

Nom et Prénoms : Groupe :

1) Analyser les complexités au pire cas en fonction de n : **(11pts)**

```
for(int i=1 ; i<=n ; i++)
  for(int j=i ; j<=n ; j++)
    ..//nombre constant d'opérations
```

A**(3pts)**

- Boucle d'indice i : n **(0,5)**
- Boucle d'indice j : A chaque itération de i, elle se répète (n-i+1) fois, soit n, n-1, n-2, 2, 1 **(1)**
- Complexité totale :
 $n + (n-1) + (n-2) + \dots + 2 + 1 = O(n^2)$ **(1,5)**

```
int i = 100 ;
while (i>0) i-- ;
```

B**(1 pt)**

Nbr constant d'opération (100), **(0,25)**
 donc $O(1)$ **(0,75)**

```
for (int i = n ; i>0 ; i = i/2) {
  for (int j = 1 ; j<n ; j = j*2) {
    for (int k = 0 ; k<n ; k = k+2) {
      ..//nombre constant d'opérations
    }
  }
}
```

C**(3 pts)**

- La boucle d'indice i : i varie comme suit n, n/2, n/2², jusqu'à $n/2^k = 1 \Rightarrow k = \log_2(n)$ **(1)**
- La boucle d'indice j : j varie comme suit 1, 2, 2², jusqu'à $2^k = n \Rightarrow k = \log_2(n)$ **(1)**
- La boucle d'indice k : n/2 **(0,5)**
- Complexité totale : $(n/2) \cdot (\log_2(n))^2$
 $= O(n(\log_2(n))^2)$ **(0,5)**

```
int f1(int T[50],int debut , int fin) {
  //1er appel debut = 0, fin = n-1;
  if (debut < fin) {
    milieu = (debut + fin)/2;
    f1(T, debut, milieu) ;
    f1(T, milieu+1, fin);
    f2(T, debut, milieu,fin); //O(n)
  }
}
```

D**(3,5 pts)**

$$T(n) = \begin{cases} O(1) & \text{si } n = 1 \\ 2 * T(n/2) + O(n) & \text{sinon} \end{cases} \quad \begin{matrix} (0,25) \\ (1,25) \end{matrix}$$

Si omission d'un terme, cad trouver: $T(n/2) + n$ ou bien $2 T(n/2) \rightarrow$ **(0,5)**, et donner à la suite la moitié de la note si le raisonnement est juste

On développe et on obtient : **(1)**

$$\begin{aligned} T(n) &= 2 * T(n/2) + n \\ T(n) &= 2 * [2 * T(n/2^2) + n/2] + n \\ T(n) &= 2^2 * T(n/2^2) + 2n \\ T(n) &= 2^2 * [2 * T(n/2^3) + n/2^2] + 2n \\ T(n) &= 2^3 * T(n/2^3) + 3n \end{aligned}$$

En général :

$$T(n) = 2^k * T(n/2^k) + kn \quad (0,25)$$

Arrêt lorsque $n/2^k = 1 \Rightarrow k = \log_2 n$ **(0,25)**

$$\text{Donc } T(n) = n + n \log_2 n \quad (0,25)$$

$$T(n) = O(n \log_2 n) \quad (0,25)$$

2) Ordre (du plus rapide au plus lent) : B ($O(1)$) - D ($O(n \log_2 n)$) - C ($O(n(\log_2(n))^2)$) - A ($O(n^2)$) **(0,5)**

3) Ecrire les fonctions suivantes en C/C++ **(9 pts)****Suppression à la fin d'une liste chaînée simple d'entiers****(5 pts)**

```

struct bloc { int val, bloc* suiv } (0,5)

void supprimer(bloc * &tete)

if (tete == NULL) cout<<"liste vide"; (0,5)
else {
    bloc * precedent = tete; (0,25)
    bloc * courant = tete; (0,25)
    while (courant->suiv != NULL){ (0,5)
        precedent = courant ; (0,25)
        courant = courant -> suiv; (0,25)
    }
    bloc * q = courant; (0,25)
    if (tete -> suiv == NULL) //1 elmt (0,25)
        tete = NULL; (0,25)
    else precedent -> suiv = NULL; (0,5)
    delete q; (0,25)
}
}

```

Complexité au meilleur casListe vide ou 1 elmt: $O(1)$ **(0,5)****Complexité au pire cas**Parcours de n éléments $O(n)$ **(0,5)****Insertion en tête d'une liste contigue de n entiers****(4 pts)**

```

const int max = 100; (0,5)
struct tableau { int T[max], n};
(accepter aussi une déclaration simple de
type int T[max])

void inserer (tableau liste, int x) {

    if (liste.n == max-1) (0,5)
        cout<<"tableau plein..";
    else { //décalage à droite
        for (int i=liste.n-1; i>0; i--) (0,5)
            liste.T[i] = liste.T[i-1]; (0,5)

        liste.T[0]= x; (0,5)
        liste.n ++; (0,5)
    }

}

```

Complexité au meilleur casListe pleine ou vide (aucun décalage) : $O(1)$ **(0,5)****Complexité au pire cas**Décalage de n elmts : $O(n)$ **(0,5)**