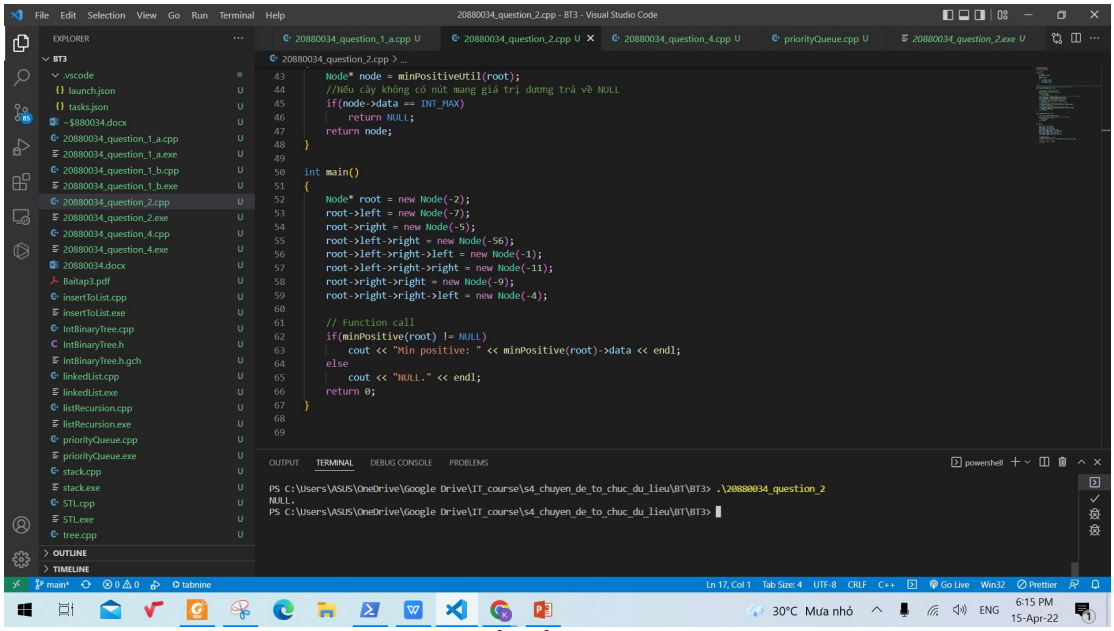
```cpp
Node* minPositive(Node* root)
{
    Node* node = minPositiveUtil(root);
    //Nếu cây không có nút mang giá trị dương trả về NULL
    if(node->data == INT_MAX)
        return NULL;
    return node;
}
```

```cpp
int main()
{
    Node* root = new Node(-2);
    root->left = new Node(-7);
    root->right = new Node(-5);
    root->left->right = new Node(-56);
    root->left->right->left = new Node(-1);
    root->left->right->right = new Node(-11);
    root->right->right = new Node(-9);
    root->right->right->left = new Node(-4);
```

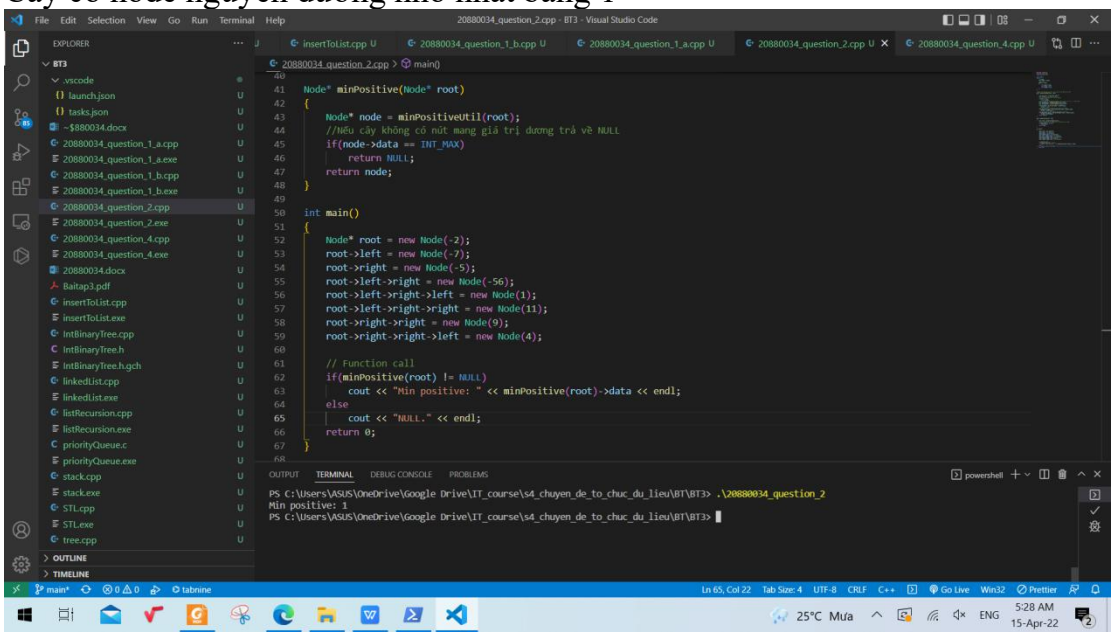
```cpp
    // Function call
    if(minPositive(root) != NULL)
        cout << "Số nguyên dương nhỏ nhất trong các node là: " << minPositive(root)->data << endl;
    else
        cout << "Cây không có node chứa số nguyên dương." << endl;
    return 0;

}
```

Chạy chương trình:
Cây không có node dương -> trả về NULL



Cây có node nguyên dương nhỏ nhất bằng 1



Bài 3. Cây biểu thức số học.

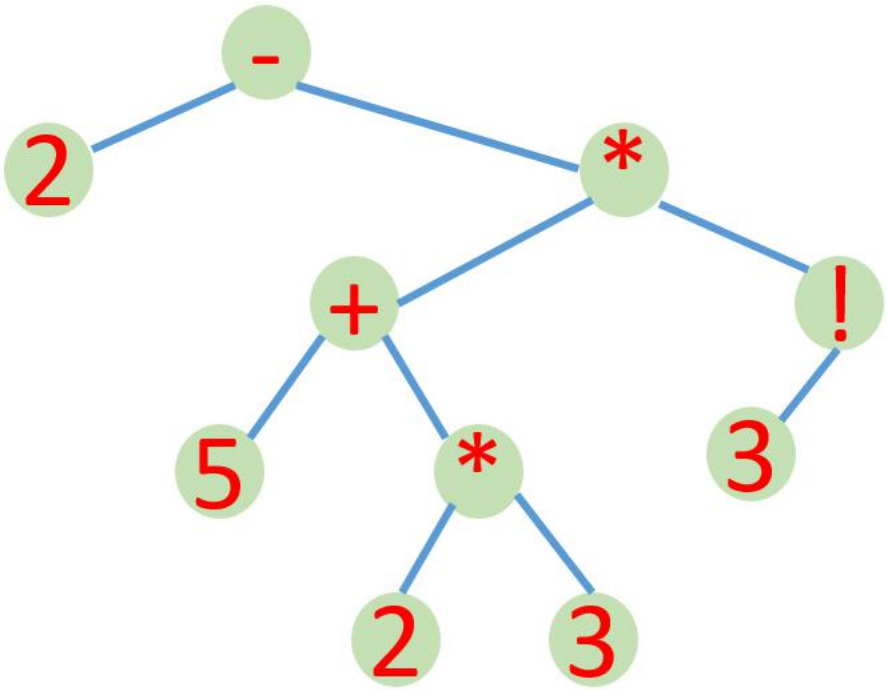
a) Vẽ cây biểu thức số học của biểu thức `2 - (5 + 2 * 3) * 3!`

b) Duyệt trước cây (a) để in ra biểu thức dạng tiền tố.

c) Duyệt sau cây (a) để in ra biểu thức dạng hậu tố.

Bài làm:

a) Vẽ cây biểu thức số học của biểu thức `2 - (5 + 2 * 3) * 3!`



b) Duyệt trước cây (a) để in ra biểu thức dạng tiền tố.
Biểu thức dạng tiền tố của cây (a): **2 - * + 5 * 2 3 ! 3**
c) Duyệt sau cây (a) để in ra biểu thức dạng hậu tố.
Biểu thức dạng hậu tố của cây (a): **2 5 2 3 * + 3 ! * -**

Bài 4. Mô hình dữ liệu *hàng đợi ưu tiên* là mô hình dữ liệu cho phép quản lý một tập các đối tượng theo độ ưu tiên. Độ ưu tiên của mỗi đối tượng là một con số nguyên. Một hàng đợi ưu tiên có hai thao tác đi kèm:
- *Insert*: chèn một đối tượng có độ ưu tiên được cho vào hàng đợi.
- *RemoveMax*: Gỡ đối tượng có độ ưu tiên lớn nhất ra khỏi hàng đợi và trả về đối tượng đó.
a) Hãy đưa ra cấu trúc dữ liệu phù hợp và cài đặt cụ thể một hàng đợi ưu tiên quản lý các số nguyên.
b) Từ cấu trúc dữ liệu đã xây dựng ở câu (a) hãy viết hàm sắp xếp tăng dần một dãy số nguyên.
Bài làm:
a) Hãy đưa ra cấu trúc dữ liệu phù hợp và cài đặt cụ thể một hàng đợi ưu tiên quản lý các số nguyên.
Có thể dùng danh sách liên kết hoặc cây nhị phân tìm kiếm để cài đặt hàng đợi ưu tiên. Trong bài này em cài đặt hàng đợi ưu tiên sử dụng cây nhị phân tìm kiếm với các thao tác:
- *Insert*: Chèn một đối tượng có độ ưu tiên được cho vào hàng đợi sử dụng đệ qui.
- *RemoveMax*: Gỡ đối tượng có độ ưu tiên lớn nhất - phần tử dưới cùng bên phải (right most) ra khỏi hàng đợi bằng cách thay đổi các liên kết và trả về phần tử đó.
Cài đặt một hàng đợi ưu tiên quản lý các số nguyên

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct NODE
{
  int data;
  int priority;
  NODE *left, *right;
  //Hàm Tạo
  NODE(int data, int priority)
  {
    this->data = data;
    this->priority = priority;
    this->left = NULL;
    this->right = NULL;
  }
};
```

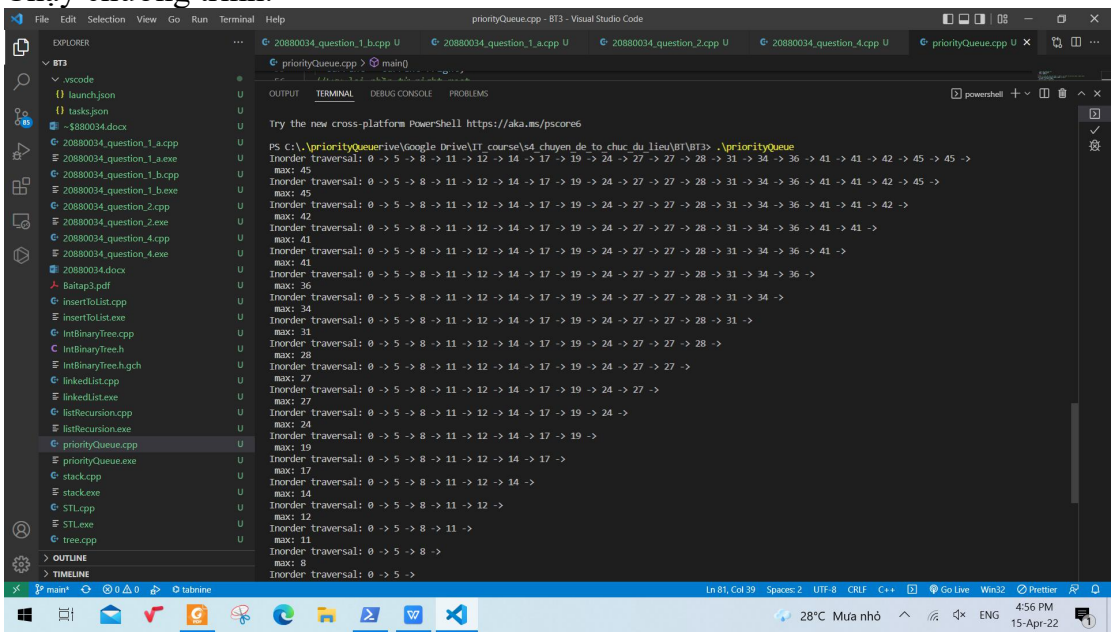
```
// Duyệt giữa cây có nút gốc là root
void inorder(NODE *root)
{
  if (root != NULL) {
    inorder(root->left);
    printf("%d -> ", root->data);
    inorder(root->right);
  }
}
```

```
// Chèn nút với data và độ ưu tiên vào cây.
NODE *Insert(NODE *node, int data, int priority)
{
  //Nếu node bằng NULL thì tạo mới một node
  if (node == NULL) return new NODE(data, priority);
  //Tìm vị trí phù hợp cho node mới
  if (priority < node->priority)
    node->left = Insert(node->left, data, priority);
  else
    node->right = Insert(node->right, data, priority);
  return node;
}
```

```
//Tìm node có độ ưu tiên cao nhất
NODE *RemoveMax(NODE *&node) {
  NODE *current = node;
  //Trường hợp cây không có cây con phải. Node gốc có độ ưu tiên cao nhất.
  if(node->right == NULL)
  {
    //Lưu lại node gốc vào maxPri
    NODE *maxPri = node;
    //Gán lại node gốc mới
    node = node->left;
    return maxPri;
  }
  //Phần tử có độ ưu tiên cao nhất là phần tử xa nhất về phía bên phải (right most)
  while (current && current->right->right != NULL)
    current = current->right;
  //Lưu lại phần tử right most
  NODE *maxPri = current->right;
  //Nếu phần tử right most có cây con trái. Điều chỉnh lại liên kết.
  if(maxPri->left != NULL)
    current->right = maxPri->left;
  else
    current->right = NULL;
  return maxPri;
}
```

```
int main() {
  //Tạo một priorityQueue để lưu các số nguyên.
  NODE *root = NULL;
  //Phát sinh ngẫu nhiên 20 số nguyên có độ ưu tiên là giá trị các số nguyên đó và insert vào priorityQueue.
  for(int i = 0; i < 20; i++)
  {
    int x  = rand()%50;
    root = Insert(root, x, x);
  }
  //Duyệt giữa in các node của cây theo thứ tự tăng dần
  printf("Inorder traversal: ");
  inorder(root);
  //Gỡ bỏ từng phần tử có độ ưu tiên cho đến khi hàng đợi rỗng
  while(root != NULL)
  {
    NODE *max = RemoveMax(root);
    printf("\n max: %d\n", max->data);
    printf("Inorder traversal: ");
    inorder(root);
  }
}
```

Chạy chương trình:



b) Từ cấu trúc dữ liệu đã xây dựng ở câu (a) hãy viết hàm sắp xếp tăng dần một dãy số nguyên.

Với việc cài đặt hàng đợi ưu tiên sử dụng cây nhị phân tìm kiếm. Ta có thể sắp xếp tăng dần các số nguyên bằng cách duyệt giữa cây nhị phân tìm kiếm với độ ưu tiên chính là giá trị của các số nguyên đó.