**Rapport TP complexité**

**TP2**

**Rapport réalisé par : Bougaoua Ibrahim**

**M1 IL G3 / 161631037867**

**Recherche d’un élément :**

**1) Recherche tab non triés.**

**void rechEletsTabNonTries(int \*tab,int n,int value) {**

**int i = 1; // 1**

**while( i <= n ) // n + 1 or 1**

**{**

**if( tab[i] == value ) // n or 1**

**break; // 1**

**i++; // 2\*n - 1 or 0**

**}**

**if( i > n ) // 1**

**printf("This value doesn't exists : value = %d \n",value); // 1**

**else**

**printf("This value exists : value = %d \n",value); // 1**

**}**

**Complexité théorique :**

**Au pire cas :**

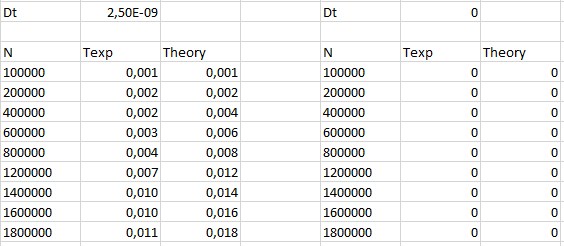
**f(n) = 1 + n + 1 + n + 1 + 2\*n - 1 + 1 + 1**

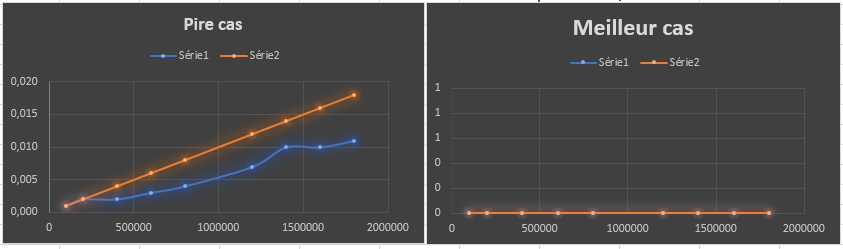
**f(n) = 4n + 4**

**Au meilleur cas :**

**f(n) = 1 + 1 + 1 + 1 + 1**

**f(n) = 5**

****

****

**2) Recherche tab triés.**

**void rechEletsTabTries(int \*tab,int n,int value) {**

**int i = 1; // 1**

**if( (tab[i] <= value) && (tab[n] >= value) ) // 3**

**{ while( i <= n && value <= tab[i] ) // 2\*(n + 1) or 1**

**{**

**if( tab[i] == value ) // n or 1**

**break; // 1**

**i++; // 2\*n - 1 or 0**

**}**

**if( i > n ) // 1**

**printf("This value doesn't exists : value = %d \n",value); // 1**

**else**

**printf("This value exists : value = %d \n",value); // 1**

**} else {**

**printf("This value doesn't exists : value = %d \n",value); // 1**

**}**

**}**

**Complexité théorique :**

**Au pire cas :**

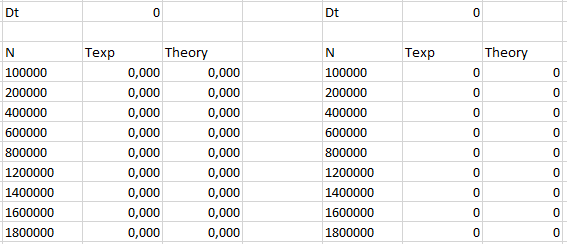
**f(n) = 1 + 3 + n + 1 + 2\*(n + 1) + 2\*n - 1 + 1 + 1**

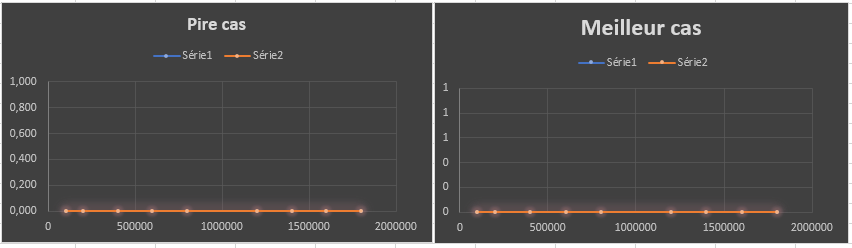
**f(n) = 6n + 9**

**Au meilleur cas :**

**f(n) = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1**

**f(n) = 9**

****

****

**3) Recherche tab triés (dichotomy).**

**void rechEletsDicho(int \*tab,int sup,int value) {**

**int inf = 1,mil; // 1**

**if( (tab[inf] <= value) && (tab[sup] >= value) ) // 3**

**{**

**while( inf < sup ) // 1**

**{**

**mil = (inf + sup)/2; // 2**

**if(tab[mil] == value) { // 1**

**break;**

**} else if( value > tab[mil] ) {**

**inf = mil + 1;**

**} else {**

**sup = mil - 1;**

**}**

**}**

**if( inf > sup ) // 1**

**printf("This value doesn't exists : value = %d \n",value); // 1**

**else**

**printf("This value exists : value = %d \n",value); // 1**

**} else {**

**printf("This value doesn't exists : value = %d \n",value); // 1**

**}**

**}**

**Complexité théorique :**

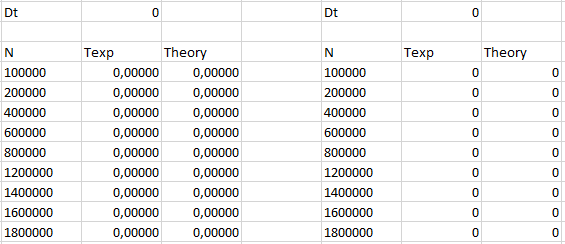
**Au pire cas :**

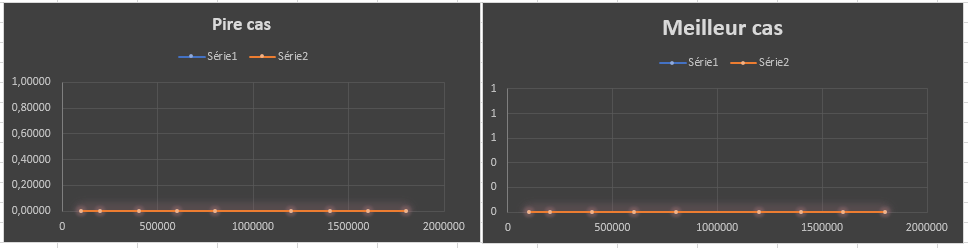
**f(n) = log(n)**

**Au meilleur cas :**

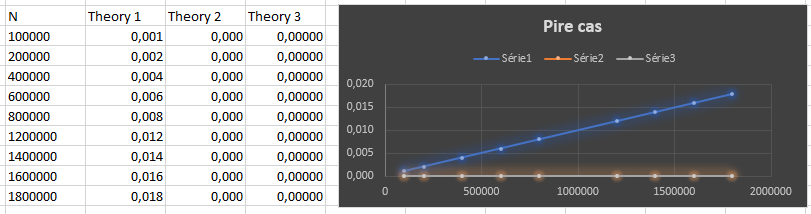
**f(n) = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1**

**f(n) = 9**

****

****

**Les 3 fonctions :**

****

**Lequel des 3 algorithme est meilleur ? on remarque, d’après les graphes, que Dichotomy est meilleur.**

**Recherche du maximum et du minimum :**

**1) Recherche Max et Min d’un ensemble non triés.**

**void maxEtMinA(int \*tab,int n) {**

**int i = 0,cpt = 0,max = tab[0],min = tab[0]; // 3**

**while( i < n ) // n + 1 or 2**

**{ if( tab[i] >= max ) // n or 1**

**{ cpt++;**

**max = tab[i]; // n or 1**

**}**

**if( tab[i] <= min ) // n or 1**

**{ cpt++;**

**min = tab[i]; //n or 1**

**}**

**i++; // 2\*n or 2**

**}**

**printf("Max = %d et Min = %d / Counting = %d \n",max,min,cpt); // 1**

**}**

**Complexité théorique :**

**Au pire cas :**

**f(n) =** **7n + 5**

**2) Recherche Max et Min d’un ensemble non triés ( plus efficace ).**

**void maxEtMinB(int \*tab,int n) {**

**int i = 0,cpt = 0,x = 0,max,min; // 3**

**while( i < n-1 ) // n**

**{**

**if( tab[i] > tab[i+1] ) // n or 1**

**{**

**cpt++;**

**x = tab[i];**

**tab[i] = tab[i+1];**

**tab[i+1] = x;**

**}**

**i = i + 2; // 2\*n or 2**

**}**

**min = tab[0];**

**i = 2;**

**while( i < n ) // n/2-1**

**{**

**if( min > tab[i] ) // n or 1**

**{**

**cpt++;**

**min = tab[i];**

**}**

**i = i + 2; // 2\*n or 2**

**}**

**min = tab[1];**

**i = 3;**

**while( i < n ) // n/2-1**

**{**

**if( max < tab[i] ) // n or 1**

**{**

**cpt++;**

**max = tab[i];**

**}**

**i = i + 2; // 2\*n or 2**

**}**

**if(n%2 == 1)**

**{**

**if( max < tab[n-1] ) // n or 1**

**{**

**cpt++;**

**max = tab[n-1];**

**}**

**}**

**printf("Max = %d et Min = %d / Counting = %d \n",max,min,cpt); // 1**

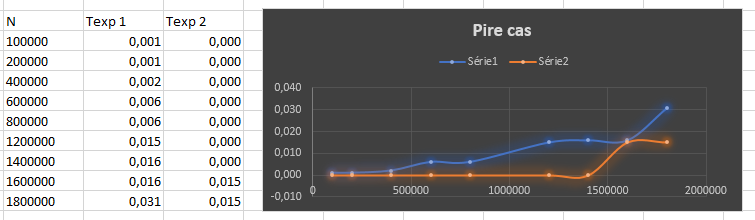
**}**

**Complexité théorique :**

**Au pire cas :**

**f(n) = 3n/2**

**Comparaison :**

****