**Corrigé-Type ASD 03**

**Exercice 01 (08 pts)**

L= { 13, 17, 5, 21, 14, 22, 9, 20, 8, 6 } (1pt)

|  |
| --- |
|  |

postorderorder (Postfixe) : 6 8 9 5 14 20 22 21 17 13 (1pt)

preorder (prefixe) : 13 5 9 8 6 17 14 21 20 22 (1pt)

inorder (infixe ) : 5 6 8 9 13 14 17 20 21 22 (1pt)

breadth-First (Largeur) : 13 5 17 9 14 21 8 20 22 6 (1pt)

Suppression

|  |  |
| --- | --- |
| Suppression 9 : (1pt) | Suppression 17 (1pt) |

L’arbre AVL

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | |  |
|  | | (1pt) | |

Exercice 02 (04 pt)

// En cas de détection d'erreurs critiques dans le programme, il ne sera pas noté.

bool rechercheElement(Node\* arbre, int v) { (0.5pt)

if (arbre == nullptr) (0.5pt)

return false;

if (arbre->value == v) (1pt)

return true;

if (v < arbre->value)

rechercheElement(arbre->left, v); (1pt)

else

rechercheElement(arbre->right, v);

}

**Complexité** : en moyenne \_(log *n*), en pire \_(*n*) (1pt)

**Exercice 03**

// En cas de détection d'erreurs critiques dans le programme, il ne sera pas noté.

int\* linkedListToArray(Node\* head) { 02pts)

int count = countNodes(head);

int\* arr = new int[count];

int index = 0;

while (head) {

arr[index] = head->data;

head = head->next;

index++;

}

return arr;

}

Node\* arrayToLinkedList(const int arr[], int size) { ( 02pts)

if (size == 0) {

return nullptr;

}

Node\* head = new Node(arr[0]);

Node\* current = head;

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

current->next = new Node(arr[i]);

current = current->next;

}

return head;

}

Algorithme de tri: selon votre choix (3 pts)

Cpmplexité de l’algorithme : selon l’algorithme choisi (1pt)