1 Présentation

2 Réalisation de l'algorithme

2.1 Tri des points

Question 1 *Citer 3 algorithmes de tris et donner leur complexité dans le meilleur des cas, le pire des cas et le cas moyen.*

Correction

	Tri par insertion	Tri rapide	Tri fusion
Pire des cas	$C(n)=\mathcal{O}(n^2)$	$C(n)=\mathcal{O}(n^2)$	$C(n)=\mathcal{O}(n^2)$
Cas moyen	$C(n)=\mathcal{O}(n^2)$	$C(n)=\mathcal{O}(n\log n)$	$C(n)=\mathcal{O}(n\log n)$
Meilleur des cas	$C(n)=\mathcal{O}(n)$	$C(n)=\mathcal{O}(n\log n)$	$C(n)=\mathcal{O}(n\log n)$

Question 2 *On donne l'algorithme d'un tri dans le document réponse (à la question suivante). Donner son nom et remplacer les fonctions «mystere» par un nom plus adéquat.*

Question 3 *Modifier l'algorithme de tri donné pour qu'il prenne en argument la liste des points* `P`*. Pour cela vous barrerez et réécrirez les lignes à modifier.*

Correction
Il s'agit du tri rapide.

▪ Python
```
def segmente(tab,i,j):
    g =i+1
    d=j
    p=tab[i][0]
    while g<=d :
        while d>=0 and tab[d][0]>p:
            d=d-1
        while g<=j and tab[g][0]<=p:
            g=g+1
        if g<d :
            tab[g],tab[d]=tab[d],tab[g]
            d=d-1
            g=g+1
    k=d
    tab[i],tab[d]=tab[d],tab[i]
    return k

def tri_quicksort(tab,i,j):
    if i<j :
        k = segmente(tab,i,j)
        tri_quicksort(tab,i,k-1)
        tri_quicksort(tab,k+1,j)
```

2.2 Fonction orientation

Question 4 *En utilisant les points dans le tableau de la page précédente, donner le résultat du test d'orientation pour les points d'indices suivants :*

- $i=0$, $j=1$, $k=2$;
- $i=6$, $j=7$, $k=8$.

Correction
Les points P_0, P_3 et P_4 les points sont orientés positivement. On peut le voir graphiquement. Sinon, on a : $\overrightarrow{P_0P_1}=(2,1)$

et $\overrightarrow{P_0P_2} = (2,6)$. $\begin{vmatrix} 2 & 2 \\ 1 & 6 \end{vmatrix} = 12-2 = 10$.

Mes points P_6, P_7 et P_8 les points sont orientés négativement. On peut le voir graphiquement. Sinon, on a : $\overrightarrow{P_6P_7} = (2,4)$ et $\overrightarrow{P_6P_8} = (3,1)$. $\begin{vmatrix} 2 & 3 \\ 4 & 1 \end{vmatrix} = 2-12 = -10$.

Question 5 *Écrire une fonction* `orientation(tab,i,j,k)` *prenant comme paramètres la liste des points ainsi que l'indice des points à tester. Cette fonction renverra* -1, 0 *ou* 1 *suivant le résultat du test d'orientation des points d'indices* i, j *et* k.

Correction

Python
```
def orientation(tab,i,j,k):
    p1, p2, p3 = tab[i], tab[j], tab[k]
    p12 = [p2[0]-p1[0],p2[1]-p1[1]]
    p13 = [p3[0]-p1[0],p3[1]-p1[1]]
    det = p12[0]*p13[1]-p12[1]*p13[0]
    if det <0 :
        return -1
    elif det >0 :
        return 1
    else :
        return 0
```

2.3 Création de l'enveloppe supérieure

Question 6 *À l'instant* 0, *quelle instruction permet de tester si* `Es` *est vide ? Que faut-il alors faire pour initialiser* `Es` *? À l'instant* i, *que faire si* `Es` *ne contient qu'un seul élément ?*

Correction À l'instant 0, `Es` est vide. On peut le tester avec la fonction `est_vide`. On empile donc l'élément 0.
À l'instant i, si `Es` est vide, on empile l'élément i.
À l'instant i, si `Es` ne contient qu'un seul élément, on empile l'élément i.

Question 7 *Écrire la fonction* `enveloppe_Sup(tab,Es,i)`.

Correction

Python
```
def enveloppe_Sup_rec(tab,Es,i):
    if est_vide(Es):
        push(Es,i)
    else :
        p1 = i
        p2 = pop(Es)
        if est_vide(Es):
            push(Es,p2)
            push(Es,p1)
        else :
            p3 = top(Es)
            if orientation(tab,p1,p2,p3)>0:
                push(Es,p2)
                push(Es,p1)
            else :
                enveloppe_Sup_rec(tab,Es,i)
```

2.4 Analyse de l'algorithme final

Question 8 *On donne dans le document réponse l'algorithme permettant de retourner la pile contenant les indices des sommets de l'enveloppe convexe à partir d'un nuage de points. Commenter les blocs d'instructions de ce programme. Donner la complexité de l'algorithme dans le pire des cas.*

Correction

Python

```
def enveloppe_convexe(tab):
    tri(tab)                        # Tri de la liste des points
    es = creer_pile()               # Initialisation de l'enveloppe supérieure
    ei = creer_pile()               # Initialisation de l'enveloppe inférieure
    for i in range (len(tab)):      # On balaye les points et on crée les enveloppes inf et sup
        enveloppe_Inf(tab,ei,i)
        enveloppe_Sup(tab,es,i)
    pop(es)                         # Tous les points vont être transférés dans l'enveloppe inf pour
                                    # créer l'enveloppe finale. Le dernier point de la liste initiale
                                    # est dans les deux enveloppes. On supprime donc le doublon.

    while (not(est_vide(es))):      # On dépile l'enveloppe sup dans l'enveloppe inf
        push(pop(es),ei)            #
    pop(ei)                         # Le point 0 étant un doublons on le supprime.
    return (ei)                     #
```