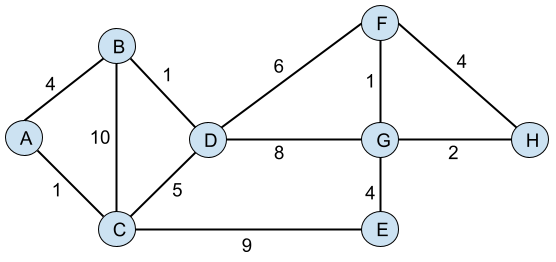
**Posible estructura de examen:**

1. Recursividad.
2. Emplear lista.
3. Algoritmo de ordenación.
4. Algoritmo de búsqueda (Posiblemente Kruskal).
5. Algo de montículos.

**Recorrido sobre grafos:**



**BFS** (cola):

1.Meto el primer nodo en la cola

2.Sacó un nodo de la cola

3.Visito ese nodo

4.Meto en la cola sus nodos adyacentes

5.Vuelta al paso 2 hasta que la cola esté vacía

Y el **DFS**(pila) es igual cambiando cola por pila

Ejemplo examen:

**DFS**(DFS pre-order de un árbol)(Pila)-> A-C-E-G-H-F-D-B

**BFS**(BFS de un árbol)(Cola)->A-B-C-D-E-F-G-H al ser cola se saca igual que entra

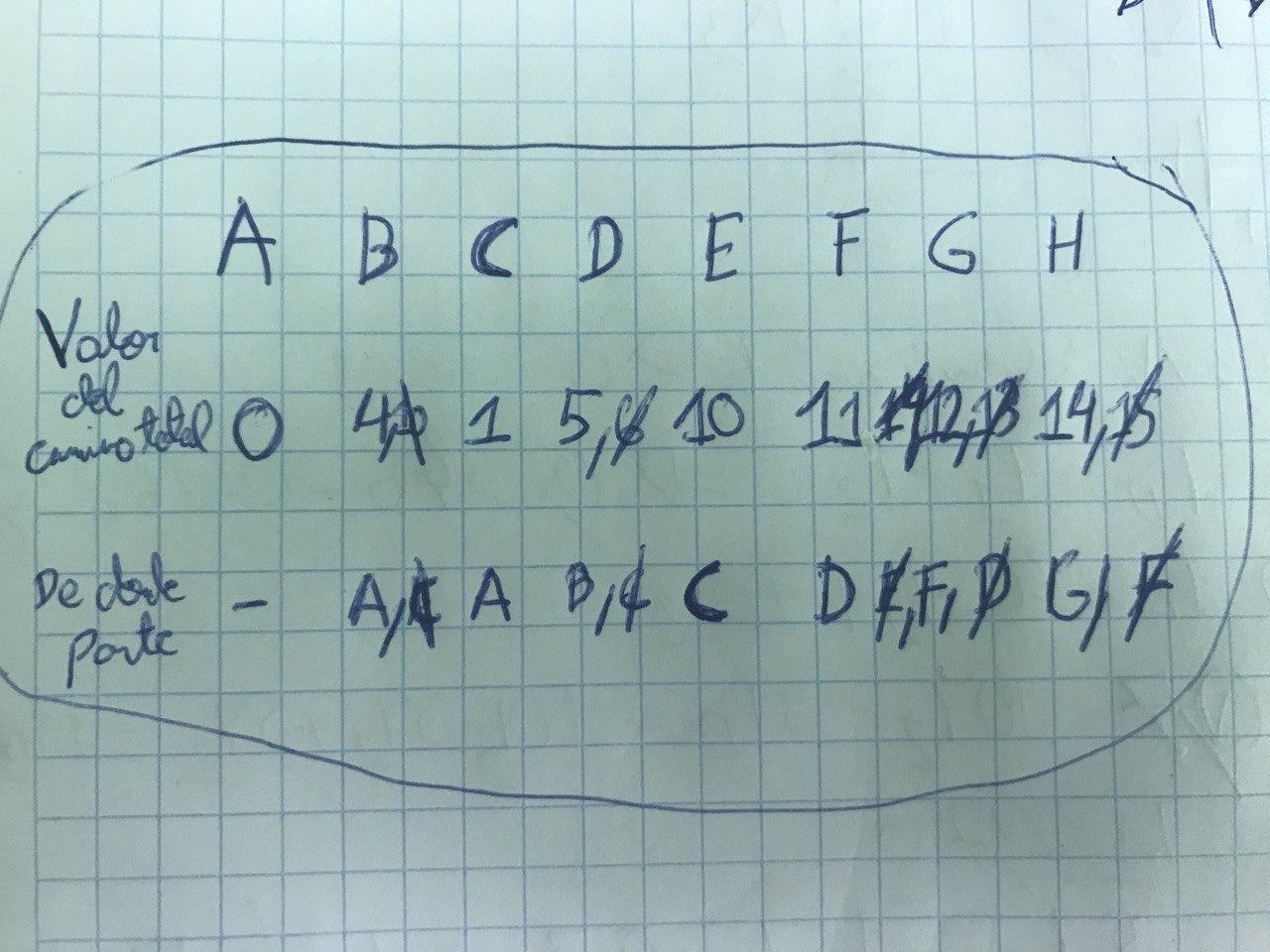
**Dijkstra**:(caminos minimos)

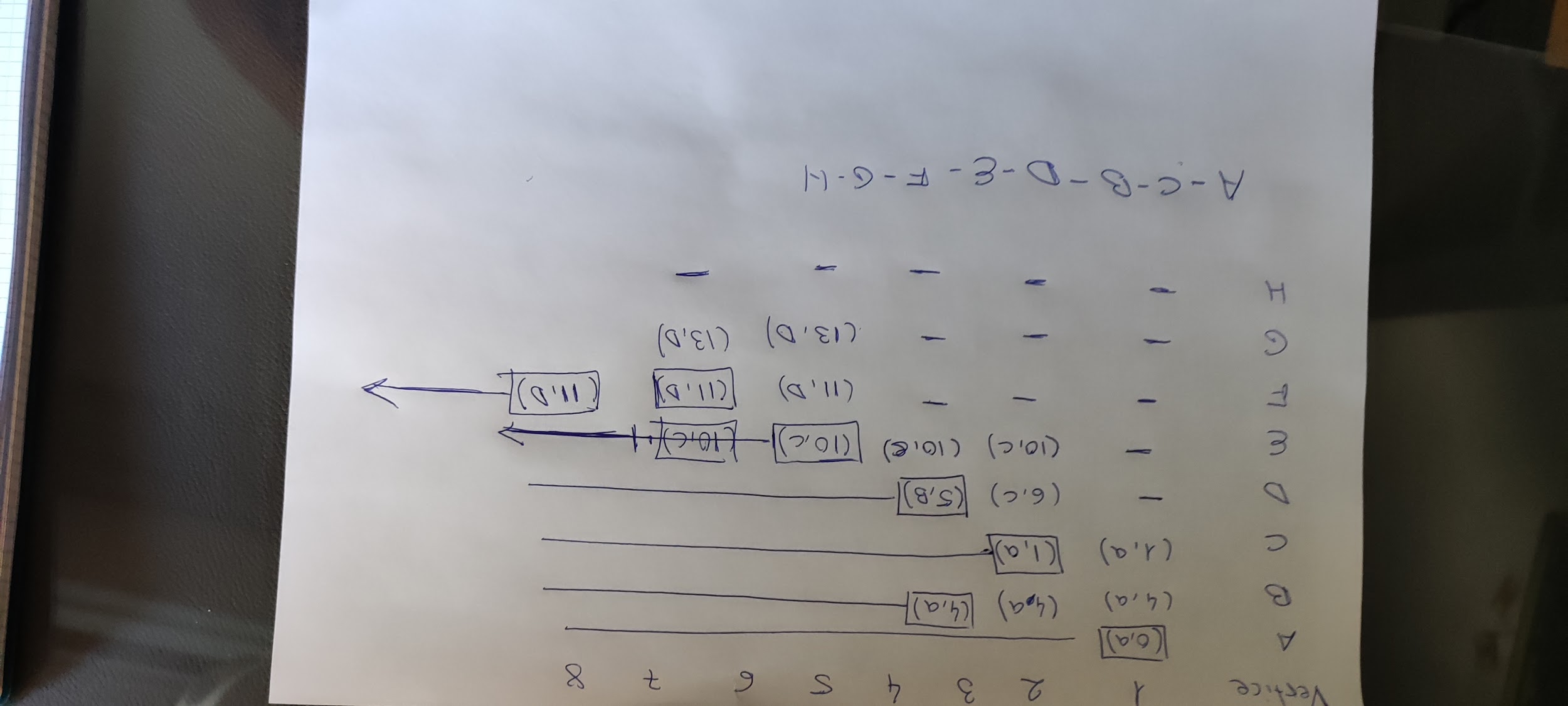
Tratas de encontrar un camino minimo(camino mas corto) desde un vértice de origen hasta el resto de los vértices de un grafo, en un grafo conexo etiquetado. Para realizarlo en un ejercicio de forma teórica se realiza una tabla.

Pasos:

1. Se inicia desde el nodo desde el cual se quieren calcular los caminos
2. Se calcula la distancia acumulada desde ese nodo hasta sus adyacentes no marcados, y se marca el nodo
3. Se escoge el nodo más cercano (no marcado) y se vuelve al paso anterior

Ejemplo del examen : A-C-B-D-E-F-G-H





Ejemplo :[https://jgsogo.es/eda/19.graphs-dijkstra.html#/2/2](https://jgsogo.es/eda/19.graphs-dijkstra.html" \l "/2/2)

Video explicando: <https://www.youtube.com/watch?v=fgdCNuGPJnw>

**Prim**:(árbol de conceptura minimo )

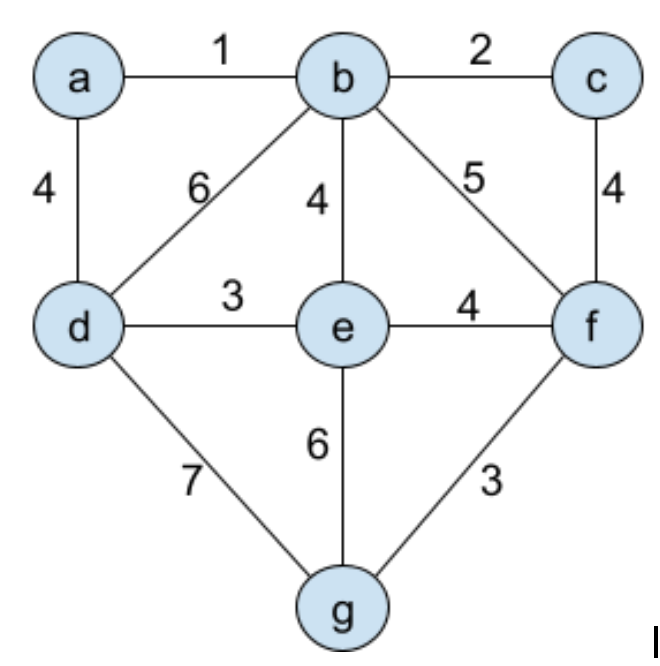
Partes de un nodo y eliges el camino con menor valor o más corto entre todos los posibles:[https://jgsogo.es/eda/17.graphs-prim.html#/2/2](https://jgsogo.es/eda/17.graphs-prim.html" \l "/2/2)

Pasos:

1. Se escoge un vértice al azar.
2. Se coge la arista de menor peso.
3. Paso 2 hasta que se hayan cogido todas las aristas del árbol.

Ejemplo examen:

A-C-B-D-F-G-H-E



A-B-C-D-E-F-G / A-B-C-F-G-E-D / A-B-C-E-D-F-G

Peso grafo: 1+2+4+3+4+3=17

**Kruskal**:(árbol de conceptura minimo)

Se escoge el camino más corto que hay si no hay nodos enlazados , independientemente del nodo que haya al lado . Estos se escogería del 1 al menor necesario en el árbol y teniendo en cuenta que se descarten aquellos que unen otra vez caminos ya unidos anteriormente: [https://jgsogo.es/eda/18.graphs-kruskal.html#/2/2](https://jgsogo.es/eda/18.graphs-kruskal.html" \l "/2/2)

Pasos:

1. Se crea un vértice donde cada vértice es un nodo separador.
2. Se escoge la arista de menor peso que una dos árboles diferentes.
3. Paso 2 hasta que quede un árbol.

-DFS pre-order: primero el nodo raíz/padre/yo y después los hijo y, así recursivamente(en programación se hace con una pila)

-DFS pos-order: primero los hijos y después el nodo padre/raíz/yo, así recursivamente.

-DFS in-order: Se recorre primero el subárbol y luego el nodo raíz y se hace lo mismo con los restantes

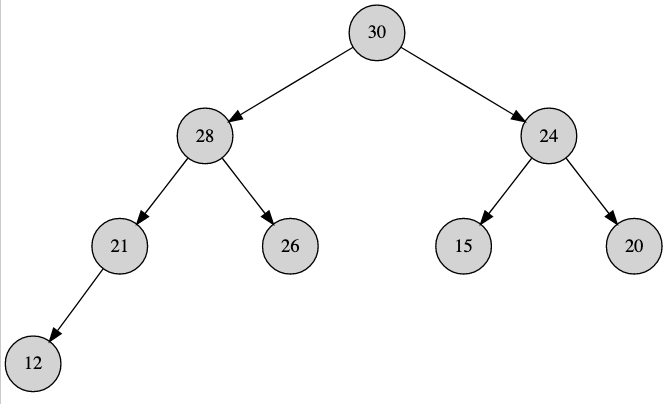
-BFS : Se recorren todos los nodos de un mismo nivel antes de subir al siguiente nivel.

**Montículos:**

En un montículo los nodos se introducen en la última posición del árbol binario y se ascienden hasta que se vuelva a satisfacer la condición de montículo. Para extraer el máximo la operación consiste en intercambiarlo por el último elemento y descender la nueva raíz hasta que se vuelva a satisfacer la condición de montículo.

Colas de prioridad mínimas: Todo nodo es menor que sus descendientes, el nodo raíz es el minimo, la altura es logarítmica respecto al número de nodos, si un sólo elemento no cumple la propiedad de montículo se puede reestablecer mediante ascensos (intercambio con su padre) o descensos (intercambio con el mayor de sus hijos)

Colas de prioridad máximas: El nodo padre en mayor que sus hijos/descendientes, siendo el nodo padre el valor maximo.



Hay que rellenar cada nivel antes de pasar, rellenando primero la izq y luego la derecha.

**ALGORITMOS DE ORDENACIÓN**

**Bubble sort:**

void bubble\_sort(std::vector<int>& elements) {

for (int i=0; i<elements.size()-1; i++) { // 'n-1' iterations

for (int k=0; k<elements.size()-1; k++) { // 'n-1' iterations

if (elements[k] > elements[k+1]) { // compare adyacent values

std::swap(elements[k], elements[k+1]);

}

}

}

Optimización #1

En la pasada i-ésima, todos los últimos i elementos ya están ordenados 🤔

void bubble\_sort(std::vector<int>& elements) {

for (int i=0; i<elements.size()-1; i++) { // 'n-1' iterations

for (int k=0; k<elements.size()-1-i; k++) { // 'n-1-i' iterations

if (elements[k] > elements[k+1]) { // compare adyacent values

std::swap(elements[k], elements[k+1]);

}

}

}

}

Optimización #2

Si en una pasada no intercambio ningún elemento es que todos YA están ordenados 🤔🤔

void bubble\_sort(std::vector<int>& elements) {

for (int i=0; i<elements.size()-1; i++) { // 'n-1' iterations

bool any\_swap = false;

for (int k=0; k<elements.size()-1-i; k++) { // 'n-1-i' iterations

if (elements[k] > elements[k+1]) { // compare adyacent values

std::swap(elements[k], elements[k+1]);

any\_swap = true;

}

}

if (!any\_swap) { break;} // return early

}

}

**Merge Sort:**

vector<int> merge(const vector<int>& lhs, const vector<int>& rhs)

{

vector<int> ret;

auto left\_iterator = 0;

auto right\_iterator = 0;

while (left\_iterator != lhs.size() && right\_iterator != rhs.size()) {

if (lhs[left\_iterator] < rhs[right\_iterator]) {

ret.push\_back(lhs[left\_iterator]);

left\_iterator++;

}else {

ret.push\_back(rhs[right\_iterator]);

right\_iterator++;

}

}

for (int i=left\_iterator; i<lhs.size(); i++) {

ret.push\_back(lhs[i]);

}

for (int i=right\_iterator; i<rhs.size(); i++) {

ret.push\_back(rhs[i]);

}

return ret;

}

vector<int> merge\_sort(const vector<int>& elements) {

if (elements.size() <= 1) {

return elements;

}else {

auto middle = elements.size()/2;

vector<int> left{elements.begin(), elements.begin() + middle};

vector<int> right{elements.begin() + middle, elements.end()};

left = merge\_sort(left);

right = merge\_sort(right);

return merge(left, right);

}

}

**Quick Sort:**

int partition(vector<int>& elements, int left\_index, int right\_index)

{

auto pivot = elements.at(left\_index);

int i = left\_index, j = right\_index;

while(true) {

while( elements.at(i) <= pivot && i <= j ) ++i;

while( elements.at(j) > pivot ) --j;

if( i >= j ) break;

swap(elements.at(i), elements.at(j));

}

swap(elements.at(left\_index), elements.at(j));

return j;

}

void quick\_sort(vector<int>& elements, int left\_index, int right\_index) {

if (left\_index >= right\_index) {

return;

}else {

int pivot\_index = partition(elements, left\_index, right\_index);

quick\_sort(elements, left\_index, pivot\_index-1);

quick\_sort(elements, pivot\_index+1, right\_index);

}

}

**Selection Sort:**

void selection\_sort(std::vector<int>& elements) {

for (int i=0; i<elements.size(); i++) {

int indexMin = i;

for (int k=i+1; k<elements.size(); k++) {

if (elements[k] < elements[indexMin]) {

indexMin = k;

}

}

swap(elements[i], elements[indexMin]);

}

}

**Busqueda Binaria:**

std::vector<int> slice(std::vector<int> values, int begin, int end) {

if (begin >= end) { return std::vector<int>{}; }

return std::vector<int>(values.begin() + begin, values.begin() + end);

}

bool binary\_search(std::vector<int> values, int value\_to\_find) {

if (values.empty()) { return false; }

int middle = values.size() / 2;

if (value\_to\_find < values.at(middle)) {

auto new\_vector = slice(values, 0, middle);

return binary\_search(new\_vector, value\_to\_find);

}else if (value\_to\_find > values.at(middle)) {

auto new\_vector = slice(values, middle+1, values.size());

return binary\_search(new\_vector, value\_to\_find);

}else {

return true;

}

}

int BusquedaBinariaRecursiva(const std::vector<int>& v, int valorAbuscar, int primero, int ultimo) {

if (primero > ultimo) { // Caso base

return -1;

}

int medio = (primero+ultimo)/2;

if (valorAbuscar < values.(medio)){

return BusquedaBinariaRecursiva(v, valorAbuscar, primero, medio-1)

}else if(valorAbuscar > values.at(medio)){

return BusquedaBinariaRecursiva(v, valorAbuscar, medio+1, ultimo);

}else {

return medio;

}

}

int BusquedaBinariaIterativa(const std::vector<int>& v, int valorAbuscar)

{

int primero{0}, ultimo=v.size()-1;

int medio = (primero+ultimo)/2;

while (primero<=ultimo) {

if (v.at(medio)<valorAbuscar) {

primero=medio+1;

}

else if(v.at(medio)>valorAbuscar) {

ultimo = medio - 1;

}

else {//son iguales => Encontrado

return medio;

}

medio = (primero+ultimo)/2;

}

return -1;

}

int busquedaBinariaIterattiva(vector<int> & v, int valor){

int primero{0}, ultimo = v.size()-1;

int medio = (primero+ultimo)/2;

while(primero <= ultimo){

if(valor > v.at(medio)){

primero = medio+1;

}else if(valor < v.at(medio)){

ultimo = medio-1;

}else{

return medio;

}

medio = (primero+ultimo)/2;

}

return -1;

}