

1 Exercice 2

1.1 Algo naïf

```
1: function SOUSSUITENAIF( $E$ )
2:    $somme \leftarrow E[0]$ 
3:    $d \leftarrow 0$ 
4:    $f \leftarrow 0$ 
5:   for  $i = 0; i < t; n; i++$  do
6:     for  $j = n - 1; j \geq i; i++$  do
7:        $p \leftarrow 0$ 
8:       for  $k = i; k \leq j; k++$  do
9:          $p+ = E[k]$ 
10:      end for
11:      if  $p > somme$  then
12:         $somme \leftarrow p$ 
13:         $d \leftarrow i$ 
14:         $f \leftarrow j$ 
15:      end if
16:    end for
17:  end for
18:  return ( $d, f$ )
19: end function
```

1.2 Complexité

$$C \in \Theta(n^2)$$

1.3 Chercher à droite

```
1: function CHERCHER_A_DROITE( $E, i, j$ )
2:    $s \leftarrow E[i]$ 
3:    $k \leftarrow i + 1$ 
4:    $f \leftarrow i$ 
5:    $smax \leftarrow E[i]$ 
6:   while  $k \leq j$  do
7:      $s+ = E[k]$ 
8:     if  $s > smax$  then
9:        $smax \leftarrow s$ 
10:       $f \leftarrow k$ 
11:    end if
12:  end while
13:  return ( $f, smax$ )
14: end function
```

1.4 Complexité

$$C \in \Theta(n)$$

1.5 Spécifier l'algorithme diviser pour régner

- Prend un tableau E en argument
- Prend en argument les bornes du tables i et j
- Algorithme récursif
- Renvois le triplet (s, d, f) où :
 - s est la somme de la suite $[d, f]$
 - d est l'index de début de la suite
 - f est l'index de fin de la suite

1.6 Expliquer son principe (3 phases)

Phase 1 : Diviser par 2 la liste

Phase 2 : Régner sur chaque moitié

Phase 3 : Combiner les cas entre

- Maximum de chaque moitié
- Les cas de "recollement" dans le cas où la division a "coupé" notre sous-suite de somme maximale (utilisation de *chercher_a_droite* et *chercher_a_gauche*)

1.7 Écrire l'algorithme

```
1: function SOUSSUITE( $E, i, j$ )
2:   if  $i == j$  then                                     ▷ Cas d'arrêt
3:      $smax \leftarrow E[i]$ 
4:      $dmax \leftarrow i$ 
5:      $fmax \leftarrow j$ 
6:   else
7:      $k \leftarrow (i + j)/2$ 
8:   end if
9:    $(smax1, dmax1, fmax1) = Sous\_Suite(E, k + 1, j)$       ▷ Régner
10:   $(smax2, dmax2, fmax2) = Sous\_Suite(E, i, k)$ 
11:   $(smax3, dmax3) = chercher\_a\_gauche(E, i, k)$ 
12:   $(smax4, dmax4) = chercher\_a\_droite(E, k + 1, j)$ 
13:   $(smax3, dmax3, fmax3) \leftarrow (s3 + s4, dmax3, fmax3)$   ▷  $s3 + s4 ??$ 
14:   $(smax4, dmax4, fmax4) \leftarrow (s3 + s4, dmax4, fmax4)$   ▷  $s3 + s4 ??$ 
15:   $smax \leftarrow \max(smax1, smax2, smax3, smax4)$           ▷ Combinaison
16:  if  $smax == smax1$  then
17:    return  $(smax, dmax1, fmax1)$ 
18:  else if  $smax == smax2$  then
19:    return  $(smax, dmax2, fmax2)$ 
20:  else if  $smax == smax3$  then
21:    return  $(smax, dmax3, fmax3)$ 
22:  else
23:    return  $(smax, dmax4, fmax4)$ 
24:  end if
25: end function
```

1.8 Complexité

??

1.9 Algorithme à complexité linéaire

Demandez à Alexandre.