

Nom :

Prénom :

Groupe :

Le barème est indicatif et susceptible d'être ajusté. Durant toute l'épreuve, il ne sera répondu à **aucune** question.

1 Tris (5 points)

1.1 Tri par insertion (2)

Question 1.1.1 Complétez le diagramme suivant en donnant l'état du tableau après chaque étape intermédiaire du tri par insertion.

init	7	3	1	4	8	9	0	6	5	2
end										

Question 1.1.2 Quelle est la complexité asymptotique du tri par insertion sur un tableau de n éléments dans le meilleur des cas et dans le pire des cas ? Expliquez.

1.2 Tri par tas (3)

Question 1.2.1 Complétez le diagramme suivant en donnant l'état du tableau après chaque étape intermédiaire du tri par tas.

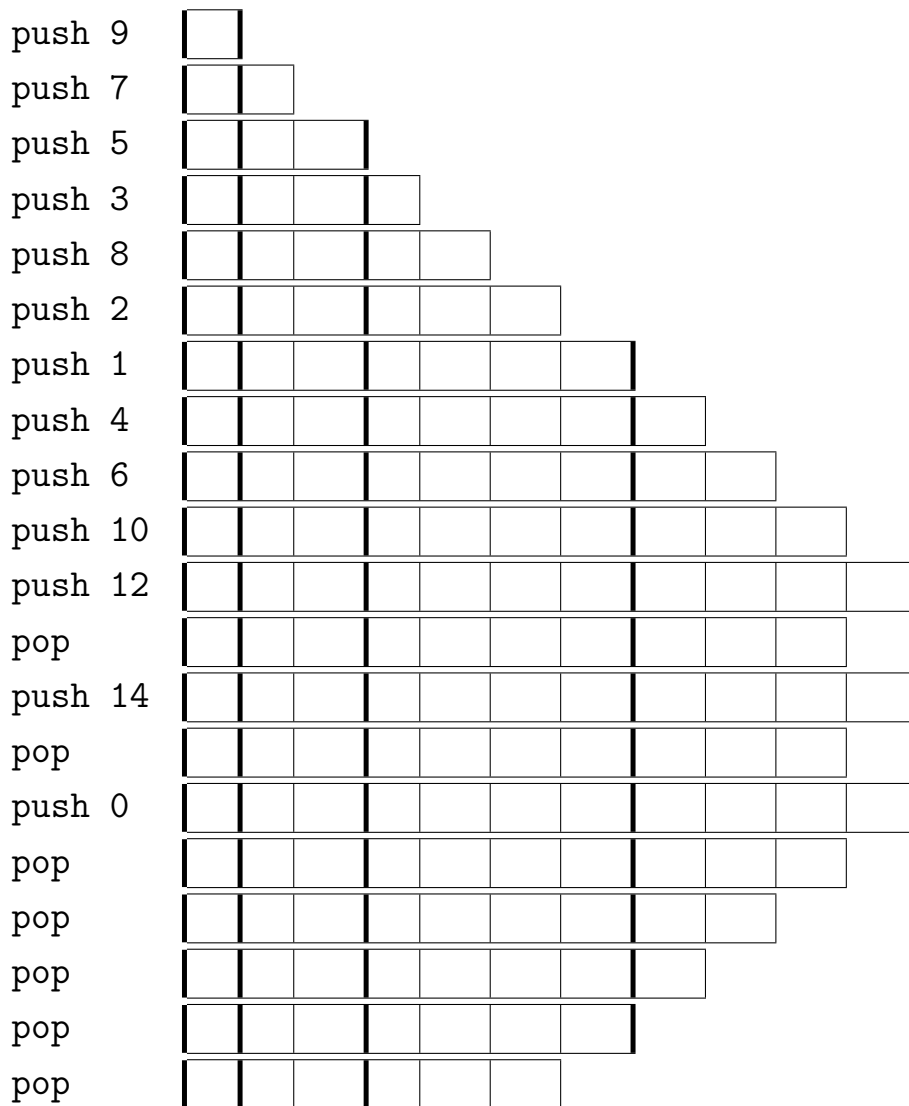
init	7	3	1	4	8	9	0	6	5	2
heapify	9									
end										

Question 1.2.2 Quelle est la complexité asymptotique du tri par tas sur un tableau de n éléments dans le meilleur des cas et dans le pire des cas ? Expliquez.

Question 1.2.3 Quel est l'avantage du tri par tas par rapport au tri fusion ? Expliquez.

2 Tas (3 points)

Question 2.1 Compléter le diagramme suivant en donnant l'état du tas sous forme de tableau **après** chaque opération indiquée à gauche. On considère ici un tas-min (le minimum est accessible en $\mathcal{O}(1)$) disposant de deux opérations usuelles : **push** pour ajouter une valeur sur le tas et **pop** pour enlever le minimum.



Question 2.2 Quelle est la complexité asymptotique des méthodes **push**, **pop** et **heapify** pour un tas de n éléments ?

3 Tas optimisé pour la suppression (4 points)

Le problème des tas est que la suppression d'un élément autre que l'extremum n'est pas possible de manière efficace. On propose ici d'ajouter à un tas, la possibilité de supprimer un élément quelconque du tas rapidement. Pour cela, on utilise un dictionnaire `indexes` qui à chaque valeur dans le tas associe sa position dans le tableau qui représente le tas.

```
class BinaryHeap:
    def __init__(self):
        self.tab=[]
        self.indexes={}

    def swap(self,i,j):
        self.tab[i],self.tab[j]=self.tab[j],self.tab[i]
        pass

    def percolateUp(self,i): ...
    def percolateDown(self,i): ...
    def push(self,v):
        self.tab.append(v)
        self.indexes[v] = len(self.tab) - 1
        self.percolateUp(len(self.tab) - 1)

    def pop(self): ...
```

Question 3.1 La méthode `swap` donnée dans l'énoncé n'est pas satisfaisante : elle ne met pas à jour `indexes`. Écrire ici la ou les lignes nécessaires (à la place du `pass`) pour que la méthode `swap` fonctionne correctement.

```
def swap(self,i,j):
    self.tab[i],self.tab[j]=self.tab[j],self.tab[i]
```

Question 3.2 Écrire la méthode `delete` qui efface une valeur dans le tas de manière efficace tout en conservant les bonnes propriétés de la structure de données (pour effacer une entrée associée à une valeur dans un dictionnaire on peut utiliser `del dictionnaire[valeur]`)

```
def delete(self,v):
```

4 Connexité (4 points)

On considère une classe `Graph` représentant des graphes non dirigés, et on souhaite avoir une méthode `connected` qui renvoie un booléen nous indiquant si le graphe est connecté (une seule composante connexe) ou non. La classe `Graph` dispose déjà d'une méthode `vertex_count` (et `edge_count`) qui renvoie le nombre de sommets (respectivement arêtes) dans le graphe et d'une méthode `random_vertex` qui renvoie un sommet du graphe.

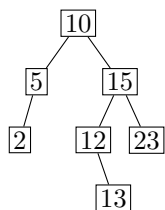
Question 4.1 Complétez la fonction `aux` et la fonction `connected`. La complexité asymptotique de votre solution doit être optimale. Voici le code provisoire de la fonction :

```
def connected(self):
    marked=set()
    def aux(v) :
        marked.add(v)
        for n in self.adjacents(v) :
```

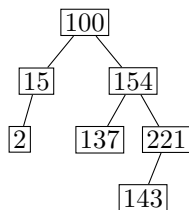
5 AVL (4 points)

Question 5.1

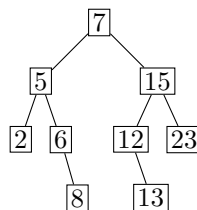
Parmi les arbres suivants, cochez les arbres qui sont des AVL.



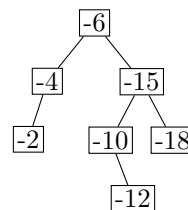
☐



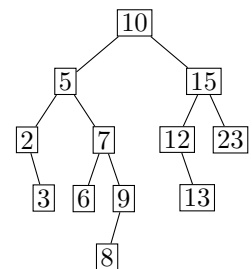
☐



☐



☐



☐

Question 5.2 Deux arbres à dessiner

On considère un AVL vide auquel on ajoute successivement les valeurs 10, 20, 30, 40, 50, 60. Dessiner l'arbre obtenu après ces 6 opérations. À partir de ce résultat, on ajoute ensuite successivement 35, 33, 36, 37, 38. Dessiner l'arbre obtenu après ces 11 opérations.