Structure de Données
Correction CC

Marie Pelleau
marie.pelleau@unice.fr

Semestre 3

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
	10					

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10					

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
		10				

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
	3	10				

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
			10			

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
		9	10			

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12
				10		

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12
			9	10		

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12
		8	9	10		

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12
2	3	8	9	10	7	12

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12
2	3	8	9	10	7	12
					10	

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12
2	3	8	9	10	7	12
				9	10	

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12
2	3	8	9	10	7	12
			8	9	10	

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12
2	3	8	9	10	7	12
		7	8	9	10	

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12
2	3	8	9	10	7	12
2	3	7	8	9	10	12

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12
2	3	8	9	10	7	12
2	3	7	8	9	10	12

Donnez le nombre de modifications effectuées

Exécutez l'algorithme du tri par insertion

```
tri_insertion(T[], n) {
  pour (i de 2 à n) {
    x <- T[i]
    j <- i
    tant que (j > 1 et
        T[j - 1] > x) {
      T[j] <- T[j - 1]
      j <- j - 1
    }
    T[j] <- x
  }
}
```

1	2	3	4	5	6	7
10	2	3	9	8	7	12
2	10	3	9	8	7	12
2	3	10	9	8	7	12
2	3	9	10	8	7	12
2	3	8	9	10	7	12
2	3	7	8	9	10	12

Donnez le nombre de modifications effectuées

13 modifications

Exécutez cet algorithme pour n = 4

```
secret(n) {
  x <- 1
  y <- 1
  tant que (x < n) {
    x <- x + 1
    y <- y + 2 * x - 1
  }
}
```

x	y

Exécutez cet algorithme pour n = 4

```
secret(n) {
  x <- 1
  y <- 1
  tant que (x < n) {
    x <- x + 1
    y <- y + 2 * x - 1
  }
}
```

x	y
1	

Exécutez cet algorithme pour n = 4

```
secret(n) {
  x <- 1
  y <- 1
  tant que (x < n) {
    x <- x + 1
    y <- y + 2 * x - 1
  }
}
```

x	y
1	1

Exécutez cet algorithme pour n = 4

```
secret(n) {
  x <- 1
  y <- 1
  tant que (x < n) {
    x <- x + 1
    y <- y + 2 * x - 1
  }
}
```

x	y
1	1
2	

Exécutez cet algorithme pour n = 4

```
secret(n) {
  x <- 1
  y <- 1
  tant que (x < n) {
    x <- x + 1
    y <- y + 2 * x - 1
  }
}
```

x	y
1	1
2	4

Exécutez cet algorithme pour n = 4

```
secret(n) {
  x <- 1
  y <- 1
  tant que (x < n) {
    x <- x + 1
    y <- y + 2 * x - 1
  }
}
```

x	y
1	1
2	4
3	

Exécutez cet algorithme pour n = 4

```
secret(n) {
  x <- 1
  y <- 1
  tant que (x < n) {
    x <- x + 1
    y <- y + 2 * x - 1
  }
}
```

x	y
1	1
2	4
3	9

Exécutez cet algorithme pour n = 4

```
secret(n) {
  x <- 1
  y <- 1
  tant que (x < n) {
    x <- x + 1
    y <- y + 2 * x - 1
  }
}
```

x	y
1	1
2	4
3	9
4	

Exécutez cet algorithme pour n = 4

```
secret(n) {
  x <- 1
  y <- 1
  tant que (x < n) {
    x <- x + 1
    y <- y + 2 * x - 1
  }
}
```

x	y
1	1
2	4
3	9
4	16

Exécutez cet algorithme pour n = 4

```
secret(n) {
  x <- 1
  y <- 1
  tant que (x < n) {
    x <- x + 1
    y <- y + 2 * x - 1
  }
}
```

x	y
1	1
2	4
3	9
4	16

Que fait cet algorithme ?

Exécutez cet algorithme pour n = 4

```
secret(n) {
  x <- 1
  y <- 1
  tant que (x < n) {
    x <- x + 1
    y <- y + 2 * x - 1
  }
}
```

x	y
1	1
2	4
3	9
4	16

Que fait cet algorithme ?

Calcule dans y la valeur de n^2

Écrivez l'algorithme ajouteEnFin(elt , L)

```
entier nombreElements(L) {
   cpt <- 0
   elt <- premier(L)
   tant que (elt ≠ nil) {
      cpt <- cpt + 1
      elt <- suivant(elt)
   }
   retourner cpt
}
```

```
insèreArpès(elt , p, L) {
   // elt n'est pas dans L, p
         est dans L
   suivant(elt) <- suivant(p)
   suivant(p) <- elt
}
```

Écrivez l'algorithme ajouteEnFin(elt , L)

```
ajouteEnFin(elt , L) {
   si (premier(L) = nil) {
      premier(L) <- elt
   } sinon {
      e <- premier(L)
      tant que (suivant(e) ≠ nil) {
         e <- suivant(e)
      }
      suivant(e) <- elt
   }
   suivant(elt) <- nil
}
```

Yvain et Gauvain

Lors d'une quête pour le graal, Yvain et Gauvain se retrouvent avec deux sacs contenant chacun 10 objets de valeurs différentes. Cependant, ayant oublié d'attacher leurs chevaux, ils ne peuvent prendre qu'un seul sac à dos. Ils choisissent donc de ne prendre que les dix objets les plus chers de leurs deux sacs à dos.

Remarque : Ces sacs à dos ont la bonne idée d'être rangés en fonction de la valeur des objets : en première position se trouve l'objet le moins cher, et ainsi de suite jusqu'à la dernière position où se trouve l'objet le plus cher.

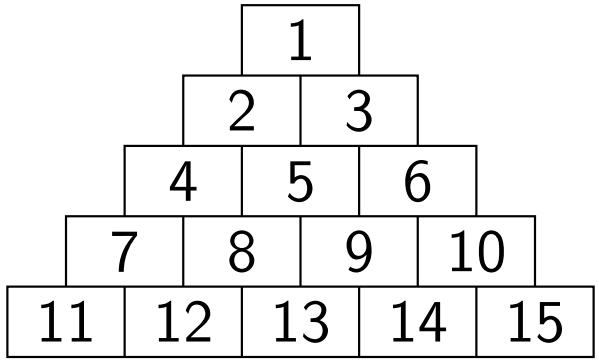
En supposant que les sacs à dos sont représentés par des tableaux de $n = 10$ éléments, écrivez l'algorithme en $O(n)$ qui permet de remplir le sac à dos final qu'ils rapporteront à la table ronde. En cas de valeur égale, on prendra l'objet qui se trouve dans le premier sac considéré.

Yvain et Gauvain

```
pourLeGraal(T1, T2, n) {
  i1 <- n
  i2 <- n
  k <- n
  tant que (k >= 1) {
    si (T1[i1] < T2[i2]) {
      T[k] <- T2[i2]
      i2 <- i2 - 1
    } sinon {
      T[k] <- T1[i1]
      i1 <- i1 - 1
    }
    k <- k - 1
  }
}
```

Pyramide

En considérant une pyramide comme suit :



On peut la considérer dans la suite, comme un tableau à une dimension (on linéarise la pyramide). La pyramide dessinée ci-dessus serait alors vue comme le tableau :

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Retrouver la case dans le tableau avec le numéro de l'élément dans une ligne

Retrouver la case dans le tableau avec le numéro de l'élément dans une ligne

```
case(e, l) {
   si (e <= l et e >= 1) {
      retourner l * (l - 1) / 2 + e
   } sinon retourner 0
}
```

Retrouver les deux fils ou les deux ancêtres d'un élément

Retrouver les deux fils ou les deux ancêtres d'un élément

Retrouver les deux fils d'un élément

```
fils(e, l) {
   retourner [case(e, l + 1), case(e + 1, l + 1)]
}
```

Retrouver les deux fils ou les deux ancêtres d'un élément

Retrouver les deux fils d'un élément

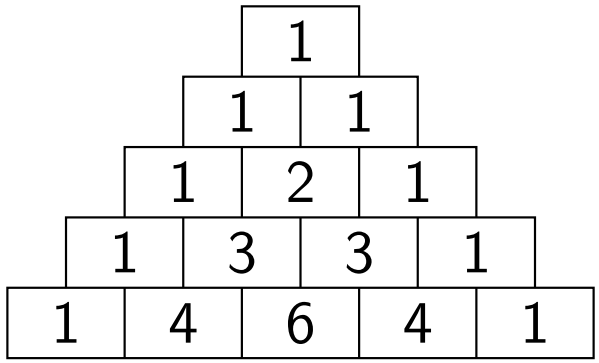
```
fils(e, l) {
   retourner [case(e, l + 1), case(e + 1, l + 1)]
}
```

Retrouver les deux ancêtres d'un élément

```
ancêtres(e, l) {
   retourner [case(e - 1, l - 1), case(e, l - 1)]
}
```

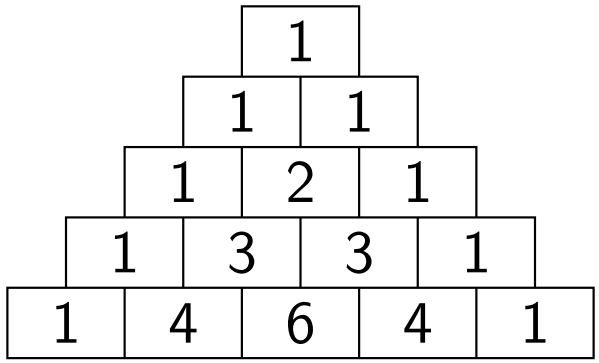
Remplir la pyramide

Si maintenant on met dans la racine une valeur k, et qu'on remplit la pyramide de la façon suivante : le fils contient la somme des deux pères ou k si un seul père.



Remplir la pyramide

Si maintenant on met dans la racine une valeur k, et qu'on remplit la pyramide de la façon suivante : le fils contient la somme des deux pères ou k si un seul père.



1	1	1	1	2	1	1	3	3	1	1	4	6	4	1

Remplir la pyramide

```
remplir(k, nbl) {
  T[1] <- k
  pour (i de 2 à nbl) {
    pour (j de 1 à i) {
      pères <- ancêtres(i, j)
      somme <- 0
      si (pères[1] ≠ 0 ) { somme <- somme + T[pères[1]]
          }
      si (pères[2] ≠ 0 ) { somme <- somme + T[pères[2]]
          }
      T[case(i, j)] <- somme
    }
  }
}
```

Remplir la pyramide

```
remplir(k, nbl) {
  n <- (nbl * (nbl + 1)) / 2
  pour (i de 2 à n) {
    T[i] <- 0
  }
  T[1] <- k
  pour (l de 1 à nbl) {
    pour (e de 1 à l) {
      maValeur <- T[case(e, l)]
      mesFils <- fils(e, l)
      T[mesFils[1]] <- T[mesFils[1]] + maValeur
      T[mesFils[2]] <- T[mesFils[2]] + maValeur
    }
  }
}
```