Ejercicio 1

#include <iostream>

#include <vector>

using namespace std;

// Estructura del árbol de segmentos

struct SegmentTree {

    vector<int> tree; // Array para almacenar el árbol

    int size; // Tamaño del árbol

    // Constructor

    SegmentTree(int n) {

        // Calcula el tamaño del árbol basado en el número de elementos

        size = 1;

        while (size < n)

            size \*= 2;

        tree.resize(2 \* size); // Redimensiona el árbol

    }

    // Actualiza el valor del elemento en la posición index con el valor val

    void update(int index, int val) {

        index += size; // Calcula la posición en el array del árbol

        tree[index] = val; // Actualiza el valor del nodo hoja

        // Propaga los cambios hacia arriba

        while (index > 1) {

            index /= 2;

            if(tree[index \* 2] > tree[index \* 2 + 1]){

                tree[index] = tree[index \* 2];

            }

            else{

                tree[index] = tree[index \* 2 + 1];

            }

        }

    }

    // Calcula la suma de valores en el rango [left, right)

    int query(int left, int right, int node, int nodeLeft, int nodeRight) {

        // Si el rango está completamente fuera del rango del nodo actual

        if (nodeRight <= left || right <= nodeLeft)

            return 0;

        // Si el rango está completamente cubierto por el nodo actual

        if (left <= nodeLeft && nodeRight <= right)

            return tree[node];

        // Divide el rango en dos y realiza las consultas recursivas

        int mid = (nodeLeft + nodeRight) / 2;

        int sumLeft = query(left, right, node \* 2, nodeLeft, mid);

        int sumRight = query(left, right, node \* 2 + 1, mid, nodeRight);

        if(sumLeft > sumRight) return sumLeft;

        else return sumRight;

    }

    // Calcula la suma de valores en el rango [left, right)

    int query(int left, int right) {

        return query(left, right, 1, 0, size);

    }

};

int main() {

    int n; // Número de elementos

    do{

        cin>>n;

    }while(n<0 || n>100);

    SegmentTree st(n);

    for (int i = 0; i < n; i++)

    {

        int x;

        cin>>x;

        st.update(i, x);

    }

    cout<<"Elementos del arbol: ";

    for (int i = 0; i < n; i++)

    {

        cout<<st.tree[n+i+1]<<" ";

    }

    cout<<endl;

    // Actualiza los valores del arbol

    cout<<"Elementos del arbol actualizado: ";

    st.update(0, 1);

    st.update(1, 2);

    st.update(2, 3);

    st.update(3, 4);

    st.update(4, 5);

    for (int i = 0; i < n; i++)

    {

        cout<<st.tree[n+i+1]<<" ";

    }

    cout<<endl;

    int q, l,r;

    cout<<"Ingrese numero de consultas: ";

    cin>>q;

    while(q--){

        cout<<"Ingrese el rango left, right: "<<endl;

        cin>>l>>r;

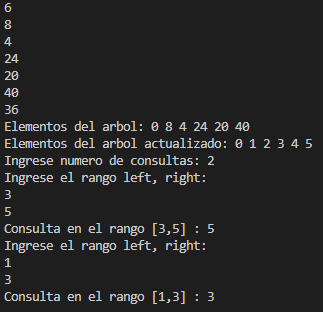
        int mayor = st.query(l, r);

        cout << "Consulta en el rango ["<<l<<","<<r<<"] : "<< mayor << endl;

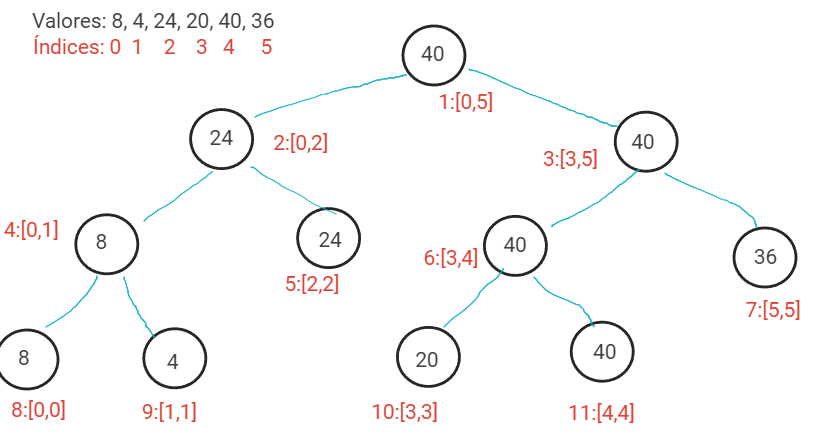
    }

    return 0;

}



Gráfico



Ejercicio 2:

#include <iostream>

#include <vector>

#include <time.h>

using namespace std;

// Función para obtener el siguiente índice

int obtenerSiguiente(int index) {

    // index - LSB(index)

    return index + (index & -index); // obtener el bit menos significativo (LSB)

}

// Función para obtener el padre del índice

int obtenerPadre(int index) {

    // index + LSB(index)

    return index - (index & -index); // obtener el padre de un nodo

}

// Función para actualizar el valor en un índice dado

void actualizar(vector<int>& arbol, int index, int valor) {

    int tamano = arbol.size();

    while (index < tamano) {

        arbol[index] += valor;

        index = obtenerSiguiente(index);

    }

}

// Función para obtener la suma acumulada hasta un índice dado

int obtenerSuma(vector<int>& arbol, int index) {

    int suma = 0;

    while (index > 0) {

        suma += arbol[index];

        index = obtenerPadre(index);

    }

    return suma;

}

// Función para construir el árbol de Fenwick a partir de un arreglo dado

vector<int> construirArbol(const vector<int>& arreglo) {

    int tamano = arreglo.size();

    vector<int> arbol(tamano + 1, 0);

    for (int i = 0; i < tamano; i++) {

        actualizar(arbol, i + 1, arreglo[i]);

    }

    return arbol;

}

void mostrar\_arbol(vector<int>& arbol){

    int i=0;

    while (i<arbol.size()){

        cout << arbol[i] << " ";

        i++;

    }

    cout << endl;

}

int main() {

    //srand(time(NULL));

    int n;

    cin>>n;

    vector<int> arreglo;

    for (int i = 0; i < n; i++)

    {

        int x;

        do{

            cin>>x;

        }while(x<100 || x>2500);

        arreglo.push\_back(x);

    }

    vector<int> arbol = construirArbol(arreglo);

    cout<<"Valores del arbol: ";

    mostrar\_arbol(arbol);

    int q;

    cout<<"Ingrese numero de consultas: ";

    cin>>q;

    while(q--){

        char ope;

        do{

            cin>>ope;

            ope = tolower(ope);

        }while(ope != 'q' && ope!= 'u');

        if(ope == 'q'){

            int l, r;

            cin>>l>>r;

            int sumaRango = obtenerSuma(arbol, r + 1) - obtenerSuma(arbol, l);

            cout << "La suma del rango [" << l << ", " << r << "] es: " << sumaRango << endl;

        }

        else if(ope == 'u'){

            int i, v;

            cin>>i>>v;

            int diferencia = v - arreglo[i];

            arreglo[i] = v;

            actualizar(arbol, i + 1, diferencia);

            cout<<"Valores del arbol con valor actualizado: ";

            mostrar\_arbol(arbol);

        }

    }

    return 0;

}

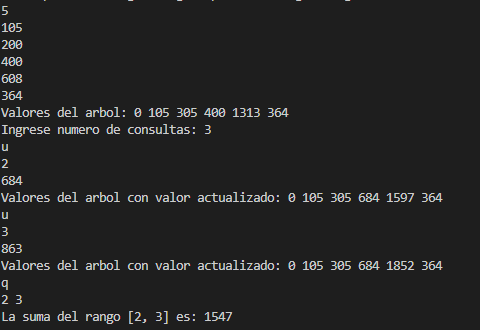


Gráfico:

