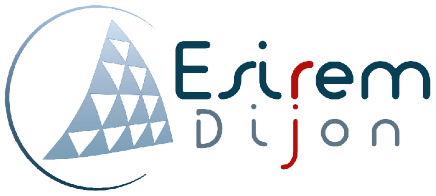

Ecole Supérieure d'Ingénieurs de Recherche en Matériaux et en Infotronique

ITC313

Compte rendu de TP de C: Comparaison d'algorithmes de tris de données

Soumis à :
Dominique Ginhac
Professeur de génie électrique
Univ. Bourgogne Franche-Comté

Par :
Wilfried L. Bounsi
& Ulrich Fonkoue
IT3A TD2 TP3

version du 22 janvier 2019

Table des matières

1 Mot sur l'évaluation des performances **2**
1.1 Principe . . . 2

2 Tri des dates **2**
2.1 Tri Basique . . . 2
2.1.1 Principe . . . 2
2.1.2 Code . . . 2
2.1.3 Analyse de la complexité . . . 3
2.1.4 Etude des performances . . . 3
2.2 Tri par Sélection . . . 11
2.3 Tri à Bulle . . . 11
2.3.1 Principe . . . 11
2.3.2 Code . . . 11
2.3.3 Analyse de la complexité . . . 11
2.3.4 Etude des performances . . . 12
2.4 Tri par Insertion . . . 20
2.4.1 Principe . . . 20
2.4.2 Code . . . 20
2.4.3 Analyse de la complexité . . . 20
2.4.4 Etude des performances . . . 21
2.5 Tri Rapide . . . 29
2.5.1 Principe . . . 29
2.5.2 Code . . . 29
2.5.3 Analyse en complexité . . . 29
2.5.4 Etude des performances . . . 31
2.6 Tri Fusion . . . 39
2.6.1 Principe . . . 39
2.6.2 Code . . . 39
2.6.3 Analyse en complexité . . . 40
2.6.4 Etude des performances . . . 40

3 Comparaison des courbes de performances des algorithmes de tri **48**
3.1 Tableaux triés dans l'ordre croissant . . . 48
3.2 Tableaux désordonnés . . . 50
3.3 Tableaux triés dans l'ordre décroissant . . . 52

Introduction

Nous nous proposons dans ce TP d'étudier 5 méthodes pour trier un tableau. Nos tableaux contiendront des **"évènements"** qui sont en réalité un type de données personnalisé dont la caractéristique la plus utile dans notre contexte est de détenir une date grâce à laquelle deux évènements pourront être comparés, nous y reviendrons en détaille par la suite.
Plus précisément, il s'agit pour nous de comprendre le principe de fonctionnement de chaque algorithme, d'écrire le programme correspondant en C, d'analyser leur complexité, d'évaluer empiriquement leurs performances selon différents critères et enfin de les comparer entre eux sur la base des performances observées. Nous en profiterons aussi pour comparer pour chacun d'entre eux, les courbes de performances obtenues aux prévisions théoriques.

Nous précisons qu'aux 5 algorithmes de tri proposés dans l'énoncé du TP, nous ajoutons l'algorithme de **tri fusion**, que nous implémentons et dont nous étudions les performances au même titre que les 5 autres.

1 Mot sur l'évaluation des performances

1.1 Principe

Nous allons récupérer et enregistrer dans un fichier, les performances de chacun de nos algorithmes de tri sur une instance de problème donnée.

Ce que nous ferons concrètement c'est exécuter chacun des algorithmes implémenté sur des tableaux de taille variant de **50** à **1400** et ce pour chacun des 3 cas trié croissant, désordonné et trié décroissant. Aussi nous prendrons systématiquement un nombre d'itérations fixé à **100** pour le cas particulier désordonné

Les performances ainsi calculées, seront enregistrés dans un fichier ***.csv** à partir du quel, d'une part nous dresserons un tableau récapitulatif et d'autre part nous tracerons des courbes de performances au moyen d'un script écrit en langage **octave** qui sera chargé d'importer les valeurs contenues dans le fichier *.csv généré par notre programme. Ce tableau et ces courbes présenterons le temps moyen d'exécution par itération, l'écart type du temps d'exécution par itération, le nombre de comparaisons, et nombre de permutations effectués par chaque algorithme en fonction de la taille du tableau considéré.

2 Tri des dates

2.1 Tri Basique

2.1.1 Principe

Ce tri consiste tout d'abord à chercher parmi tous les évènements du tableau celui qui a la plus ancienne date et le remplace avec celui qui était à la première place du tableau et enfin on reproduit ce processus sur les évènements restants jusqu'à atteindre la dernière place du tableau.

2.1.2 Code

```
1  /**
2   * Name : basic_sort
3   * @params : un tableau d'évenements {events[]} de taille {n}
4   * Description : tri le tableau selon le principe du tri basic, qui n'est en fait que
        le tri par selection.
5   */
6  void basic_sort(event events[], int n){
7    int i,j,min_index;
8    for(i = 0; i < n ; i++){
9      min_index = min_array(events, n, i);
10     if(min_index != i)
11       swap_events(events,min_index,i);
12   }
13 }
```

Listing 1 – fonction basic_sort

2.1.3 Analyse de la complexité

- **Cas favorable**
  Ici le tableau est déjà trié. Ainsi on effectue n passage dans la fonction **min_array** mais jamais suivi d'une permutation
  — Le nombre total de comparaison est $C(n) = \sum_{i=0}^{n-1} n - i = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$
  — Le nombre total de permutation est $P(n) = 0$
  — La complexité temporelle dans ce cas est $T(n) = \Theta(n^2)$
- **Cas défavorable**
  Il survient lorsque le tableau est trié à l'envers. Dans ce cas :
  — Le nombre total de comparaison est $C(n) = \Theta(n^2)$ (de valeur exacte celle du cas favorable)
  — Le nombre total de permutation est $P(n) = \Theta(n)$
  — La complexité temporelle dans ce cas est $T(n) = \Theta(n^2)$

2.1.4 Etude des performances

- **Tableaux récapitulatifs**

Tableaux triés dans l'ordre croissant

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	197	143.314340	0	1225
100	780	701.444224	0	4950
150	1643	1597.485837	0	11175
200	2321	2230.479769	0	19900
250	3293	3230.490365	0	31125
300	4855	4789.493188	0	44850
350	10695	10619.496457	0	61075
400	11175	11016.492818	0	79800
450	15283	15121.494668	0	101025
500	18750	18528.494029	0	124750
550	16258	16044.493354	0	150975
600	18356	18237.496758	0	179700
650	21329	21200.496975	0	210925
700	25661	25530.497449	0	244650
750	50408	50256.498495	0	280875
800	39616	39318.496220	0	319600
850	36482	36306.497587	0	360825
900	47888	47716.498206	0	404550
950	45562	45383.498036	0	450775
1000	50346	50159.498143	0	499500
1050	55574	55366.498128	0	550725
1100	64949	64740.498392	0	604450
1150	81083	80857.498607	0	660675
1200	73188	72943.498326	0	719400
1250	78910	78650.498352	0	780625
1300	101518	101268.498769	0	844350
1350	92718	92461.498614	0	910575
1400	101076	100813.498699	0	979300

Tableaux désordonnés

Taille des données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	146	135.605199	4600	122500
100	647	627.387982	9200	495000
150	1222	1196.017604	14600	1117500
200	2044	2005.113179	19300	1990000
250	3344	3306.219972	24146	3112500
300	4937	4890.952002	29300	4485000
350	7540	7477.630726	34535	6107500
400	8367	8300.181876	39221	7980000
450	10276	10189.047323	44543	10102500
500	15223	15125.756164	49353	12475000
550	15856	15750.412163	54433	15097500
600	18428	18311.316159	59452	17970000
650	22003	21907.833134	64555	21092500
700	27964	27969.719066	69104	24465000
750	34445	34322.736277	74499	28087500
800	32044	31884.169808	79324	31960000
850	37245	37142.984147	84294	36082500
900	47844	47682.515595	89104	40455000
950	45771	45641.247436	94156	45077500
1000	50327	50136.325222	99274	49950000
1050	55331	55117.742553	104200	55072500
1100	60346	60132.879237	109197	60445000
1150	77911	77711.014299	114229	66067500
1200	75694	75560.298609	119399	71940000
1250	82597	82418.710105	124224	78062500
1300	102611	102439.983142	129270	84435000
1350	92376	92121.116979	134111	91057500
1400	102659	102548.839486	139292	97930000

Tableaux triés dans l'ordre décroissant

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	153	133.427883	25	1225
100	679	646.475058	50	4950
150	1193	1158.485218	75	11175
200	2017	1974.489301	100	19900
250	3186	3135.491987	125	31125
300	4628	4570.493737	150	44850
350	7600	7533.495603	175	61075
400	8274	8195.495226	200	79800
450	10411	10328.496018	225	101025
500	16114	16018.497027	250	124750
550	16104	15996.496648	275	150975
600	19460	19340.496917	300	179700
650	25230	25098.497385	325	210925
700	25331	25191.497236	350	244650
750	33777	33631.497841	375	280875
800	35345	35188.497780	400	319600
850	37736	37468.496434	425	360825
900	48046	47870.498170	450	404550
950	45666	45489.498063	475	450775
1000	50430	50248.498196	500	499500
1050	55565	55351.498074	525	550725
1100	62569	62362.498346	550	604450
1150	78554	78315.498479	575	660675
1200	72277	72018.498207	600	719400
1250	78668	78438.498539	625	780625
1300	99537	99290.498760	650	844350
1350	91408	91159.498638	675	910575
1400	98907	98651.498706	700	979300

- **Courbes représentatives**

Tableaux triés dans l'ordre croissant

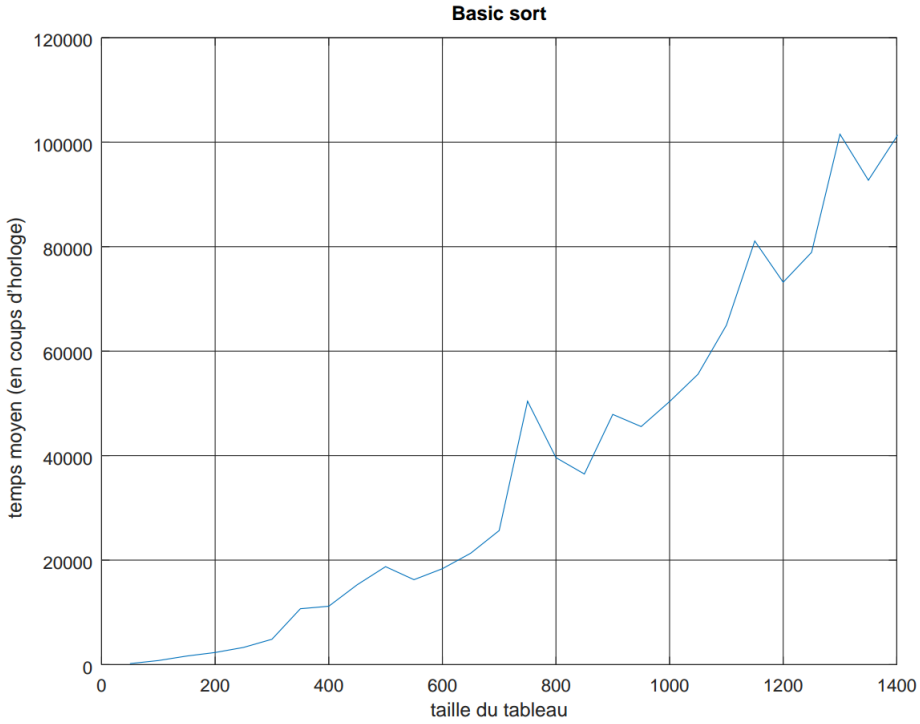

FIGURE 1 – moyenne du temps d'exécution

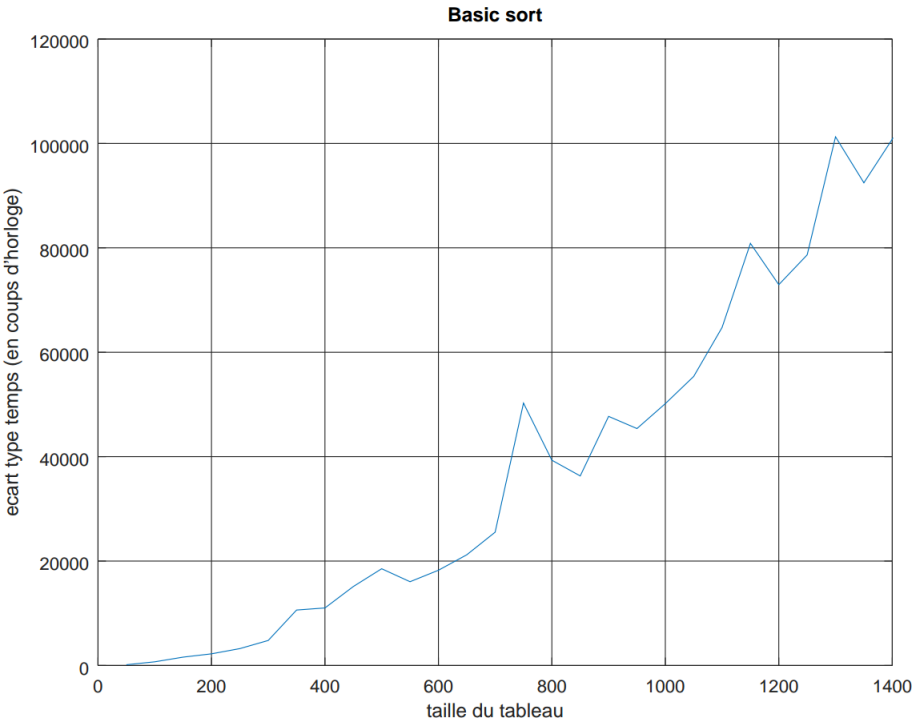

FIGURE 2 – écart type du temps d'exécution

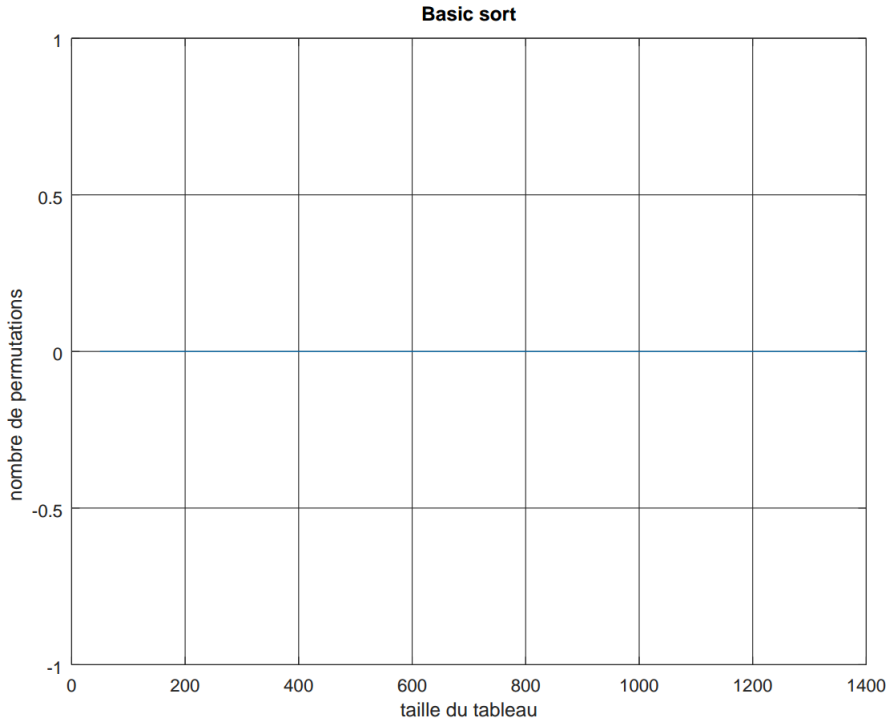

FIGURE 3 – nombre de permutations

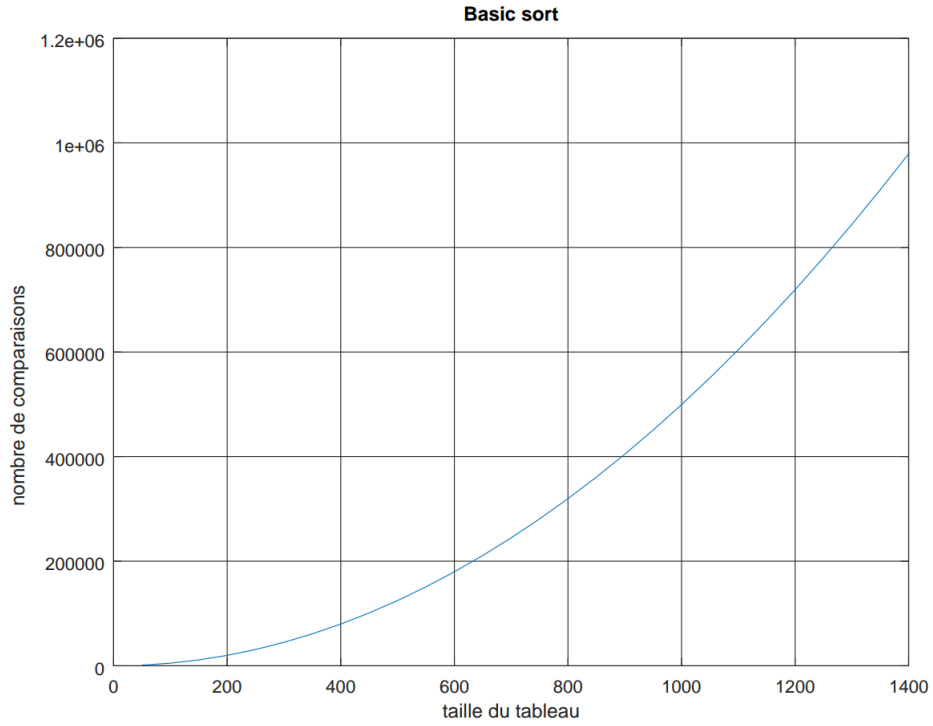

FIGURE 4 – nombre de comparaisons

Tableaux désordonnés

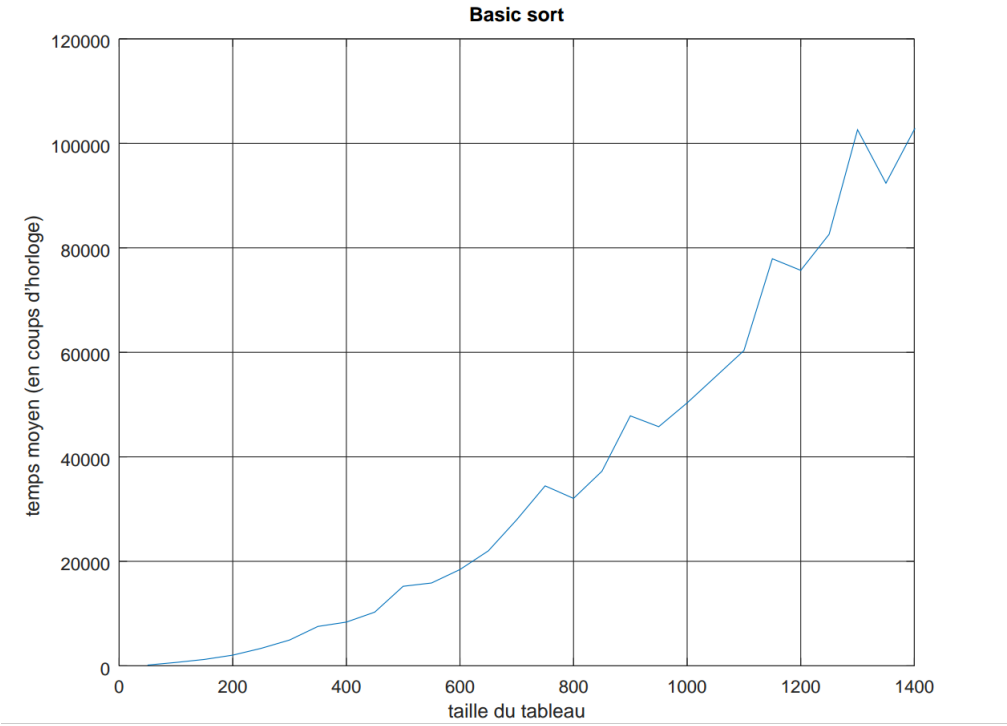

FIGURE 5 – moyenne du temps d'exécution

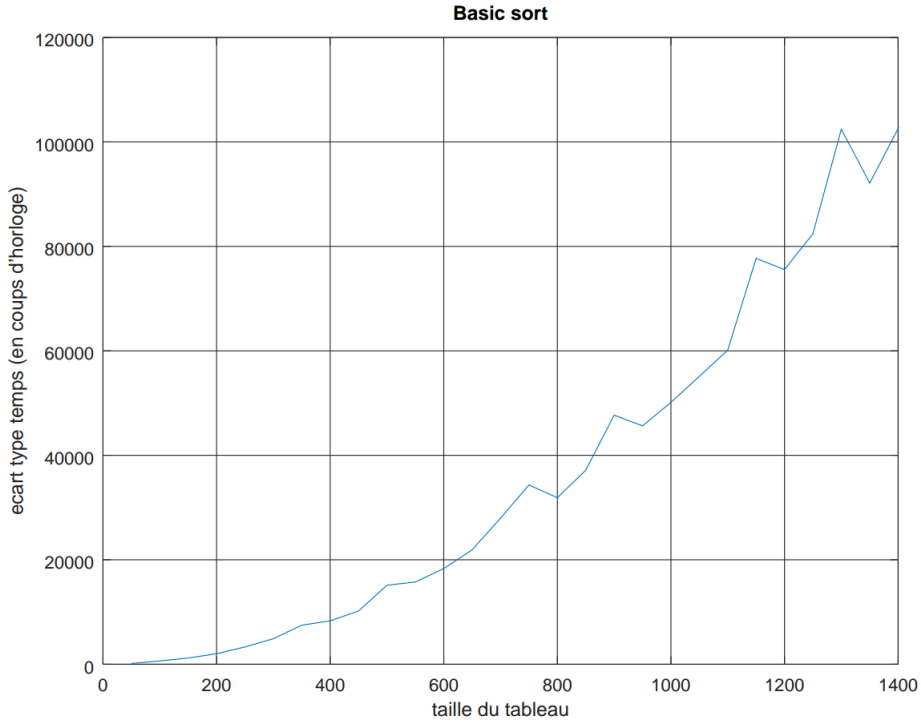

FIGURE 6 – écart type du temps d'exécution

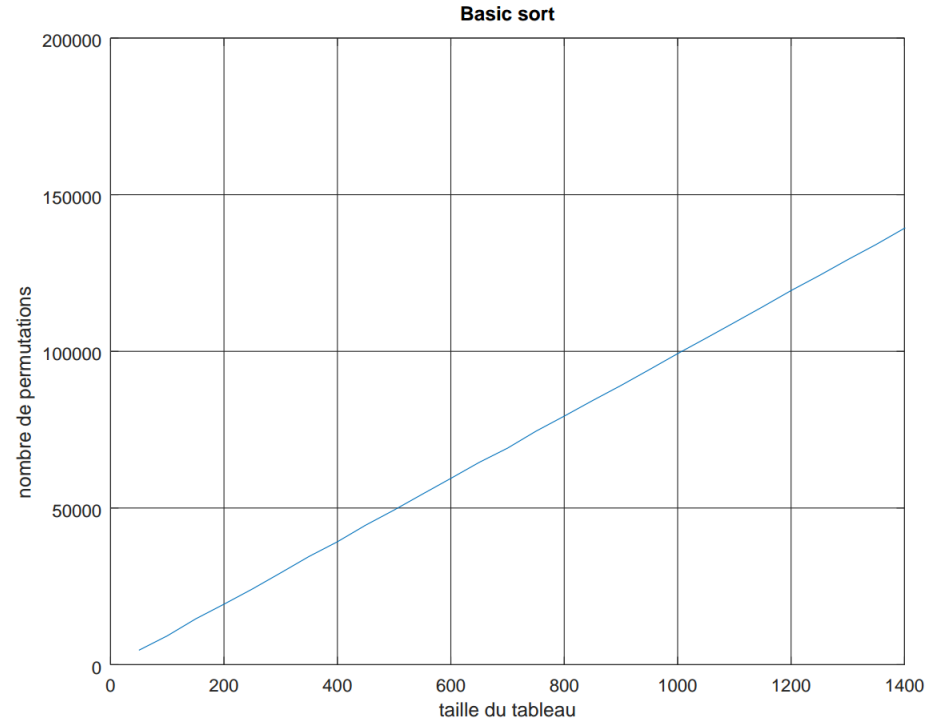

FIGURE 7 – nombre de permutations

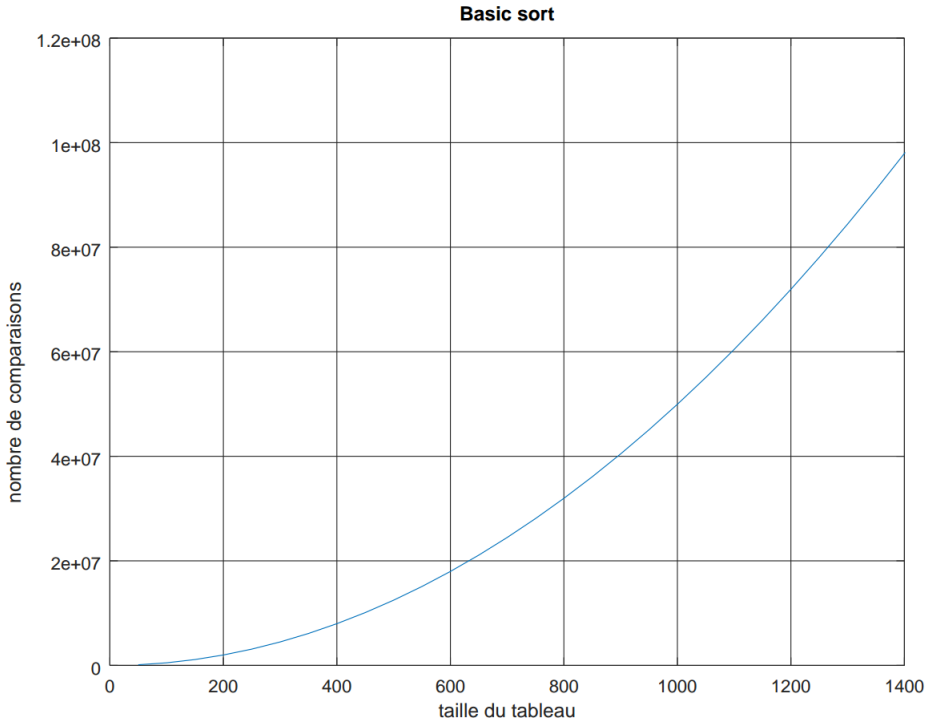

FIGURE 8 – nombre de comparaisons

Tableaux triés dans l'ordre décroissant

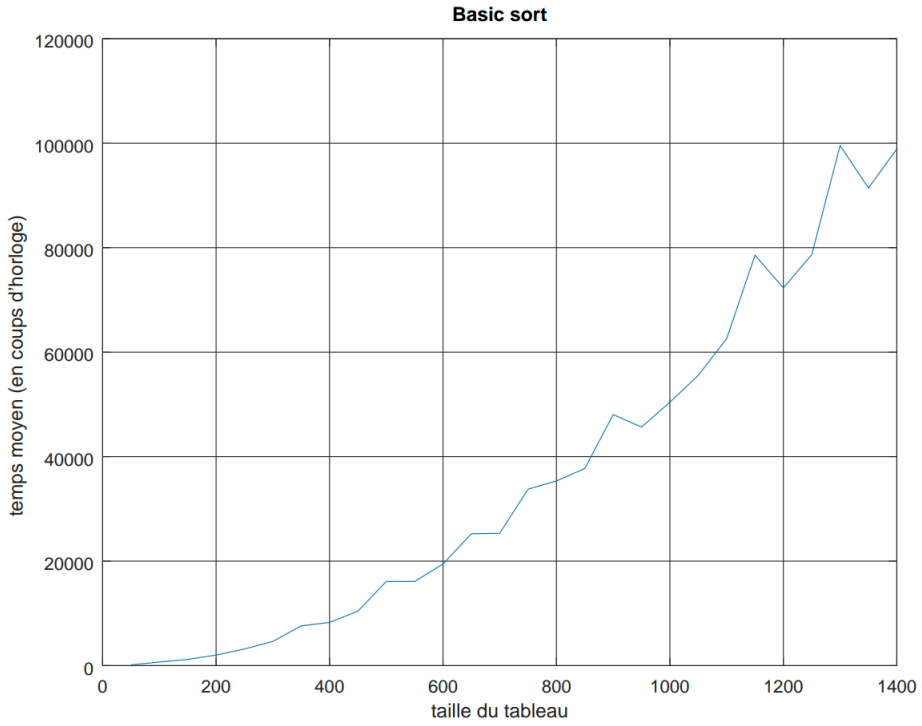

FIGURE 9 – moyenne du temps d'exécution

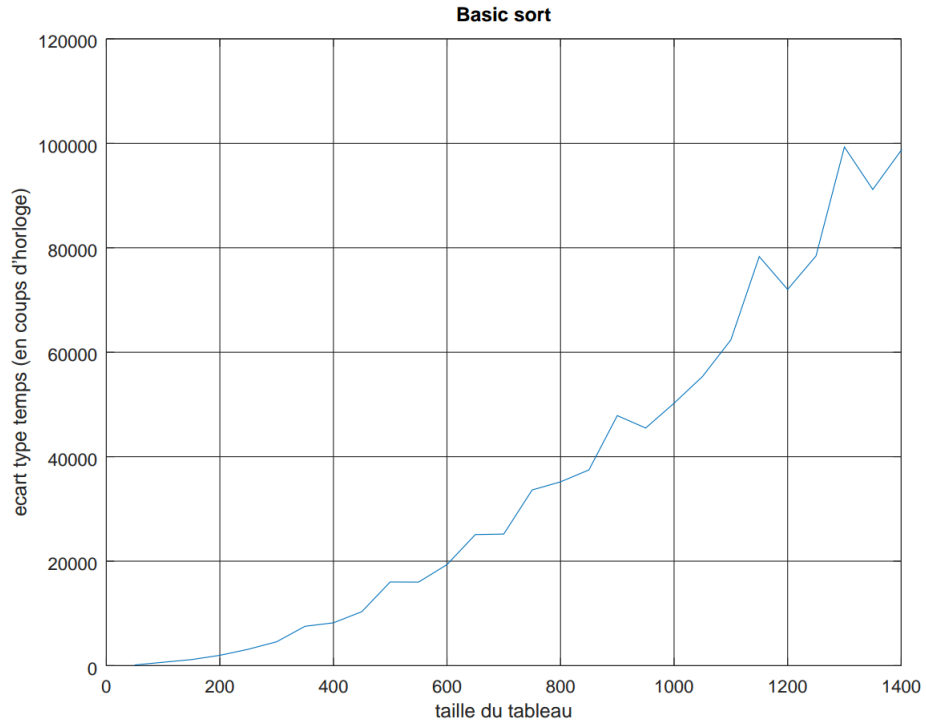

FIGURE 10 – écart type du temps d'exécution

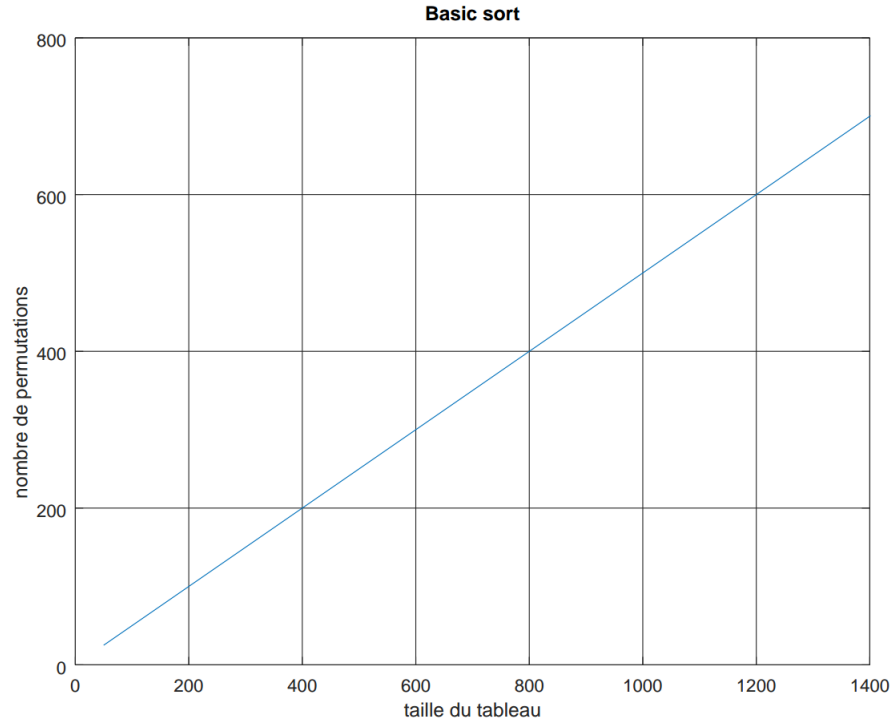

FIGURE 11 – nombre de permutations

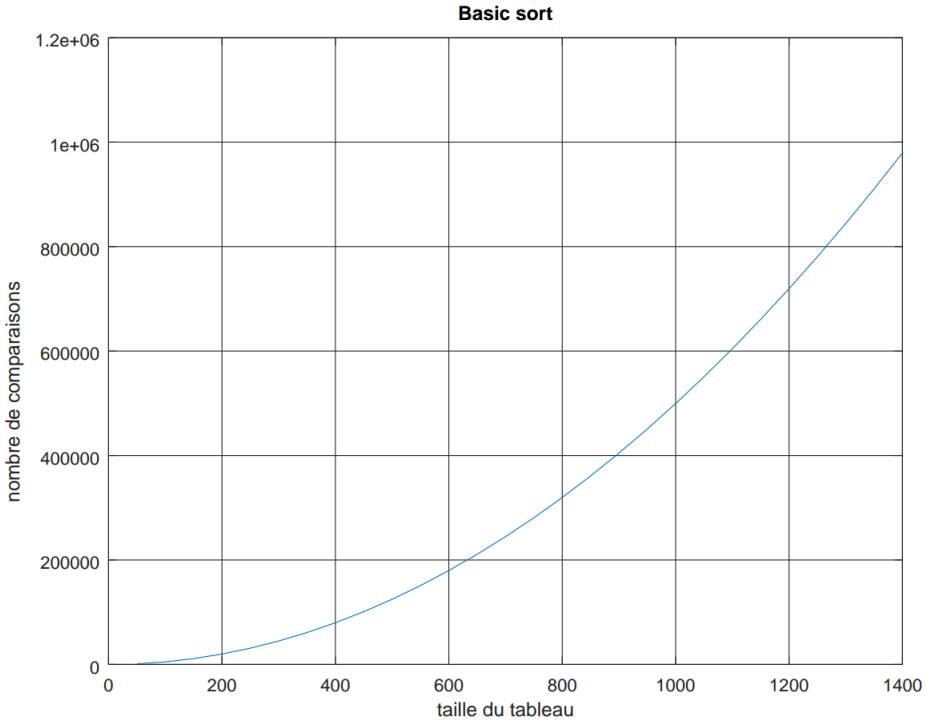

FIGURE 12 – nombre de comparaisons

Commentaire - Tri basique

Ces courbes sont conformes aux résultats théoriques. En effet, on confirme par exemple que dans le cas favorable, l'algorithme n'effectue aucune permutation et que dans tous les cas l'allure du temps moyen d'exécution est quadratique.

2.2 Tri par Sélection

Il s'agit en réalité du même principe que le tri Basique. On reprend donc le même algorithme que précédemment et bien évidement on retrouve la même complexité ainsi que les mêmes performances.

2.3 Tri à Bulle

2.3.1 Principe

Ici on parcourt tout le tableau en comparant chaque fois 2 évènements successifs au moyen de la fonction **compare_events** et lorsqu'ils ne sont pas dans l'ordre souhaité, on permute ces 2 évènements grace à la fonction **swap_events**. Cette opération sera donc reitérée jusqu'à ce que le tableau soit entièrement trié.

2.3.2 Code

```
/**
 * Name : bubble_sort
 * @params : un tableau d'évenements {events[]} de taille {n}
 * Description : tri le tableau selon le principe du tri bulle.
 */
void bubble_sort(event events[], int n){
  short int permut = 1;
  int i = 0;
  while(permut){
    permut = 0;
    for(i=1; i< n; i++){
      if(compare_events(events[i-1], events[i]) == 1 ){
        swap_events(events, i-1,i);
        permut = 1;
      }
    }
  }

}
```

Listing 2 – fonction bubble_sort

2.3.3 Analyse de la complexité

- **Cas favorable**
  Le cas favorable se présente quand le tableau est déjà trié. Ainsi on entre une seule fois dans chacune des boucles while *principale* et for *interne*. On effectue alors n comparaisons et aucune permutation. *principale*.
  — Le nombre total de comparaison est $C(n) = \sum_{i=1}^{n-1} = n - 1$
  — Le nombre total de permutation est $P(n) = 0$
  — La complexité temporelle dans ce cas est $T(n) = \Theta(n)$
- **Cas défavorable**
  Il survient lorsque le tableau est trié à l'envers. Dans ce cas :
  — Le nombre total de comparaison est $C(n) = \Theta(n^2)$
  — Le nombre total de permutation est $P(n) = \Theta(n^2)$
  — La complexité temporelle dans ce cas est $T(n) = \Theta(n^2)$

2.3.4 Etude des performances

- **Tableaux récapitulatifs**

Tableaux triés dans l'ordre croissant

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	19	2.449490	0	49
100	31	8.306624	0	99
150	41	13.564660	0	149
200	56	18.547237	0	199
250	78	28.670542	0	249
300	83	28.583212	0	299
350	97	33.585711	0	349
400	164	65.764732	0	399
450	126	43.577517	0	449
500	139	48.590122	0	499
550	151	53.609701	0	549
600	168	57.558666	0	599
650	195	74.706091	0	649
700	195	68.593003	0	699
750	228	73.464277	0	749
800	242	91.689694	0	799
850	234	83.612200	0	849
900	248	87.595662	0	899
950	259	91.596943	0	949
1000	273	97.611475	0	999
1050	320	119.670381	0	1049
1100	304	107.596468	0	1099
1150	319	112.592184	0	1149
1200	362	137.691685	0	1199
1250	343	122.609135	0	1249
1300	365	127.577427	0	1299
1350	374	130.575649	0	1349
1400	383	136.605271	0	1399

Tableaux désordonnés

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	263	251.735814	61900	215600
100	1369	1349.421076	271912	900801
150	2655	2622.955575	642100	2071100
200	4267	4230.874627	1129700	3681500
250	8518	8526.719291	1574578	5675208
300	10063	10002.317423	2231211	8344492
350	12655	12587.757109	3071340	11630076
400	20024	19949.886765	4093636	15462846
450	22116	22082.217121	4991108	18973842
500	26053	25967.511229	6212903	23834236
550	31436	31361.292010	7473020	28006686
600	39201	39114.080845	9010039	34973214
650	51488	51393.778175	10560200	40117935
700	52572	52499.873801	12283324	46679919
750	62244	62214.832925	14209939	54723438
800	80273	80188.530567	15965611	61478256
850	79487	79540.803610	18101468	69191802
900	89433	89370.106797	20131313	77830026
950	107011	108082.541568	22707685	86928400
1000	109260	109297.467120	25052928	96414489
1050	138084	138001.586079	27474914	105722416
1100	132306	132686.310324	30154207	114633393
1150	142755	142690.939131	32803314	127656198
1200	184394	184553.381896	35982242	139507247
1250	173521	173586.171724	38963300	151016590
1300	184375	184306.974699	42294608	163801302
1350	196790	196627.766202	45452211	175692411
1400	216600	216575.671760	49019653	189419004

Tableaux triés dans l'ordre décroissant

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	295	281.476464	1225	2450
100	1148	1129.491921	4950	9900
150	2554	2519.493203	11175	22350
200	4554	4514.495653	19900	39800
250	8380	8328.496923	31125	62250
300	10190	10132.497175	44850	89700
350	14003	13938.497695	61075	122150
400	21176	21097.498146	79800	159600
450	23104	23019.498170	101025	202050
500	28474	28377.498304	124750	249500
550	34574	34416.497715	150975	301950
600	40652	40531.498517	179700	359400
650	55646	55520.498872	210925	421850
700	55353	55219.498793	244650	489300
750	63382	63241.498891	280875	561750
800	83384	83209.498953	319600	639200
850	81884	81727.499044	360825	721650
900	90998	90824.499046	404550	809100
950	101211	101034.499128	450775	901550
1000	112083	111898.499177	499500	999000
1050	142337	142129.499271	550725	1101450
1100	136674	136470.499255	604450	1208900
1150	148469	148251.499267	660675	1321350
1200	185591	185360.499379	719400	1438800
1250	185587	185347.499355	780625	1561250
1300	190072	189828.499359	844350	1688700
1350	205323	205074.499395	910575	1821150
1400	219774	219515.499412	979300	1958600

- **Courbes représentatives**

Tableaux triés dans l'ordre croissant

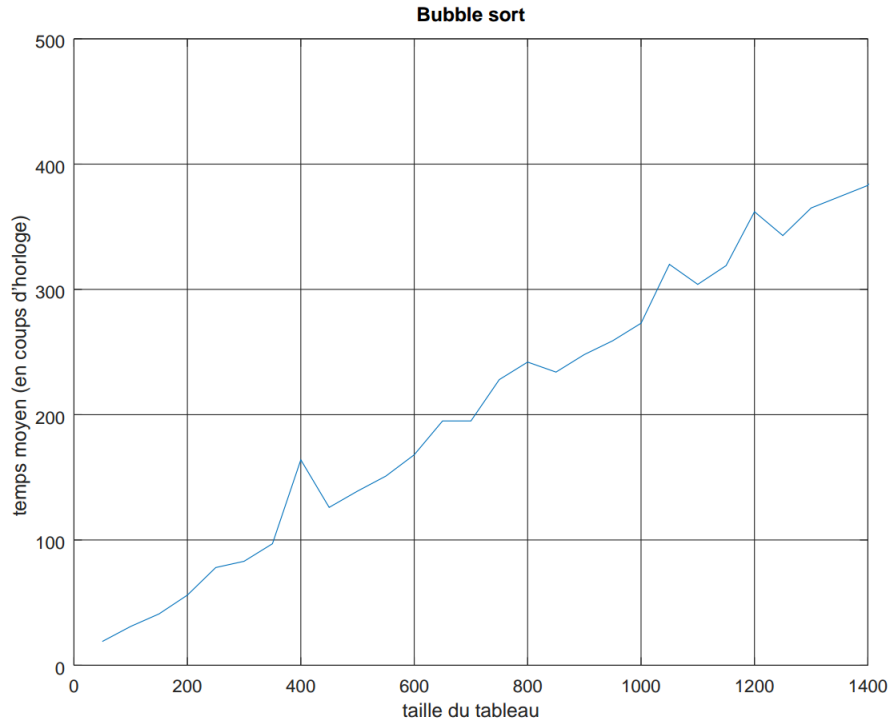

FIGURE 13 – moyenne du temps d'exécution

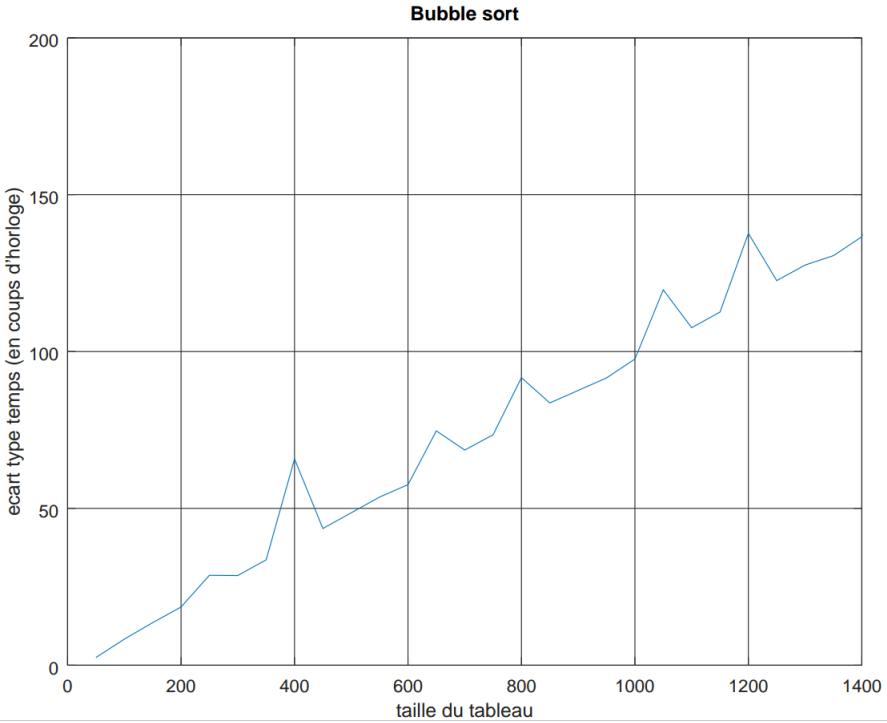

FIGURE 14 – écart type du temps d'exécution

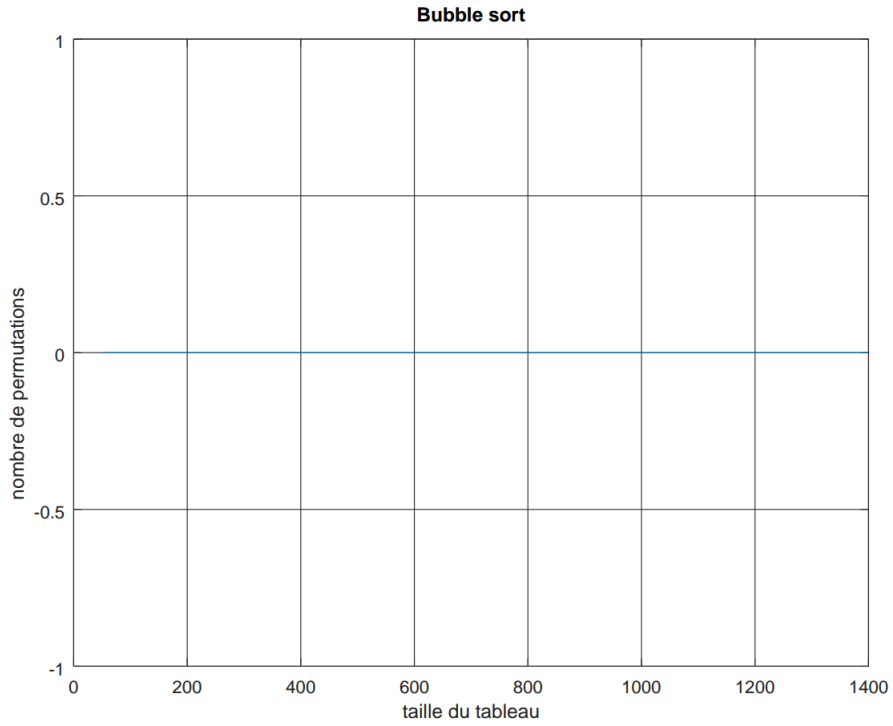

FIGURE 15 – nombre de permutations

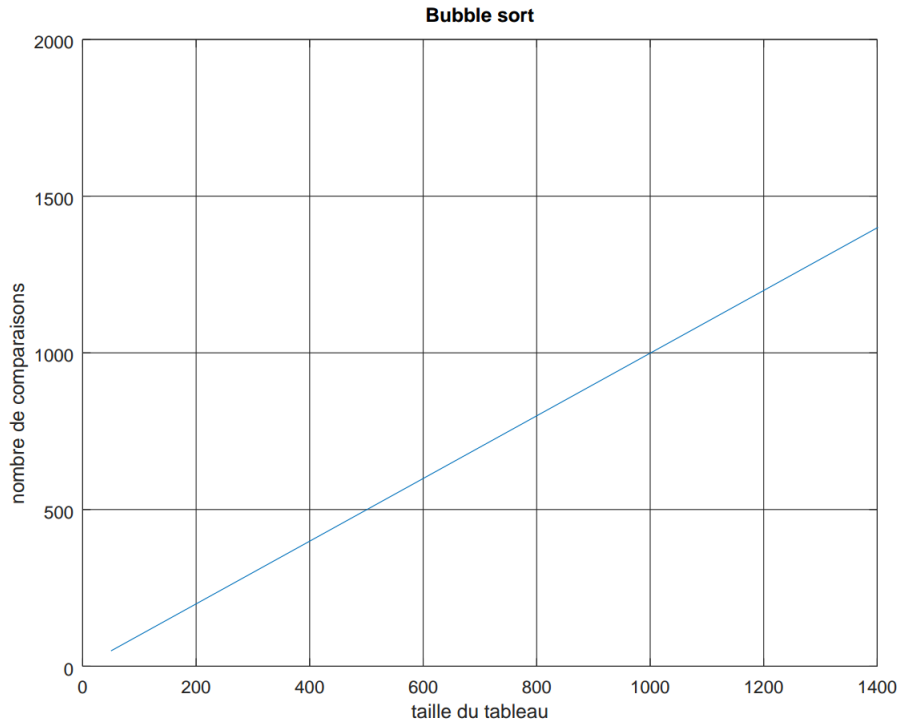

FIGURE 16 – nombre de comparaisons

Tableaux désordonnés

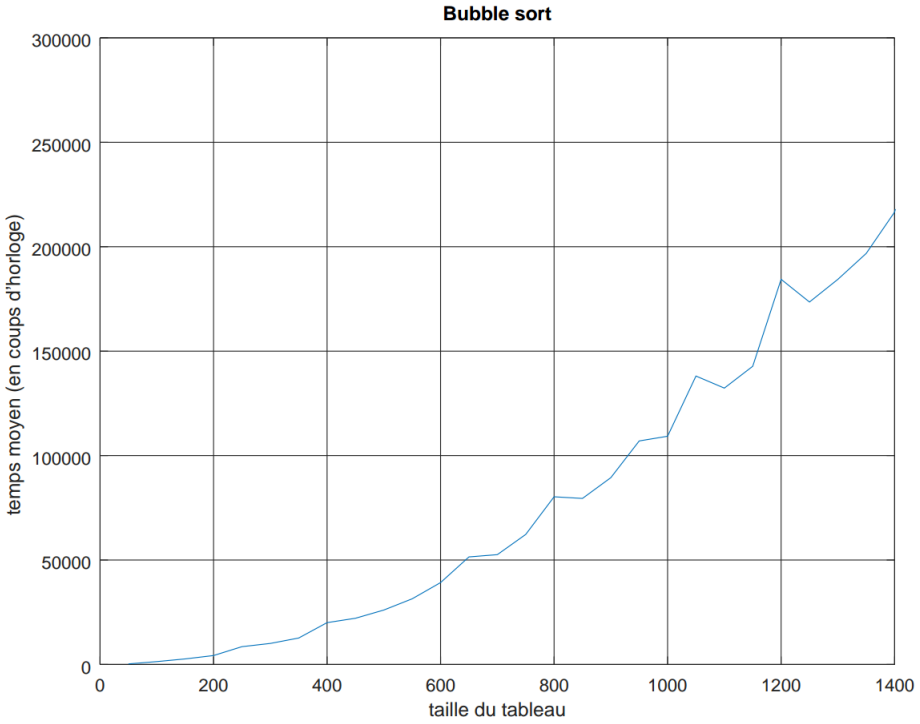

FIGURE 17 – moyenne du temps d'exécution

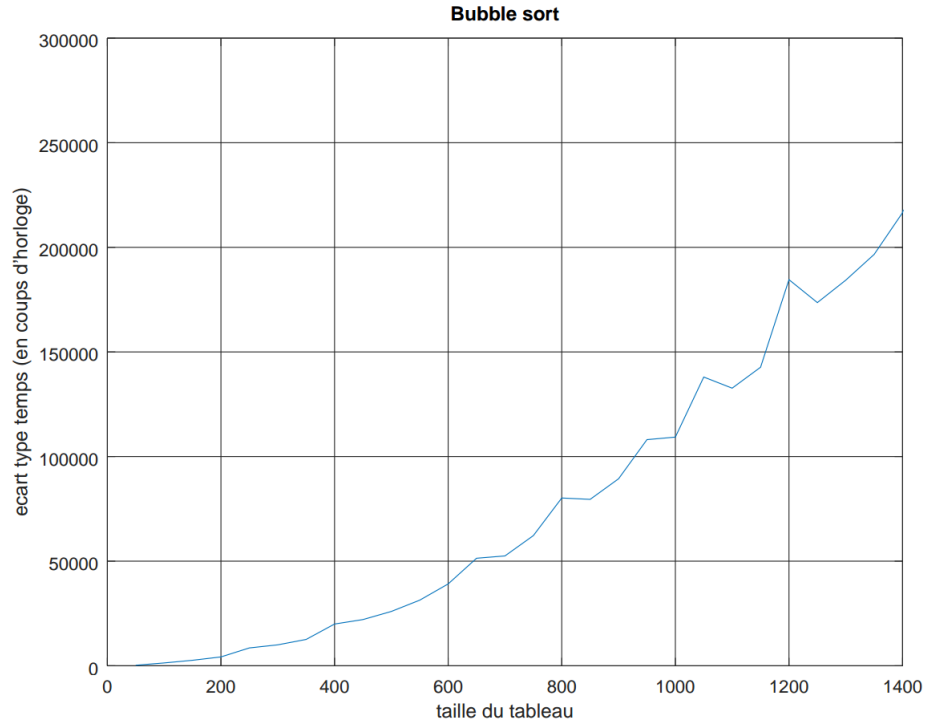

FIGURE 18 – écart type du temps d'exécution

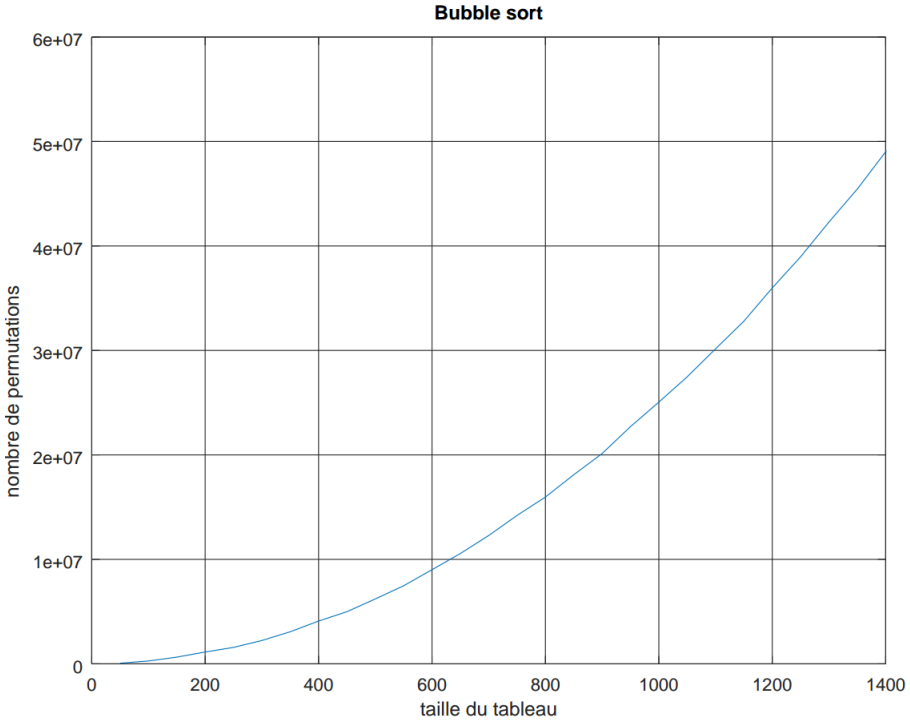

FIGURE 19 – nombre de permutations

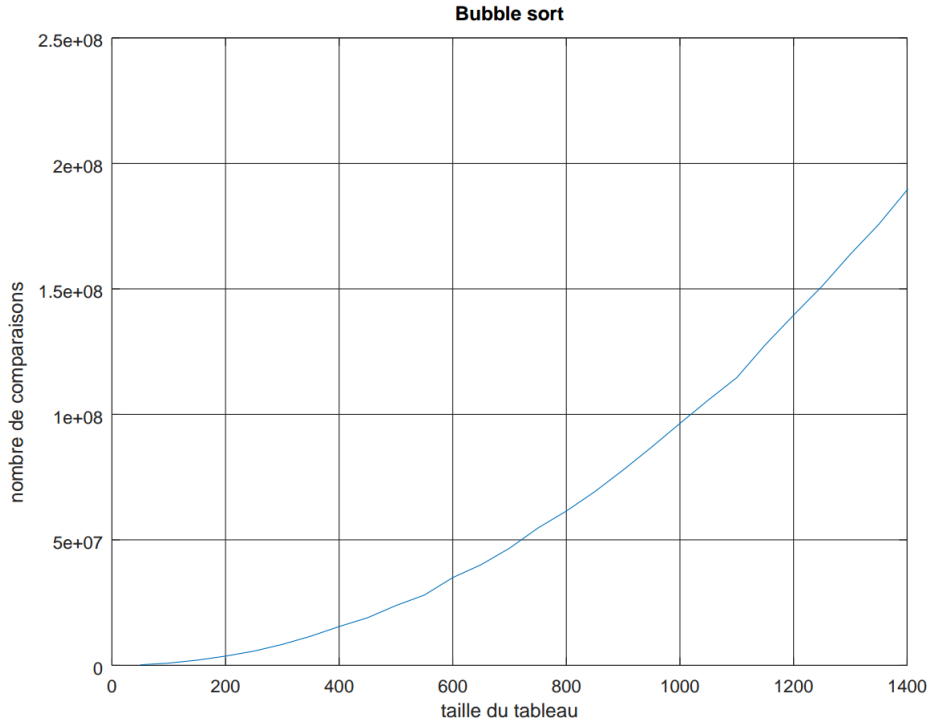

FIGURE 20 – nombre de comparaisons

Tableaux triés dans l'ordre décroissant

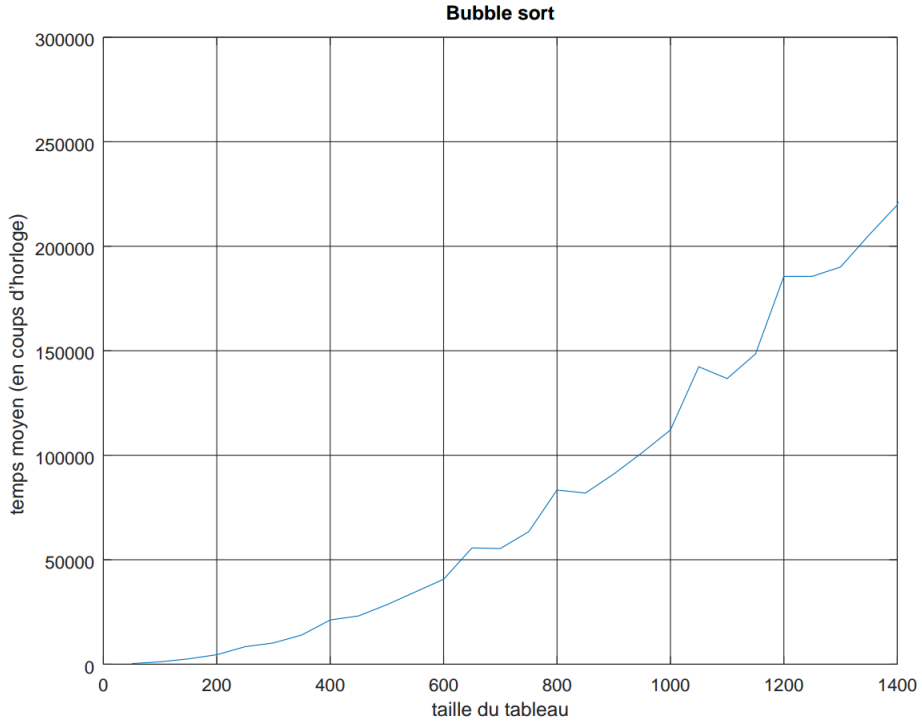

FIGURE 21 – moyenne du temps d'exécution

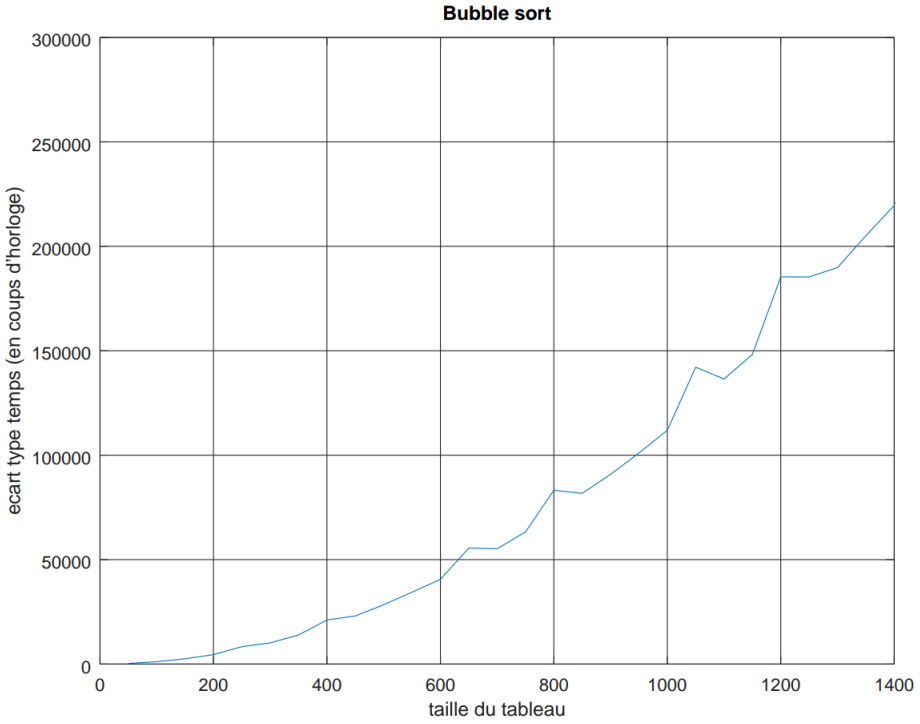

FIGURE 22 – écart type du temps d'exécution

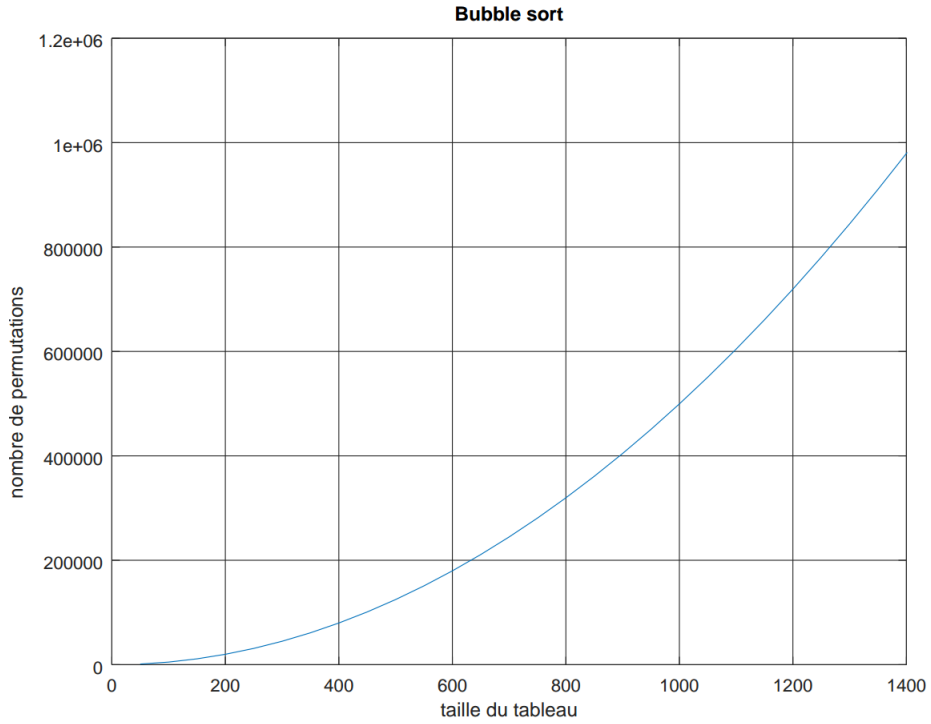

FIGURE 23 – nombre de permutations

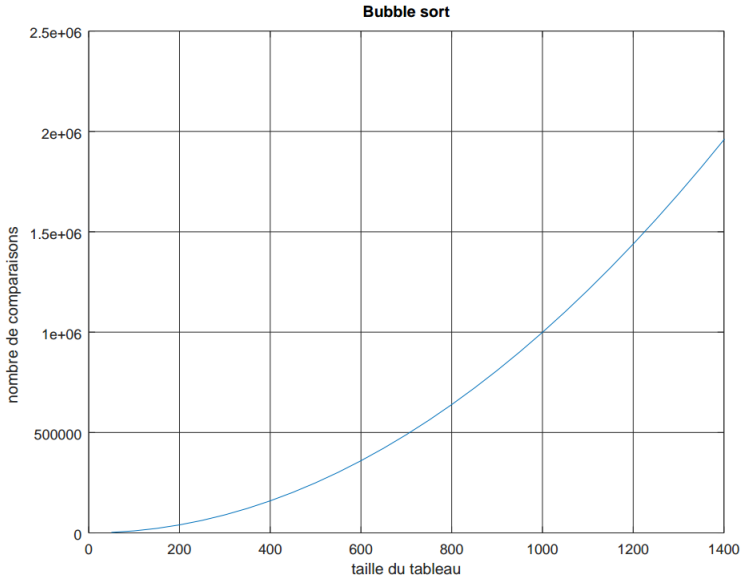

FIGURE 24 – nombre de comparaisons

Commentaire - Tri à bulle

Ces courbes sont conformes aux résultats théoriques. En effet, on confirme par exemple que dans le cas favorable, le nombre de permutation est nulle et le coût en terme de comparaison ainsi que le moyenne de temps effectué ont bien des allures linéaires. Toutefois, pour tout autre cas, les courbes correspondants à ces trois critères ont une allure quadratique ce qui est une nouvelle fois, le résultat attendu.

2.4 Tri par Insertion

2.4.1 Principe

Le tri par insertion se fait par comparison d'un évènement avec tous les autres évènements qui le précèdent dans le tableau au moyen de la fonction **compare_events** afin de trouver la place qui lui correspond et l'y insérer.

2.4.2 Code

```
/**
 * Name : insertion_sort
 * @params : un tableau d'évenements {events[]} de taille {n}
 * Description : tri le tableau selon le principe du tri insertion.
 */
void insertion_sort(event events[], int n){
  int i,j,min_index;
  event val;
  for(i = 1; i < n ; i++){
    val = events[i];
    j = i - 1;
    while(j > -1 && compare_events(events[j], val) == 1){
      events[j+1] = events[j];
      j--;
    }
    events[j+1] = val;
  }
}
```

Listing 3 – fonction insertion_sort

NB : Remarquons que cette fonction ne fait pas appel à **swap_events** car les opérations effectués dans cet algorithme sont des insertions à des places précises et non des échanges de positions.

2.4.3 Analyse de la complexité

- **Cas favorable**
  Le cas favorable se présente quand le tableau est déjà trié. Ainsi on n'entre pas dans la boucle while *interne*. On a alors une seule comparaison pour chaque passage dans la boucle pour *principale*.
  — Le nombre total de comparaison est $C(n) = \sum_{i=1}^{n-1} = n - 1$
  — La complexité temporelle dans ce cas est $T(n) = \Theta(n)$
- **Cas défavorable**
  A l'opposé du cas favorable, le cas défavorable arrive lorsque le tableau est trié à l'envers. Dans ce cas, chaque nouvelle clé examiné dans la boucle pour *principale* doit être ramenée à l'indice 0 du tableau et il faut décaler l'ensemble des clés précéemment triées.
  — Le nombre total de comparaison est $C(n) = \sum_{i=1}^{n-1}\sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1}(i-1) = \frac{n(n-1)}{2}$
  — La complexité temporelle dans ce cas est $T(n) = \Theta(n^2)$

2.4.4 Etude des performances

- **Tableaux récapitulatifs**

Tableaux triés dans l'ordre croissant

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	18	5.567764	0	49
100	30	10.677078	0	99
150	45	15.620499	0	149
200	63	24.758837	0	199
250	75	26.627054	0	249
300	87	31.654384	0	299
350	100	37.696154	0	349
400	121	43.634848	0	399
450	140	56.780278	0	449
500	144	55.722527	0	499
550	158	60.712437	0	549
600	185	78.835271	0	599
650	194	73.695319	0	649
700	202	78.727378	0	699
750	216	83.719771	0	749
800	237	89.688349	0	799
850	306	132.853303	0	849
900	262	101.720204	0	899
950	289	118.789730	0	949
1000	354	128.631256	0	999
1050	327	118.629676	0	1049
1100	316	122.719192	0	1099
1150	332	128.716743	0	1149
1200	358	134.677392	0	1199
1250	383	159.806133	0	1249
1300	381	145.698318	0	1299
1350	388	151.726728	0	1349
1400	425	178.815547	0	1399

Tableaux désordonnés

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	89	78.232794	0	62600
100	299	279.117968	0	256900
150	674	644.293124	0	597900
200	1345	1306.692703	0	1028000
250	1805	1753.898680	0	1591300
300	2561	2503.006448	0	2340868
350	3340	3272.088937	0	2913300
400	4319	4236.475568	0	3789200
450	6737	6665.020663	0	5032419
500	7165	7061.659907	0	6265504
550	8452	8353.133515	0	7605660
600	12085	12037.343148	0	9040610
650	11683	11566.661538	0	10531764
700	13828	13695.988133	0	12458792
750	15365	15233.085411	0	14002723
800	17738	17587.016195	0	16037836
850	23442	23279.627895	0	18166524
900	22575	22406.286506	0	20535169
950	25949	25808.516360	0	22747132
1000	34247	34127.712590	0	25084966
1050	30060	29902.755374	0	27365304
1100	33419	33229.859728	0	30321632
1150	35388	35170.812423	0	33494516
1200	37964	37720.937954	0	36053128
1250	48141	47912.600322	0	38777894
1300	45617	45374.711144	0	42812369
1350	49673	49566.297860	0	46196844
1400	62145	61981.448200	0	48384196

Tableaux triés dans l'ordre décroissant

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	188	170.449406	0	1225
100	556	536.482059	0	4950
150	1260	1230.488115	0	11175
200	2672	2634.492930	0	19900
250	3475	3409.490431	0	31125
300	4998	4941.494308	0	44850
350	6982	6917.495356	0	61075
400	10054	9930.493794	0	79800
450	13376	13255.495464	0	101025
500	13699	13600.496388	0	124750
550	16643	16539.496879	0	150975
600	22667	22556.497556	0	179700
650	22796	22666.497149	0	210925
700	26642	26503.497392	0	244650
750	30269	30128.497672	0	280875
800	34982	34822.497713	0	319600
850	45385	45218.498162	0	360825
900	43451	43286.498103	0	404550
950	51922	51745.498297	0	450775
1000	62395	62203.498463	0	499500
1050	65676	65470.498433	0	550725
1100	66426	66205.498337	0	604450
1150	70603	70373.498371	0	660675
1200	76468	76235.498477	0	719400
1250	98250	97985.498652	0	780625
1300	89104	88859.498626	0	844350
1350	96234	95986.498712	0	910575
1400	120746	120486.498924	0	979300

- **Courbes représentatives**

Tableaux triés dans l'ordre croissant

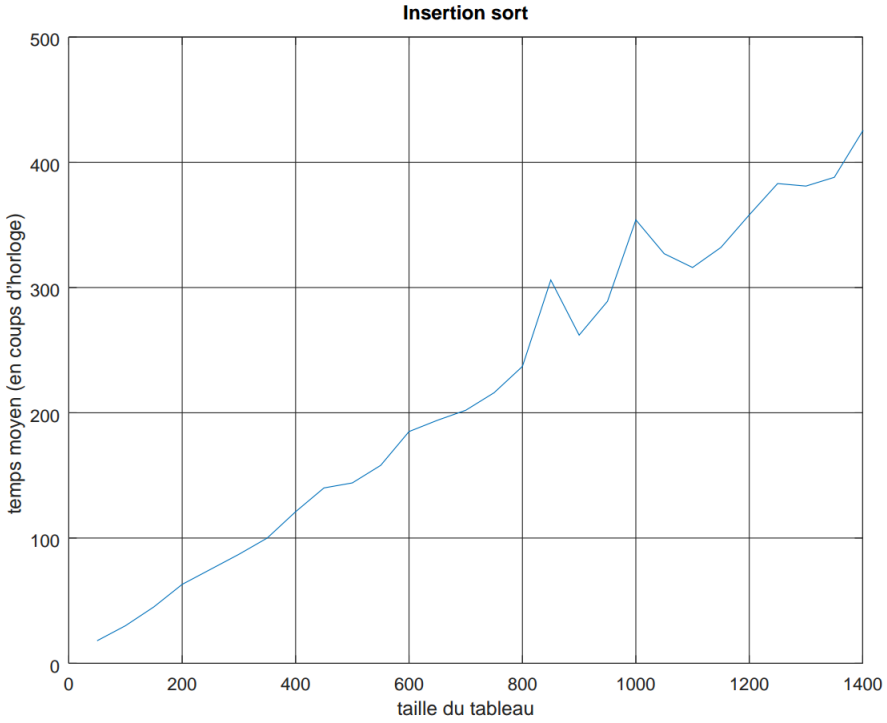

FIGURE 25 – moyenne du temps d'exécution

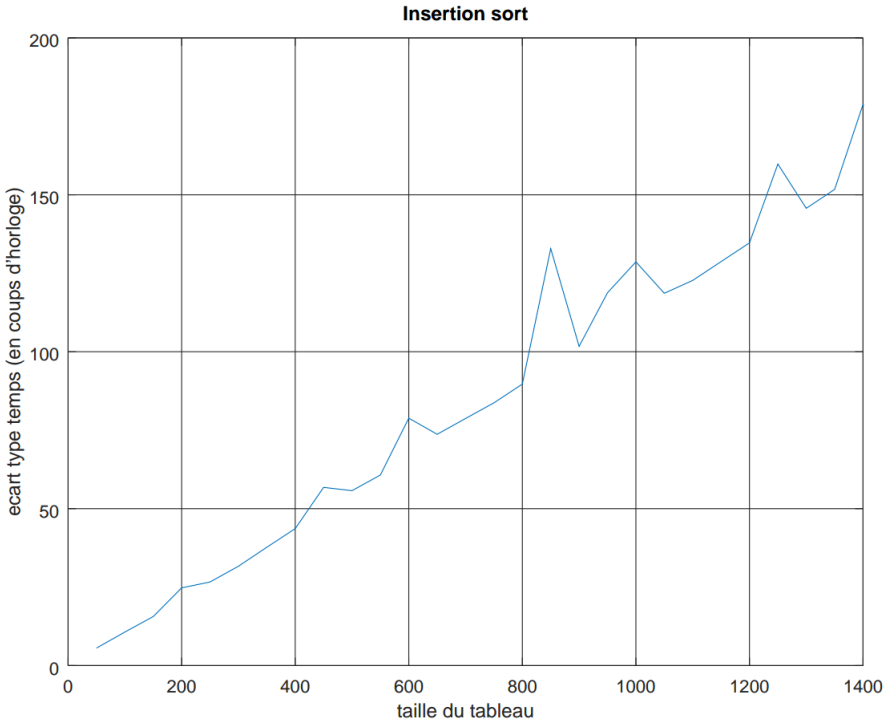

FIGURE 26 – écart type du temps d'exécution

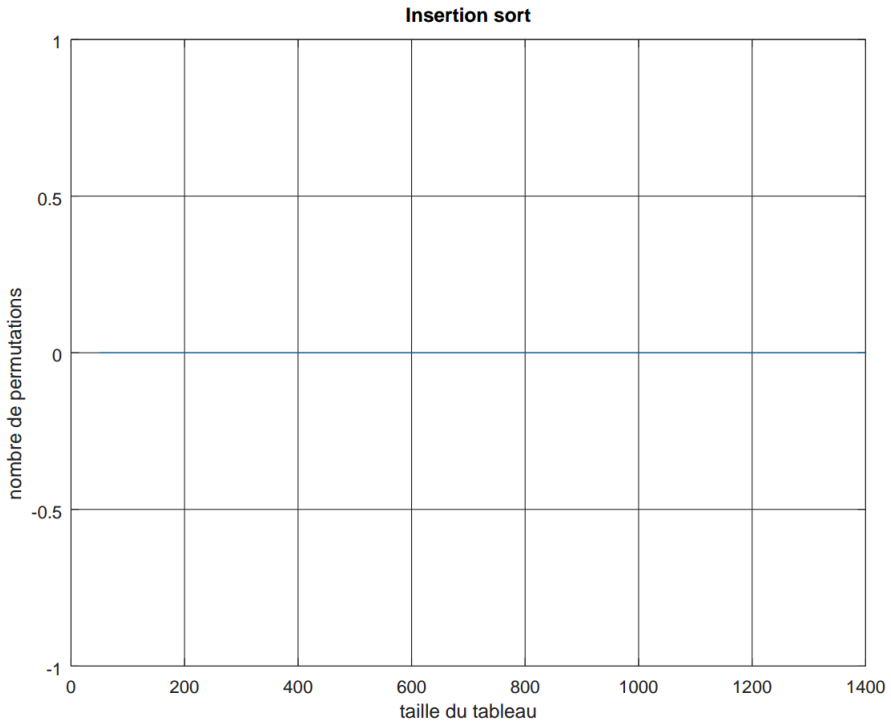

FIGURE 27 – nombre de permutations

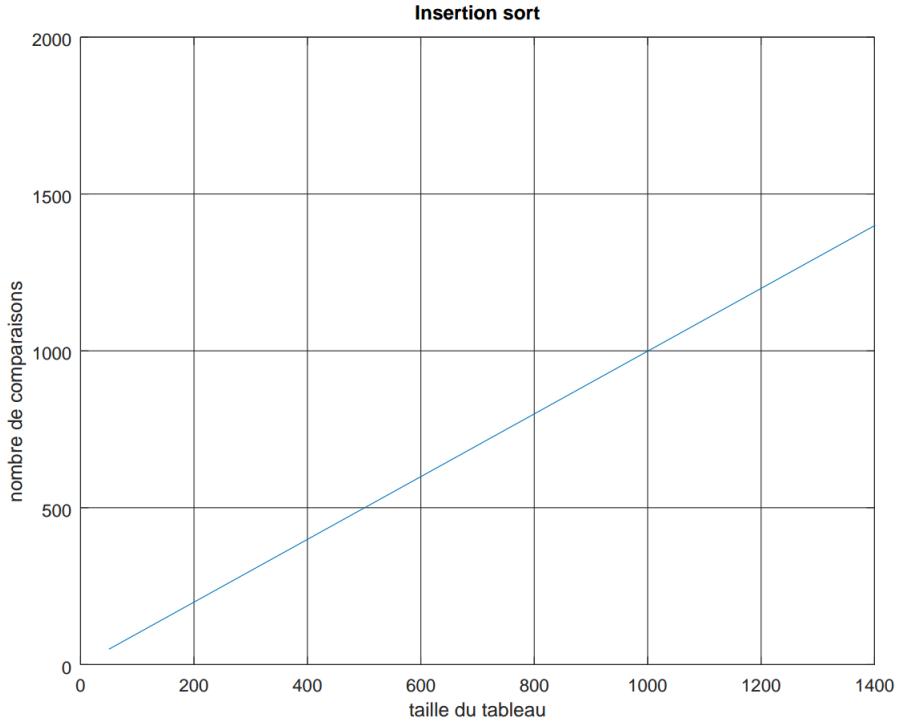

FIGURE 28 – nombre de comparaisons

Tableaux désordonnés

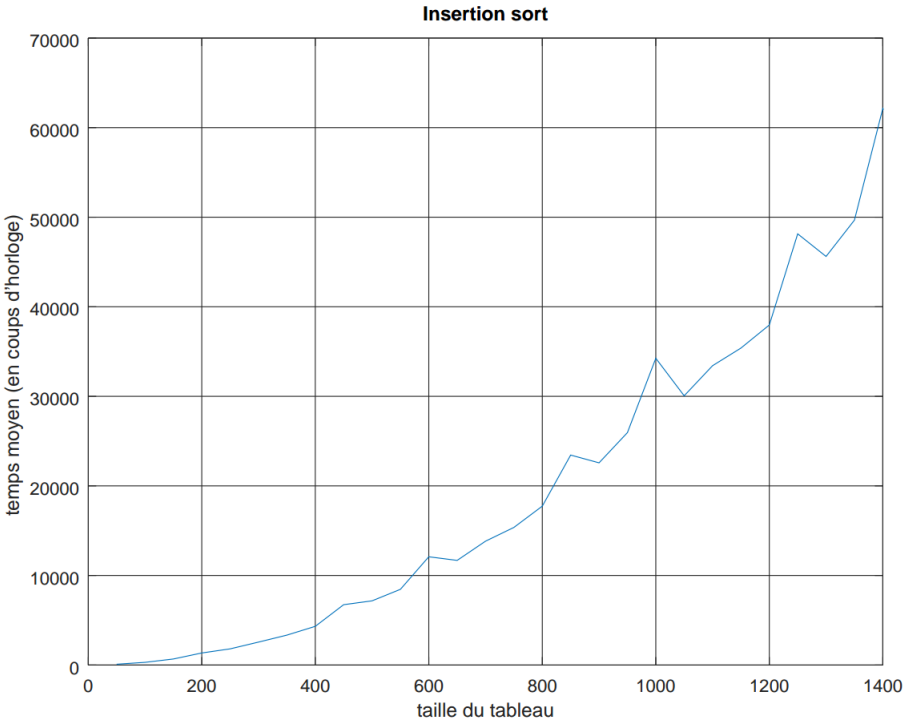

FIGURE 29 – moyenne du temps d'exécution

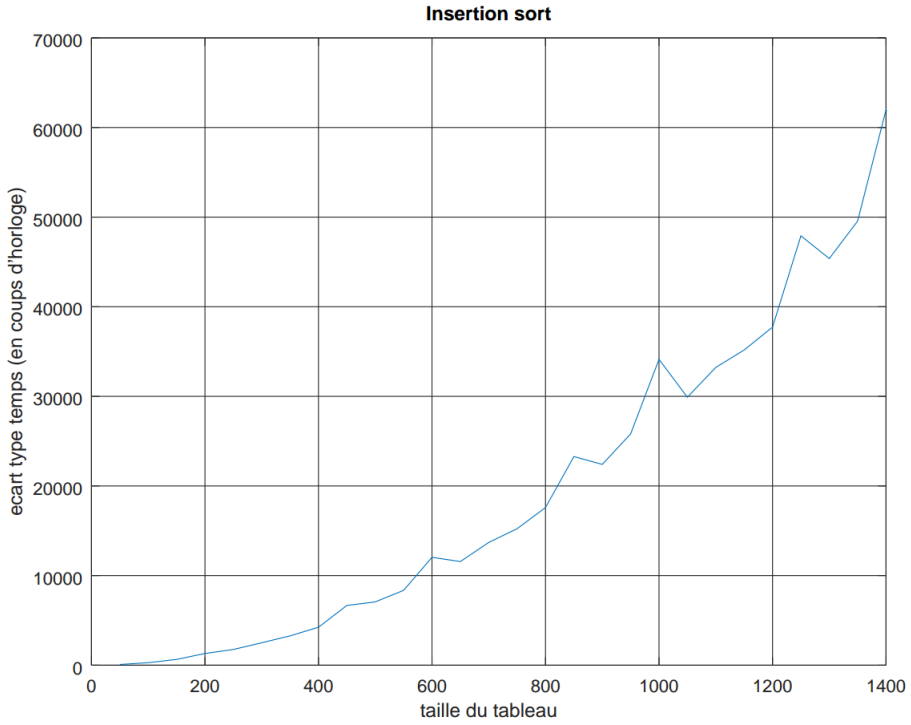

FIGURE 30 – écart type du temps d'exécution

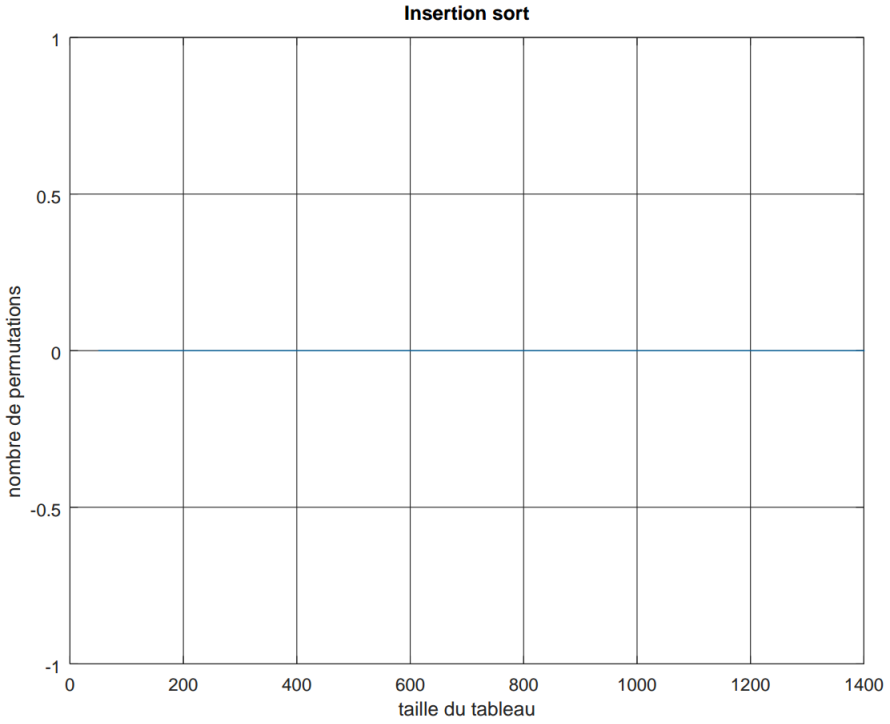

FIGURE 31 – nombre de permutations

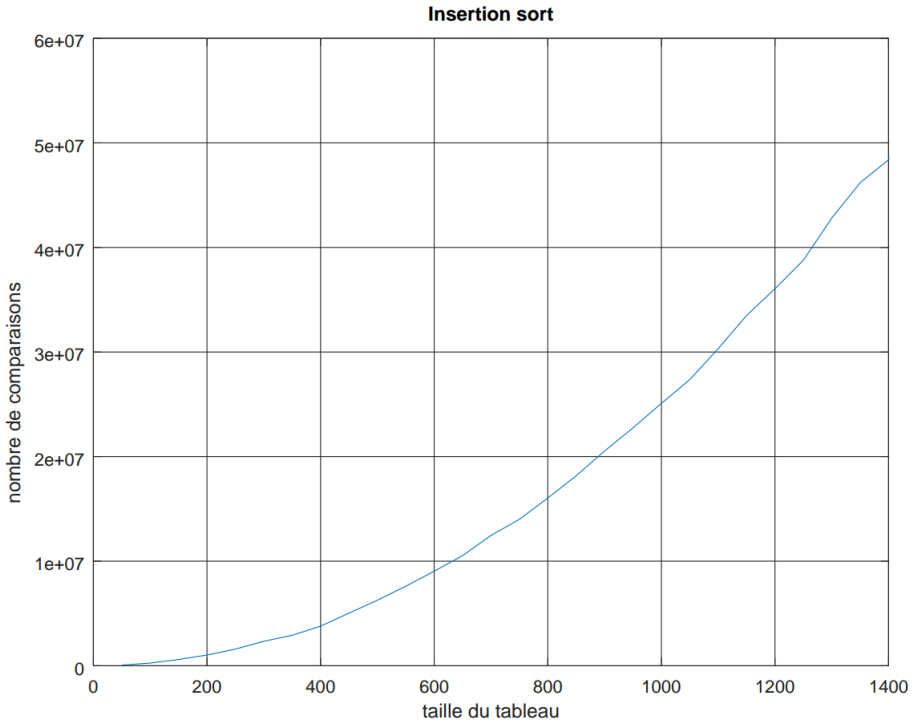

FIGURE 32 – nombre de comparaisons

Tableaux triés dans l'ordre décroissant

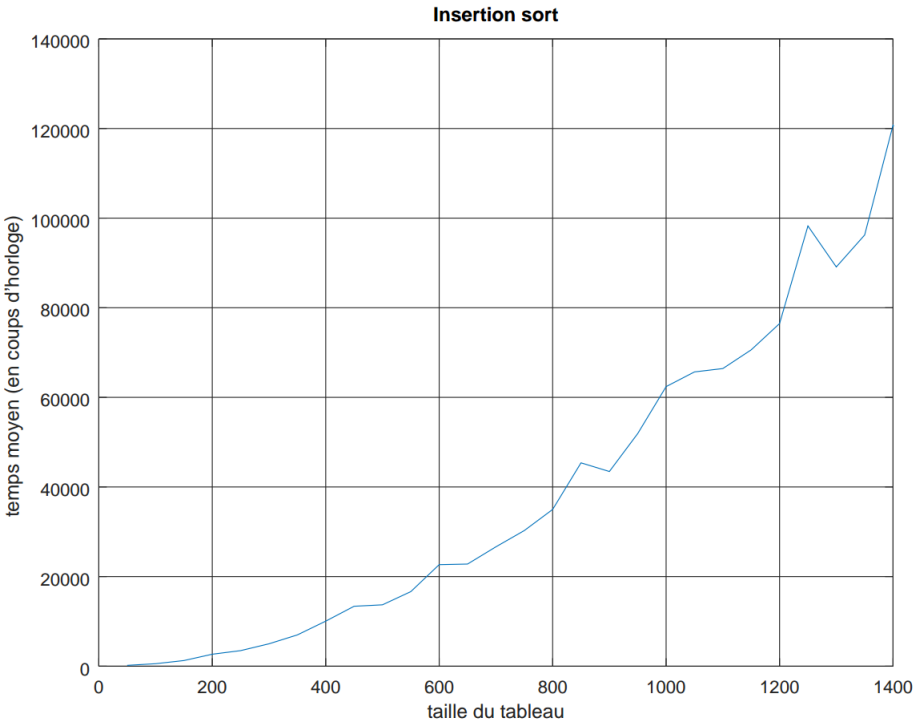

FIGURE 33 – moyenne du temps d'exécution

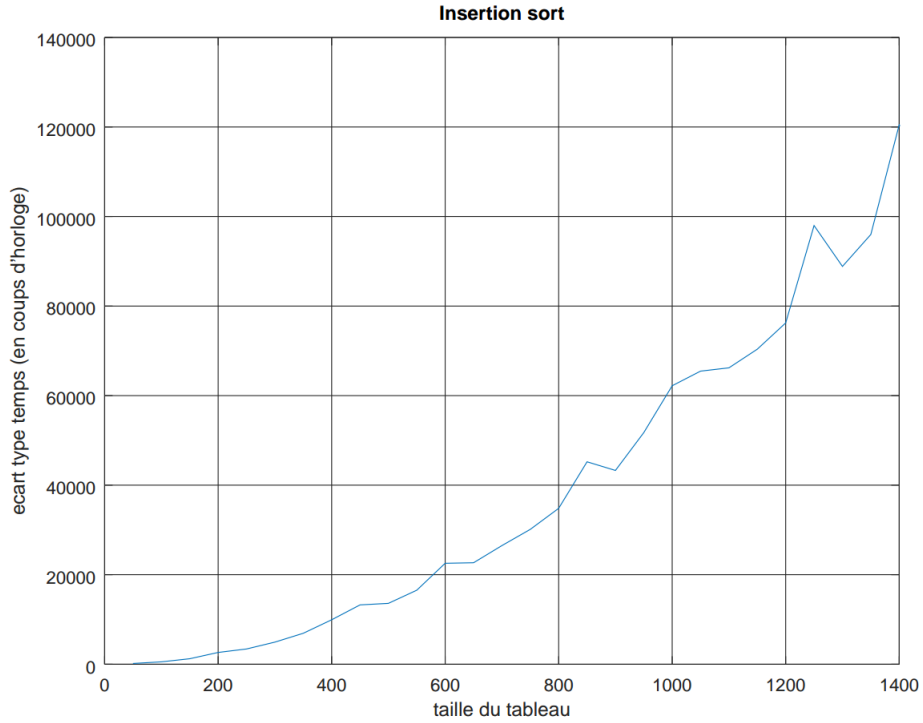

FIGURE 34 – écart type du temps d'exécution

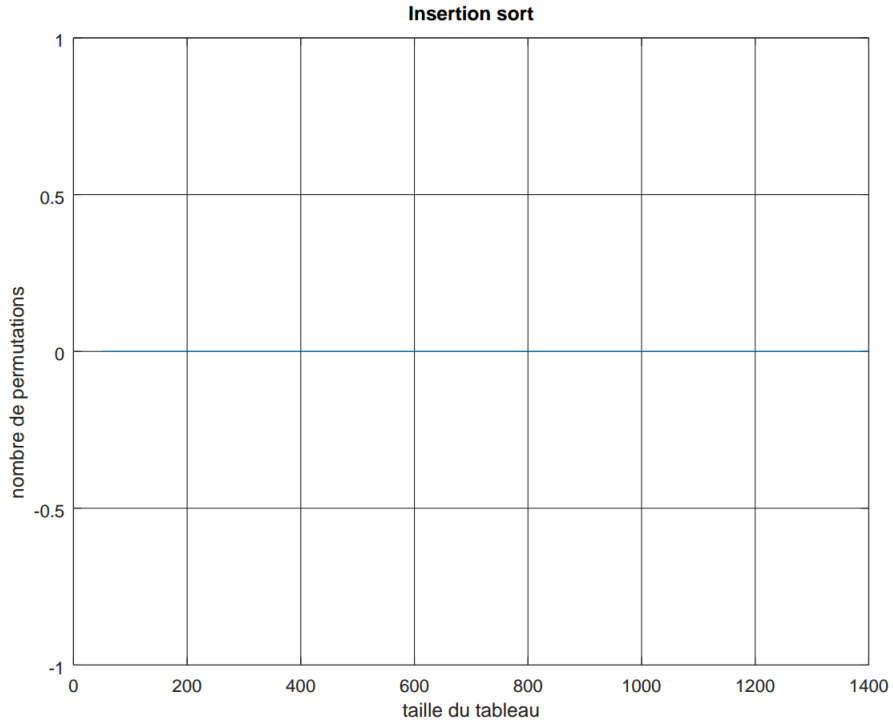

FIGURE 35 – nombre de permutations

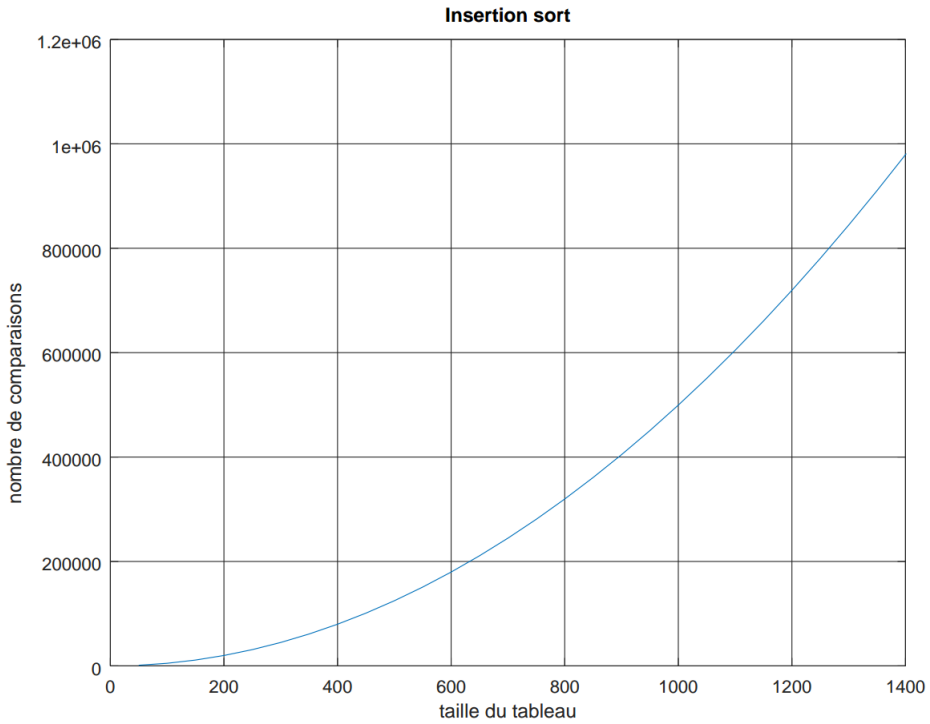

FIGURE 36 – nombre de comparaisons

Commentaire - Tri par insertion

Ces courbes sont conformes aux résultats théoriques. En effet, on confirme bien que dans le cas favorable, les allures du coût en terme de comparaisons et du temps moyen d'exécution sont linéaires. Dans les autres cas ces coûts ont bien des allures quadratiques. Remarquons également que nos courbes présentent tout le temps un nombre de permutation nul, ceci est dû au fait que dans l'algorithme du tri insertion nous n'utilisons pas

la fonction permutation car une insertion n'est pas exactement une permutation. Ce critère n'est donc pas pertinent pour l'étude des performances de cet algorithme.

2.5 Tri Rapide

2.5.1 Principe

Ce tri adopte la stratégie de "diviser pour régner" qui consiste à réduire le problème du tri d'un tableau de taille n aux tris de deux tableaux de taille n/2. La méthode consiste à placer un élément (dit pivot) à sa place définitive en permutant tous les autres de telle sorte que ceux qui lui sont inférieurs soient à sa gauche et que ceux qui lui sont supérieurs soient à sa droite. Cette opération s'appelle le **partitionnement**. Ensuite, on définit un nouveau pivot pour les tableaux obtenus et on repète le partitionnement jusqu'à ce que tous les évènements soient triés.

2.5.2 Code

```
/**
 * Name : partition
 * @params : un tableau d'évenements {events[]} un indice de début {a} et un indice
    de fin {b}
 * @return : renvoie l'indice pivot du sous tableau events[a:b]
 * Description : on choisit comme élément pivot, l'élément de fin puis ajuste sa
    position jusqu'a ce que:
 *          - tous les éléments d'indice précédents soient plus petit (au sens large)
 *          - tous les éléments d'indice suivants soient plus grands (au sens large)
 */
int partition(event events[], int a, int b){
  int pivot = a-1;
  int i;
  event e = events[b];
  for(i = a; i < b; i++){
    if(compare_events(e,events[i]) == 1){
      pivot++;
      swap_events(events, pivot, i);
    }
  }
  pivot++;
  swap_events(events,pivot,b);
  return pivot;
}
```

Listing 4 – procédure partition

```
/**
 * Name : quick_sort
 * @params : un tableau d'évenements {events[]} un indice de début {a} et un indice
    de fin {b}
 * Description : tri le sous tableau events[a:b] par le principe du tri rapide.
 */
void quick_sort(event events[], int a, int b){
  if(b > a){
    int pivot = partition(events,a,b);
    quick_sort(events,a, pivot - 1);
    quick_sort(events,pivot + 1, b);
  }
}
```

Listing 5 – fonction quick_sort

2.5.3 Analyse en complexité

- **Cas favorable**
  Le cas favorable se présente quand le partitionnement fournit deux sous-tableaux de tailles $\lfloor n/2 \rfloor$ et $\lceil n/2 \rceil - 1$ à chaque appel récursif. Ce partitionnement équilibré nous donne la récurrence suivante :

$$T(n) \leq 2T(n/2) + \Theta(n)$$

On montre que cette inégalité entraine une complexité temporelle

$$T(n) = \Theta(nlog(n))$$

- **Cas défavorable**
  Le partitionnement fournit deux sous-tableaux de tailles 0 et n-1 à chaque appel récursif. Ce partitionnement est complètement déséquilibré, il donne la récurrence suivante :

  $$T(n) = T(n-1) + n - 1$$

  donc

  $$T(n) = \sum_{i=1}^{n} i - 1 = \frac{n(n-1)}{2}$$

  donc

  $$T(n) = \Theta(n^2)$$

2.5.4 Etude des performances

- **Tableaux récapitulatifs**

Tableaux triés dans l'ordre croissant

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	178	167.469400	1274	1225
100	655	635.484854	5049	4950
150	1463	1433.489798	11324	11175
200	2601	2563.492735	20099	19900
250	4038	3990.494080	31374	31125
300	5815	5760.495291	45149	44850
350	7869	7803.495819	61424	61075
400	10958	10879.496404	80199	79800
450	13194	13104.496595	101474	101025
500	16100	16005.497056	125249	124750
550	19554	19397.495972	151524	150975
600	23180	23067.497567	180299	179700
650	27263	27138.497711	211574	210925
700	32778	32619.497574	245349	244650
750	37325	37183.498101	281624	280875
800	42166	41982.497818	320399	319600
850	47671	47511.498324	361674	360825
900	53308	53128.498313	405449	404550
950	59245	58971.497683	451724	450775
1000	65503	65314.498559	500499	499500
1050	72212	72008.498589	551774	550725
1100	78633	78429.498704	605549	604450
1150	85701	85484.498735	661824	660675
1200	93200	92973.498783	720599	719400
1250	101525	101293.498859	781874	780625
1300	109373	109135.498913	845649	844350
1350	118513	118270.498976	911924	910575
1400	124641	124388.498986	980699	979300

Tableaux désordonnés

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	40	29.340416	14900	21700
100	128	104.503636	43700	70500
150	182	148.367112	55900	101100
200	329	285.587622	128200	200100
250	364	307.575812	128800	226500
300	416	352.171648	137684	237448
350	451	377.186625	154100	274500
400	566	474.828537	191500	343600
450	692	593.117990	267100	416600
500	780	666.669513	261100	468900
550	946	824.863134	324800	567300
600	924	791.605217	315000	575500
650	1016	864.555221	338600	602200
700	1075	935.219226	342100	687700
750	1118	970.637739	363000	751800
800	1305	1144.475884	502200	882900
850	1255	1094.631915	509138	866238
900	1546	1369.978460	520700	1002100
950	1607	1410.373841	531900	957200
1000	1499	1309.835696	620200	1035300
1050	1915	1694.205628	660000	1309000
1100	1968	1753.030807	719400	1332700
1150	1868	1650.289926	742949	1264521
1200	1873	1633.833875	689000	1293000
1250	2061	1818.234999	735400	1407200
1300	2443	2190.893701	966400	1556500
1350	2757	2473.072682	868000	1558500
1400	2250	1981.933107	869360	1561682

Tableaux triés dans l'ordre décroissant

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	181	166.457202	649	1225
100	619	596.481349	2549	4950
150	1382	1353.489564	5699	11175
200	2456	2414.491458	10099	19900
250	3714	3664.493280	15749	31125
300	5368	5313.494895	22649	44850
350	7152	7083.495182	30799	61075
400	9388	9311.495906	40199	79800
450	11888	11805.496516	50849	101025
500	14605	14514.496891	62749	124750
550	17590	17489.497134	75899	150975
600	21094	20978.497253	90299	179700
650	24308	24179.497348	105949	210925
700	28425	28296.497734	122849	244650
750	32564	32427.497899	140999	280875
800	37128	36970.497873	160399	319600
850	41833	41677.498137	181049	360825
900	46801	46628.498153	202949	404550
950	52063	51892.498360	226099	450775
1000	58635	58452.498441	250499	499500
1050	64066	63855.498354	276149	550725
1100	70353	70150.498558	303049	604450
1150	76933	76723.498636	331199	660675
1200	83775	83544.498622	360599	719400
1250	90573	90346.498748	391249	780625
1300	99232	98989.498776	423149	844350
1350	106385	106131.498807	456299	910575
1400	113828	113472.498435	490699	979300

- **Courbes représentatives**

Tableaux triés dans l'ordre croissant

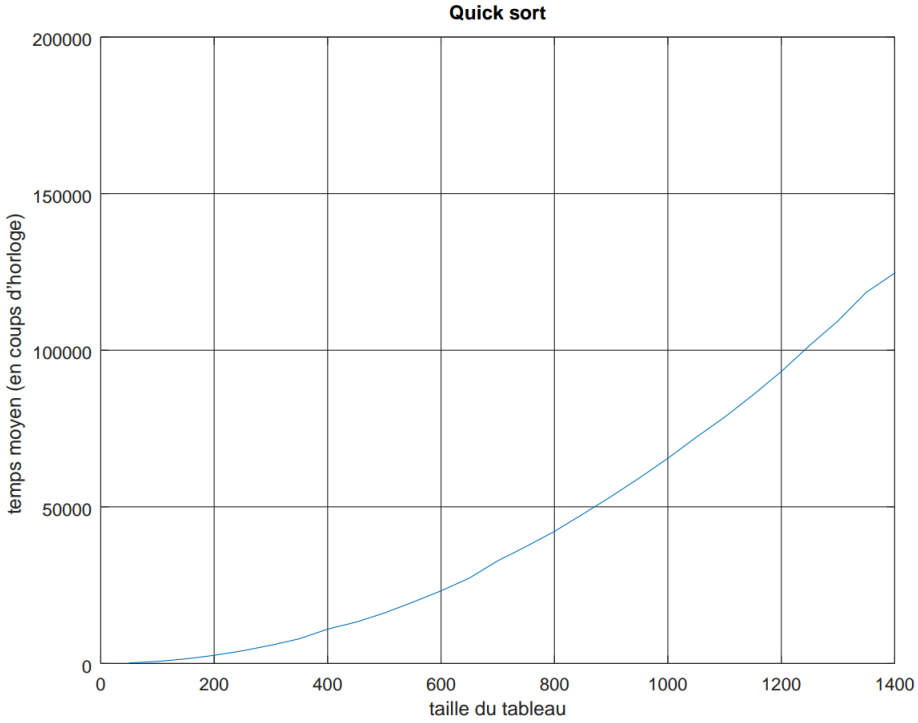

FIGURE 37 – moyenne du temps d'exécution

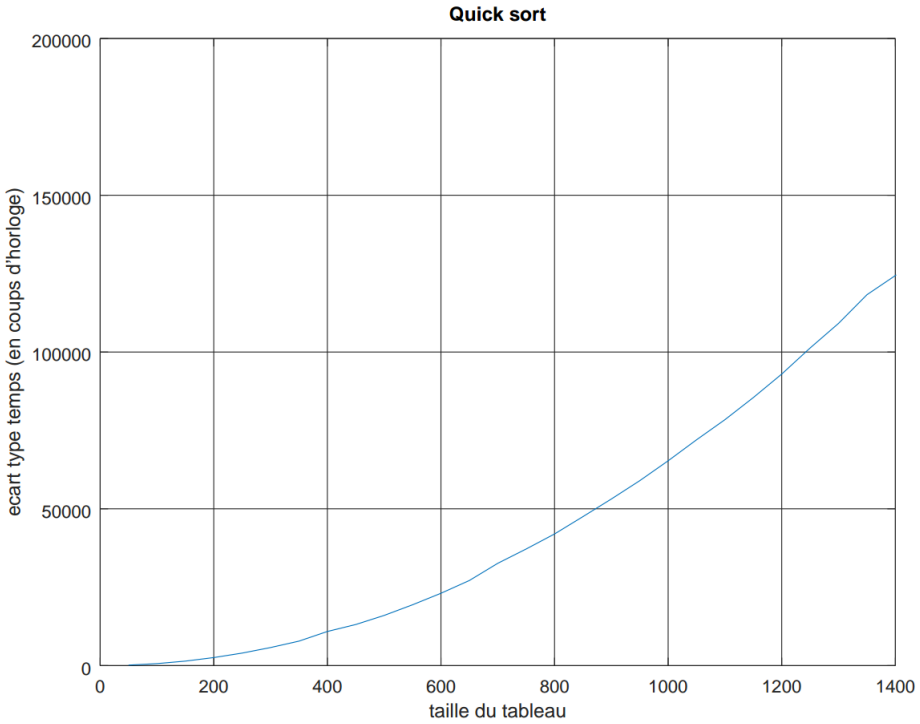

FIGURE 38 – écart type du temps d'exécution

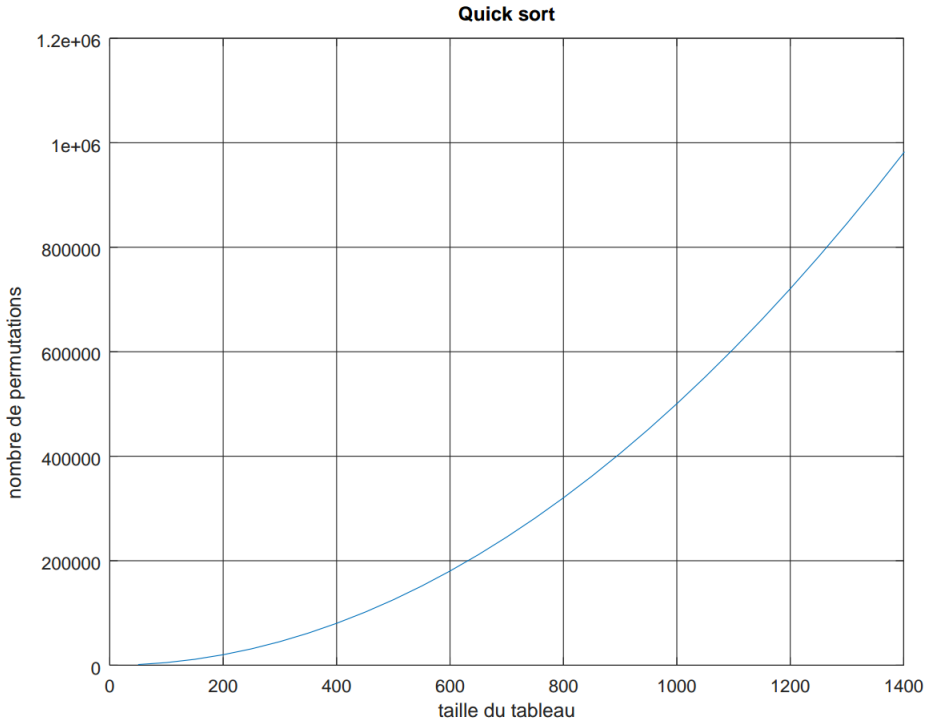

FIGURE 39 – nombre de permutations

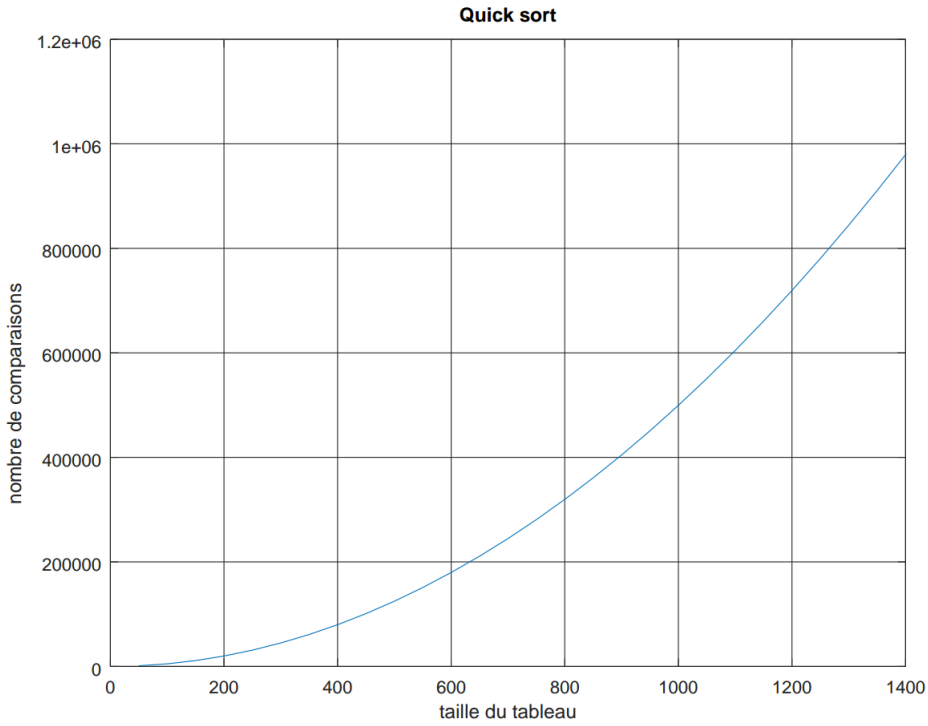

FIGURE 40 – nombre de comparaisons

Tableaux désordonnés

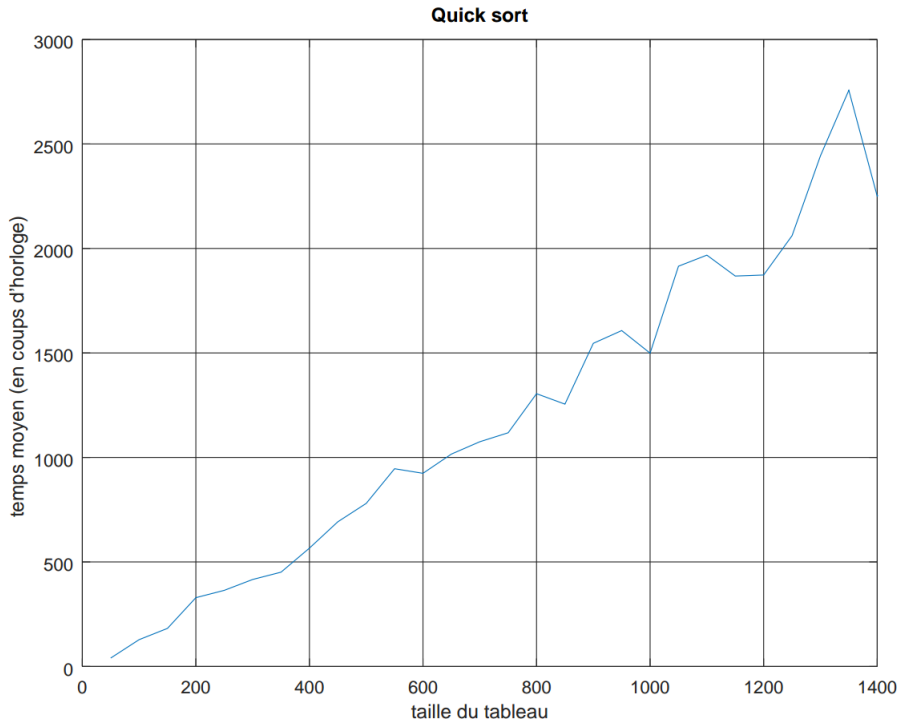

FIGURE 41 – moyenne du temps d'exécution

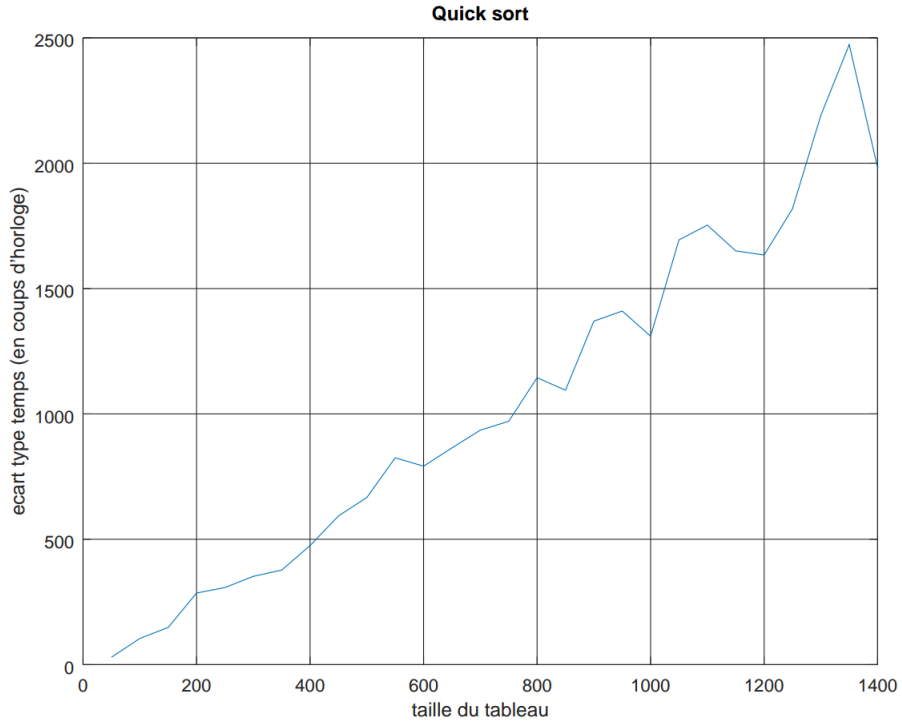

FIGURE 42 – écart type du temps d'exécution

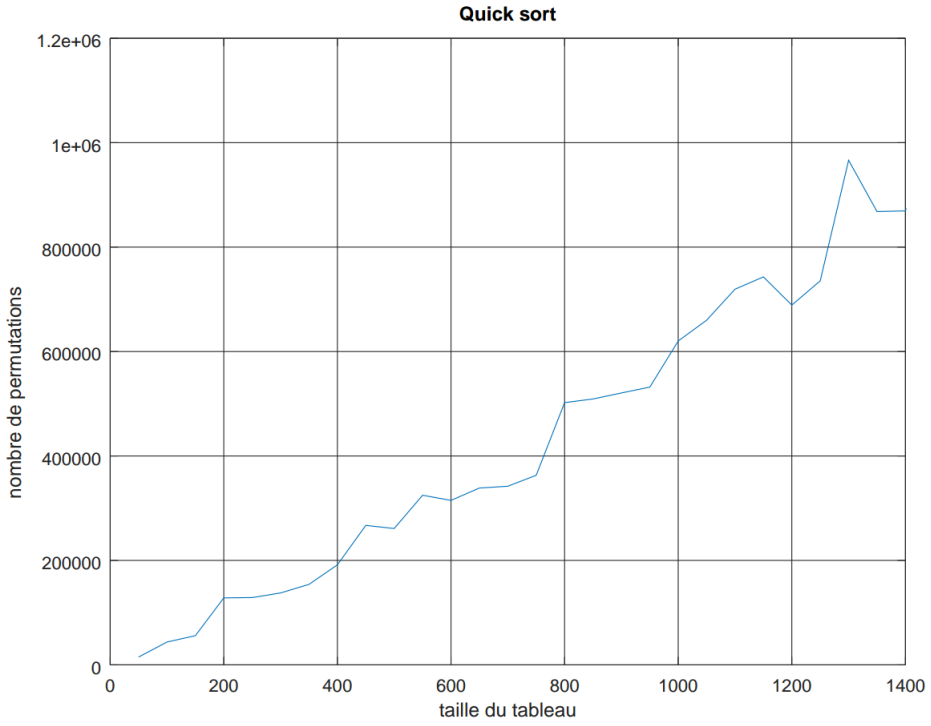

FIGURE 43 – nombre de permutations

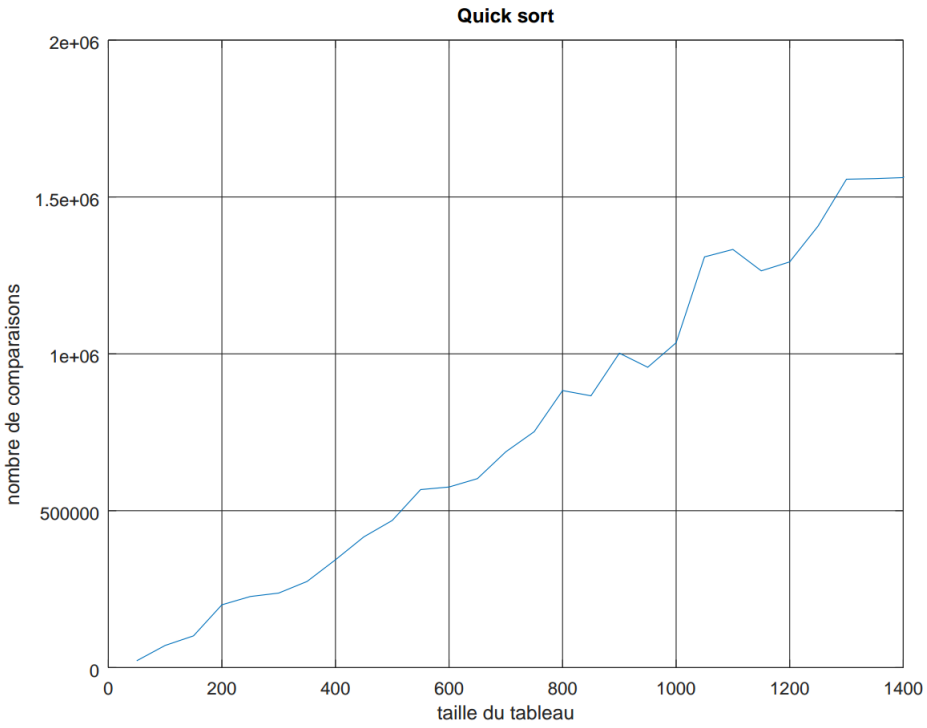

FIGURE 44 – nombre de comparaisons

Tableaux triés dans l'ordre décroissant

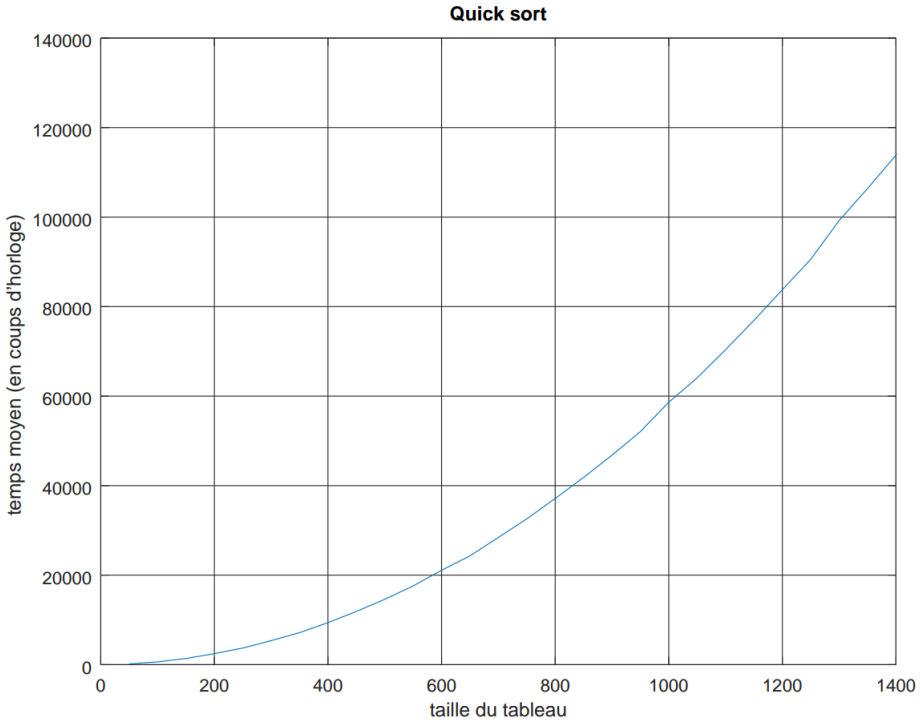

FIGURE 45 – moyenne du temps d'exécution

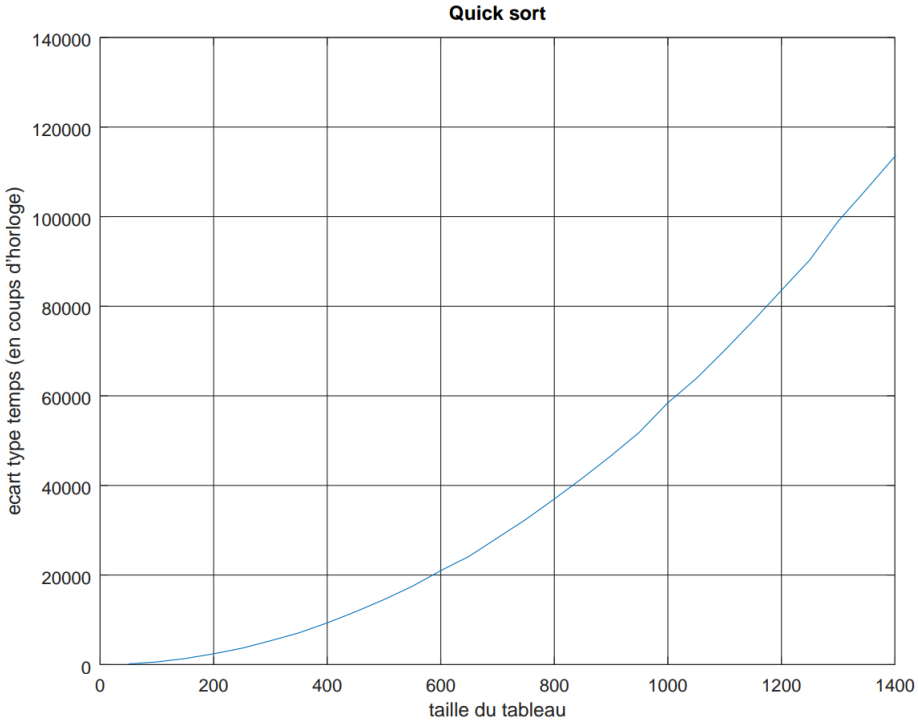

FIGURE 46 – écart type du temps d'exécution

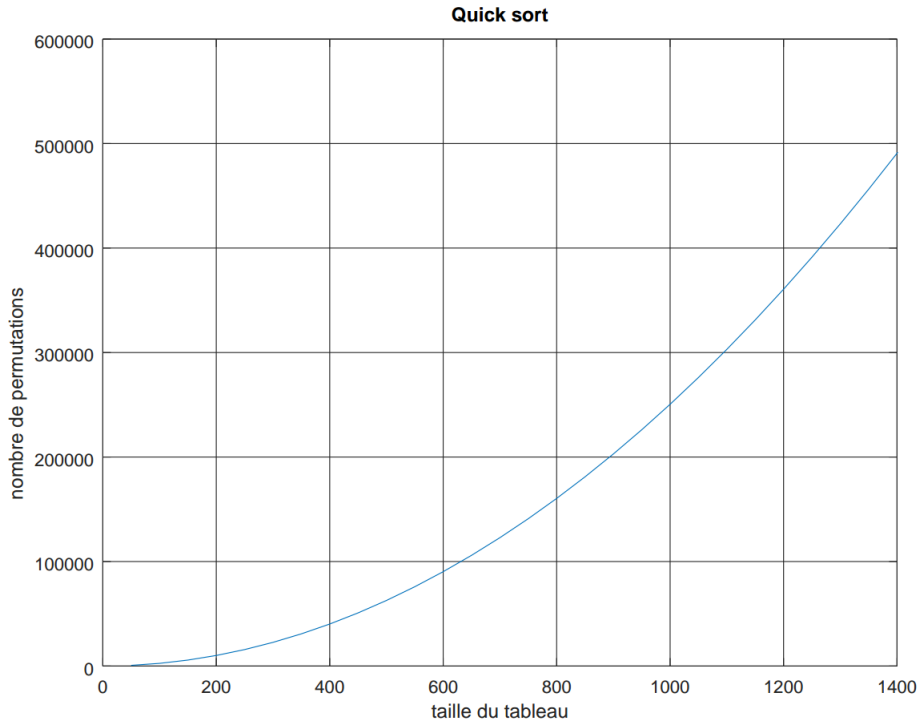

FIGURE 47 – nombre de permutations

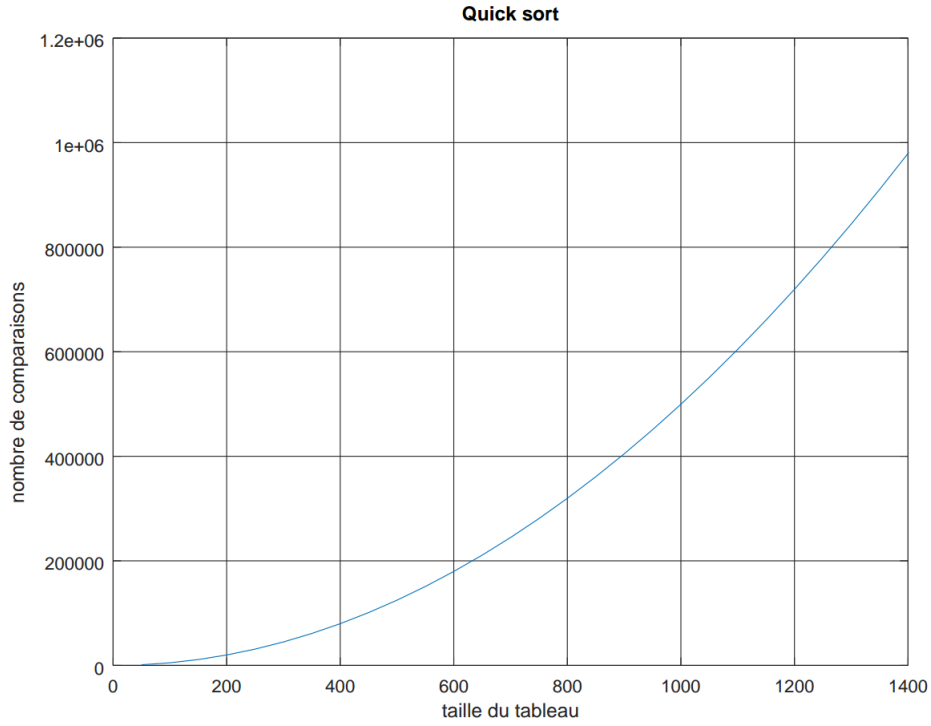

FIGURE 48 – nombre de comparaisons

Remarques - Tri Rapide

Ces courbes sont conformes aux résultats théoriques. En effet, on confirme les allures quasi linéaires du tri rapide dans le cas aléatoire, on observe aussi le comportement de dégénerescence en complexité quadratique lorsque l'on se trouve dans le cas de tableaux triés. Toutefois il est indéniable que ce tri est généralement, nettement plus efficace que les autres en terme de performance, soulignons par exemple le fait que dans le cas

aléatoire, sa moyenne de temps d'exécution est aux alentours de **3 000** coups d'horloge contre **7 000** pour le tri insertion et **300 000** pour le tri à bulle !

2.6 Tri Fusion

2.6.1 Principe

Ce tri adopte également la stratégie de "diviser pour régner". Il exploite le fait qu'à partir de deux listes triées, on peut facilement construire une liste triée comportant les éléments issus de ces deux listes (leur "fusion"). Le principe de l'algorithme de tri fusion repose sur cette observation : le plus petit élément de la liste à construire est soit le plus petit élément de la première liste, soit le plus petit élément de la deuxième liste. Ainsi, on peut construire la liste élément par élément en retirant tantôt le premier élément de la première liste, tantôt le premier élément de la deuxième liste (en fait, le plus petit des deux, à supposer qu'aucune des deux listes ne soit vide, sinon la réponse est immédiate). Ce procédé est appelé fusion et est au cœur de l'algorithme de tri développé ci-après.

2.6.2 Code

```
/**
 * Name : merge
 * @params : un tableau d'évenements {events[]} un indice de début {a}, un indice
    intermédiare {c} et un indice de fin {b}
 * Description : fusionne les sous tableaux events[a:c] et events[c+1:b], sous ré
    serve qu'ils soient déja triés.
 */
void merge(event events[], int a, int c, int b){
   int t1 = c-a+1;
   int t2 = b-c;
   int k = a, i = 0, j = 0;
  event sub1[t1];
  event sub2[t2];
  for(i = a; i <= c; i++ )
    sub1[i-a] = events[i];
  for(i = c+1; i<=b; i++)
    sub2[i-c-1] = events[i];
  i=0;
  for(k = a; k<= b; k++){
    if( i >  t1 - 1 ){
      events[k] = sub2[j];
      j++;
    }else if(j > t2 - 1){
      events[k] = sub1[i];
      i++;
    }else if( compare_events(sub1[i],sub2[j]) == 1){
      events[k] = sub2[j];
      j++;
    }else{
      events[k] = sub1[i];
      i++;
    }
  }
}
```

Listing 6 – procédure merge (fusion)

```
/**
 * Name : merge_sort
 * @params : un tableau d'évenements {events[]} un indice de début {a} et un indice
    de fin {b}
 * Description : tri le sous tableau events[a:b] par le principe du tri fusion.
 */
void merge_sort(event events[], int a , int b){
  if(b > a){
    int mil = (a+b)/2;
    merge_sort(events, a , mil);
    merge_sort(events, mil + 1, b);
```

```
11      merge(events, a, mil, b);
12    }
13  }
```

Listing 7 – fonction merge_sort

2.6.3 Analyse en complexité

Le tri par fusion est insensible aux données qu'il trie. En effet, si on regarde dep lus près le programme, on se rend compte que la boucle principale effectue systématiquement le même nombre d'opérations, quel que soit l'orrdre relatif des clés des sous-tableaux de gauche et de droite. On en déduit qu'il n'y a pas de cas favorable ou défavorable. Toutes les entrées de taille n seront traitées avec un temps identiques. La procédure **merge** appliquée à un tableau de taille n effectue $2n+2$ affectations et n comparaisons. Elle est donc en $\Theta(n)$. On voit également que la partition du problème dans **merge_sort** est en $\Theta(1)$.
On se ramène donc à la récurrence suivante :

$$T(n) = \begin{cases} \Theta(1),\ si\ n = 1 \\ 2\Theta(n/2) + \Theta(n),\ sinon \end{cases}$$

On a déjà vu dans l'étude du cas favorable du tri rapide que cette récurrence aboutit au résultat suivant :

$$T(n) = \Theta(nlog(n))$$

2.6.4 Etude des performances

- **Tableaux récapitulatifs**

Tableaux triés dans l'ordre croissant

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	124	60.991803	0	153
100	174	123.296391	0	356
150	253	192.343443	0	579
200	334	249.331105	0	812
250	405	322.372455	0	1011
300	563	422.333991	0	1308
350	672	516.349688	0	1577
400	667	487.316119	0	1824
450	577	463.377816	0	2061
500	674	555.393554	0	2272
550	974	789.383304	0	2591
600	1182	914.353870	0	2916
650	1254	998.372175	0	3211
700	1379	1093.369562	0	3504
750	1448	1130.359677	0	3769
800	1296	1019.364508	0	4048
850	1384	1079.359069	0	4333
900	1402	1105.366003	0	4572
950	1506	1198.371812	0	4829
1000	1578	1227.357324	0	5044
1050	1630	1311.378664	0	5349
1100	1375	1010.319751	0	5732
1150	1308	1092.401483	0	6071
1200	1301	1076.395838	0	6432
1250	1372	1125.390599	0	6751
1300	1418	1174.396441	0	7072
1350	1470	1222.398871	0	7407
1400	1885	1604.412665	0	7708

Tableaux désordonnés

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	45	32.698318	0	22400
100	107	84.669593	0	54200
150	171	141.121898	0	91100
200	231	190.342428	0	129100
250	340	282.452863	0	168400
300	393	327.311503	0	211300
350	468	398.621889	0	252500
400	612	518.439331	0	297900
450	682	570.964762	0	341740
500	691	590.511930	0	384800
550	749	645.180533	0	433000
600	847	726.988693	0	480300
650	907	784.991204	0	526500
700	1143	985.275058	0	574900
750	1126	971.968225	0	624400
800	1242	1071.781270	0	674600
850	1430	1230.876440	0	723700
900	1563	1352.093976	0	772750
950	1559	1350.634336	0	819900
1000	1567	1347.128045	0	867900
1050	1552	1350.127694	0	918300
1100	1635	1428.503087	0	972700
1150	1726	1506.703780	0	1024900
1200	1784	1557.516266	0	1077600
1250	1898	1652.166547	0	1129616
1300	1983	1740.096259	0	1184700
1350	2061	1808.063348	0	1235100
1400	2189	1911.961859	0	1288500

Tableaux triés dans l'ordre décroissant

Taille de données	temps moyen	écart type temps	Nbre permutations	Nbre comparaisons
50	57	32.124757	0	133
100	112	78.287930	0	316
150	178	129.313572	0	515
200	196	140.303243	0	732
250	236	187.371289	0	983
300	334	240.306055	0	1180
350	318	249.363189	0	1411
400	376	300.374766	0	1664
450	516	412.374829	0	1927
500	641	474.324783	0	2216
550	712	562.367318	0	2435
600	611	481.365765	0	2660
650	733	569.356654	0	2915
700	710	560.366844	0	3172
750	721	579.378115	0	3457
800	794	642.382285	0	3728
850	866	684.367591	0	3993
900	907	741.388562	0	4304
950	950	773.386061	0	4597
1000	1019	824.382193	0	4932
1050	1092	898.392453	0	5203
1100	1132	926.389227	0	5420
1150	1317	1090.396258	0	5681
1200	1451	1110.346793	0	5920
1250	1319	1071.384618	0	6201
1300	1359	1120.393681	0	6480
1350	1466	1218.398539	0	6745
1400	1573	1293.392052	0	7044

- **Courbes représentatives**

Tableaux triés dans l'ordre croissant

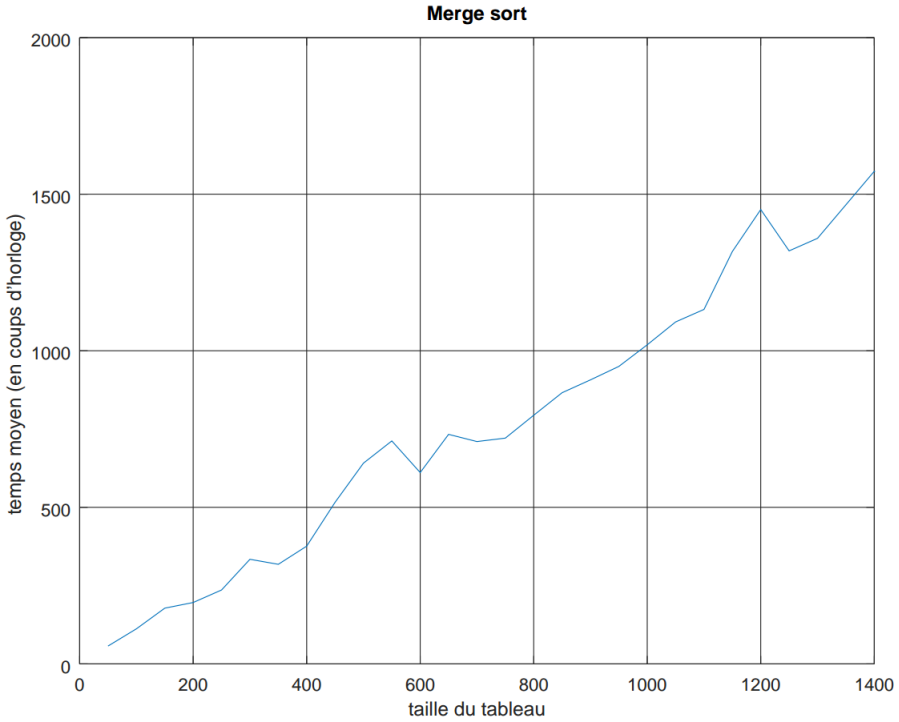

FIGURE 49 – moyenne du temps d'exécution

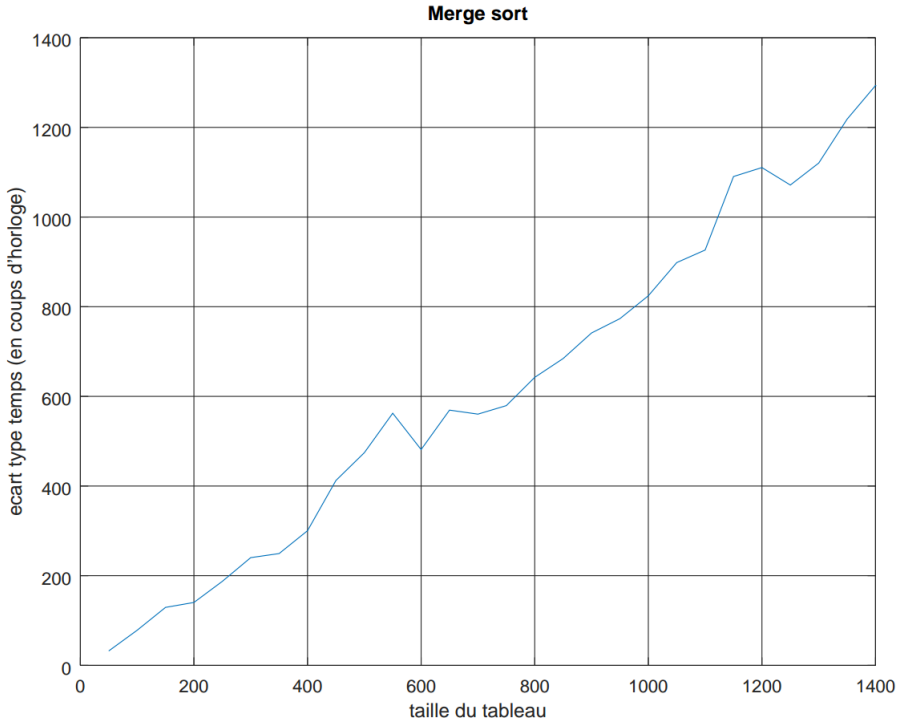

FIGURE 50 – écart type du temps d'exécution

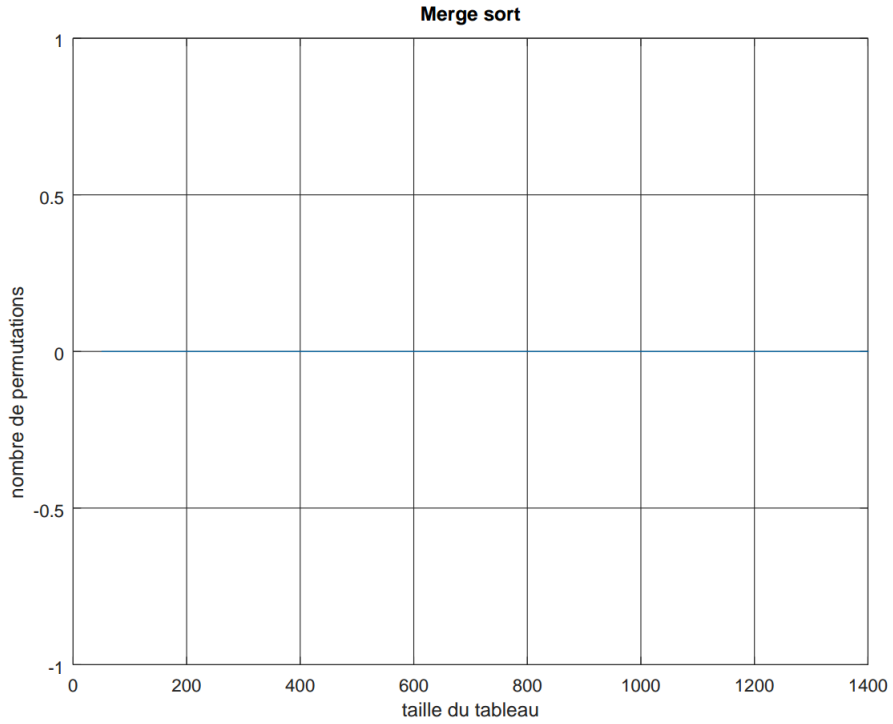

FIGURE 51 – nombre de permutations

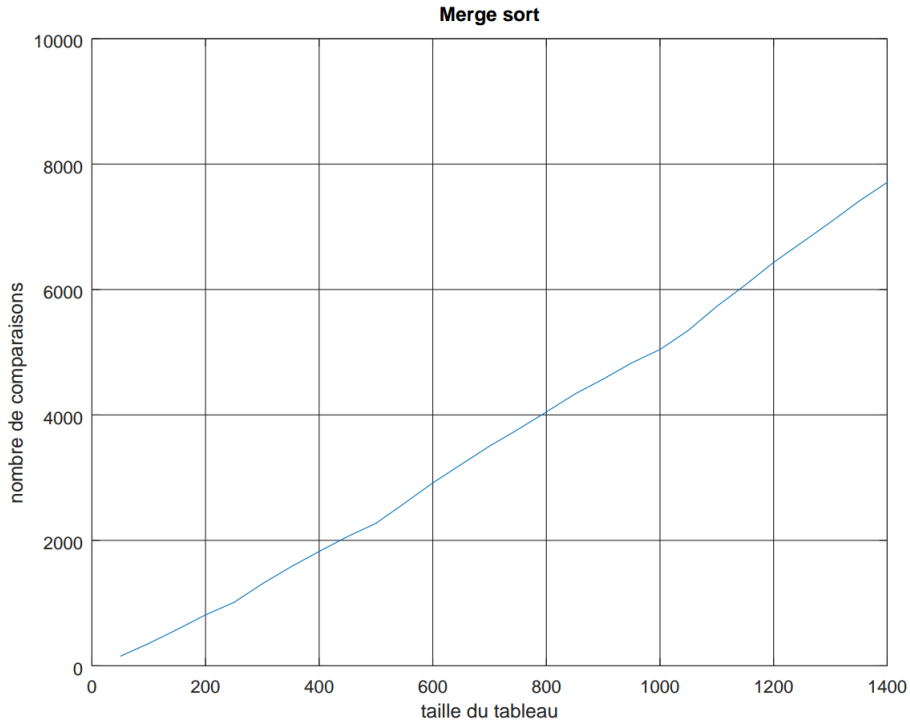

FIGURE 52 – nombre de comparaisons

Tableaux désordonnés

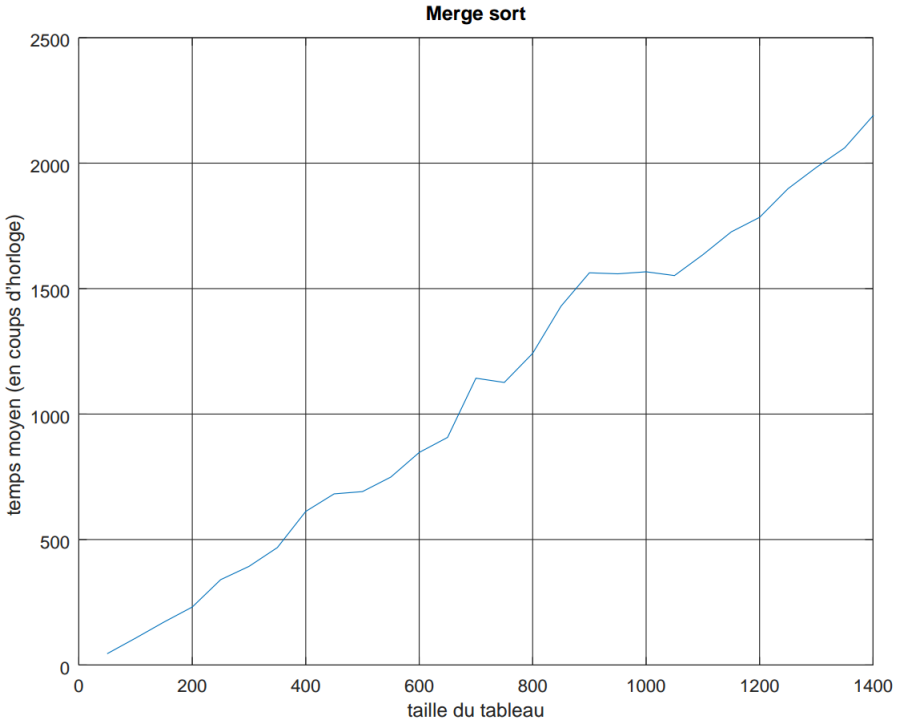

FIGURE 53 – moyenne du temps d'exécution

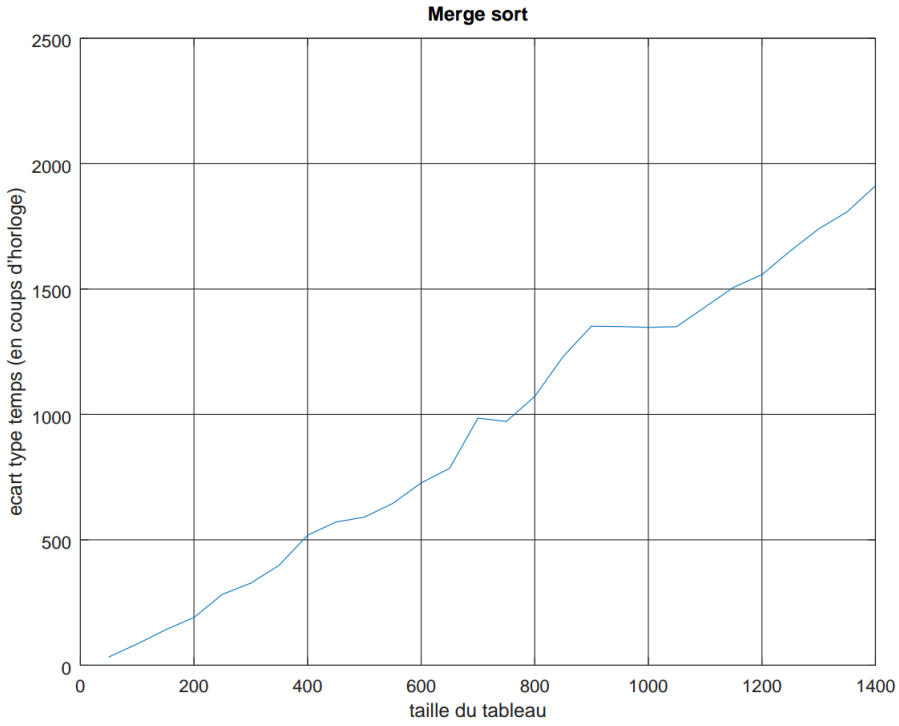

FIGURE 54 – écart type du temps d'exécution

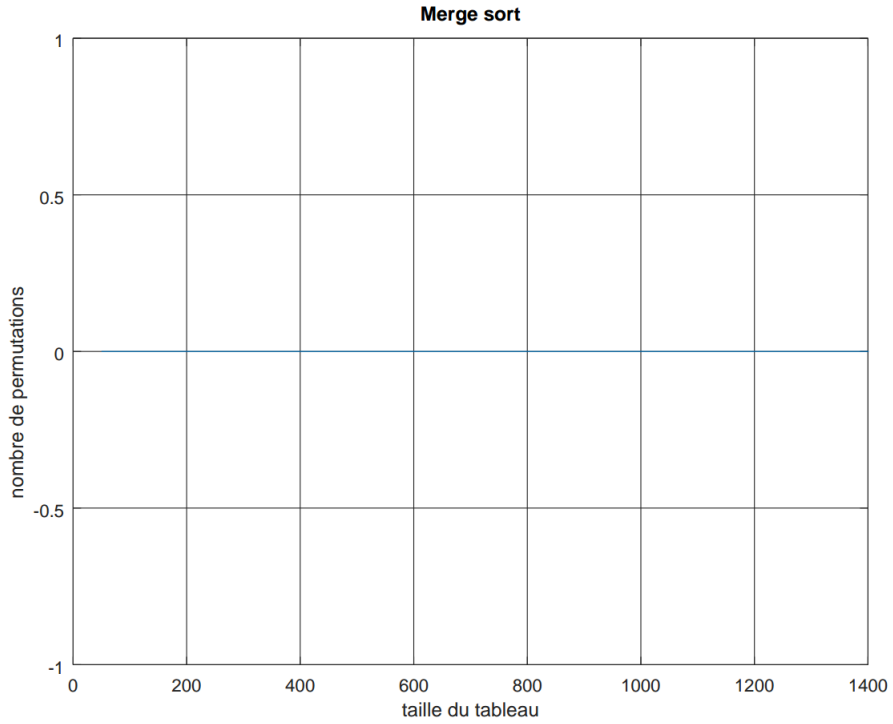

FIGURE 55 – nombre de permutations

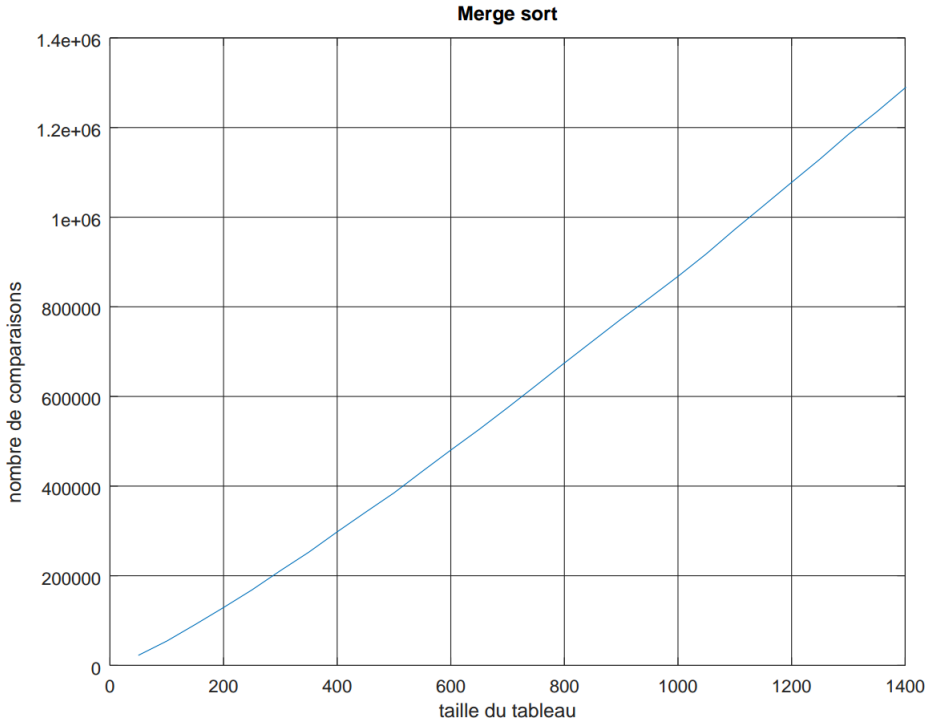

FIGURE 56 – nombre de comparaisons

Tableaux triés dans l'ordre décroissant

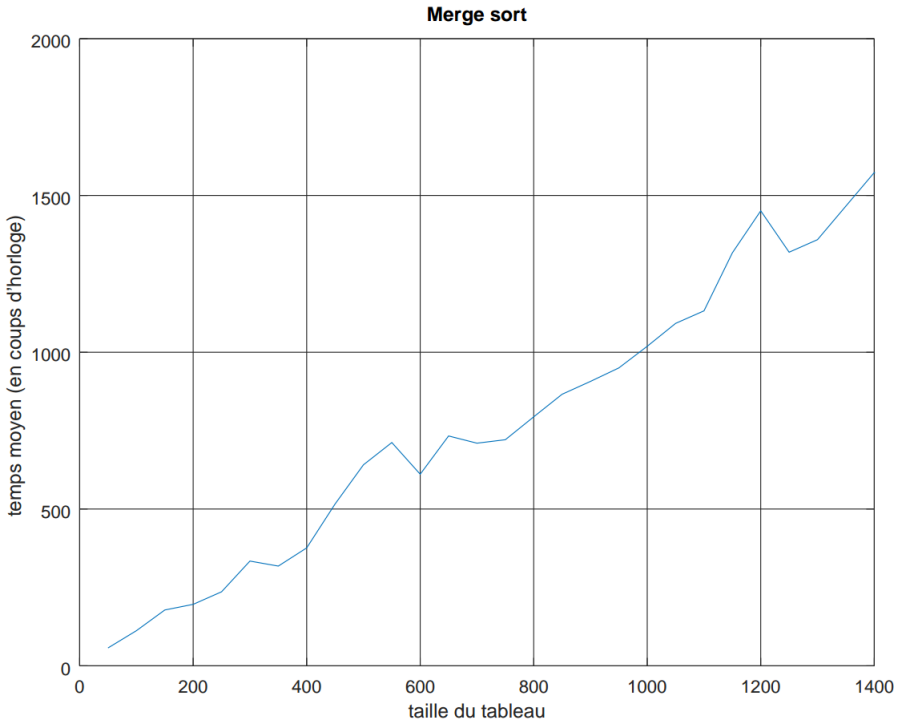

FIGURE 57 – moyenne du temps d'exécution

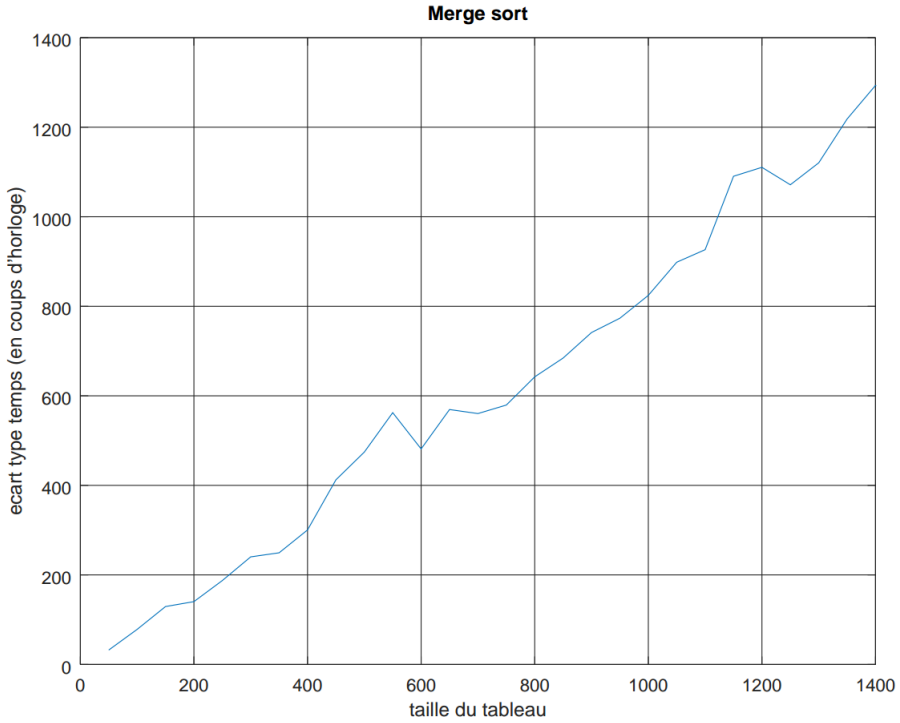

FIGURE 58 – écart type du temps d'exécution

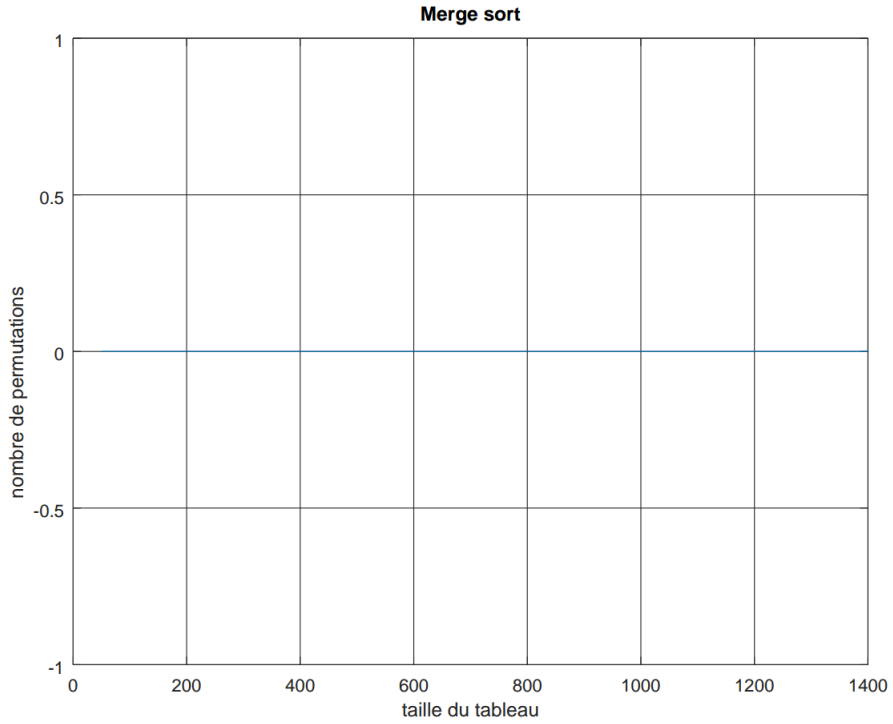

FIGURE 59 – nombre de permutations

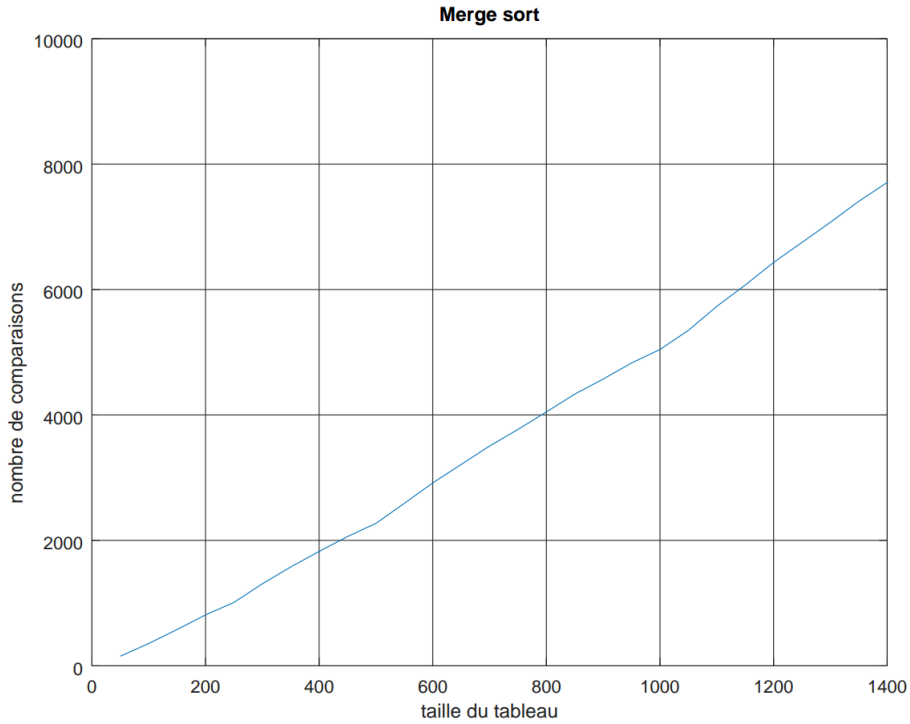

FIGURE 60 – nombre de comparaisons

Commentaire - Tri Fusion

Ces courbes sont conformes aux résultats théoriques. En effet, on constate une allure quasi linéaire dans tous les cas, le tri présente des performances très similaires à celui du tri rapide vu précédement, par exemple dans le cas aléatoire on observe que la moyenne du temps d'exécution est aux alentours de **2 500** coups d'horloge, ce qui est légèrement meilleur que le tri rapide, mais du même ordre de grandeur. En pratique, le tri rapide est

généralement meilleur, mais il peut aussi arriver que sur des instances de problème le tri fusion l'emporte.

3 Comparaison des courbes de performances des algorithmes de tri

Dans cette section nous proposons de tracer, pour chacun des cas tableaux croissants, tableaux aléatoires, tableaux décroissant et pour chacun des critères de mesure, toutes les courbes correspondants à chaque algorithme sur le même graphique afin de mieux visualiser les différences en terme de performance.

3.1 Tableaux triés dans l'ordre croissant

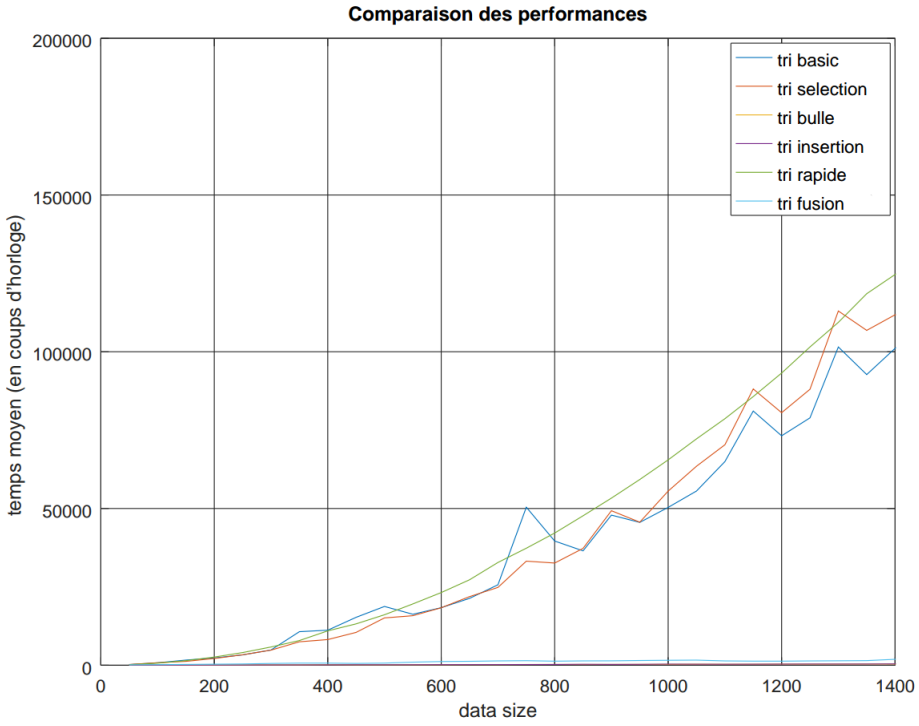

FIGURE 61 – moyenne du temps d'exécution

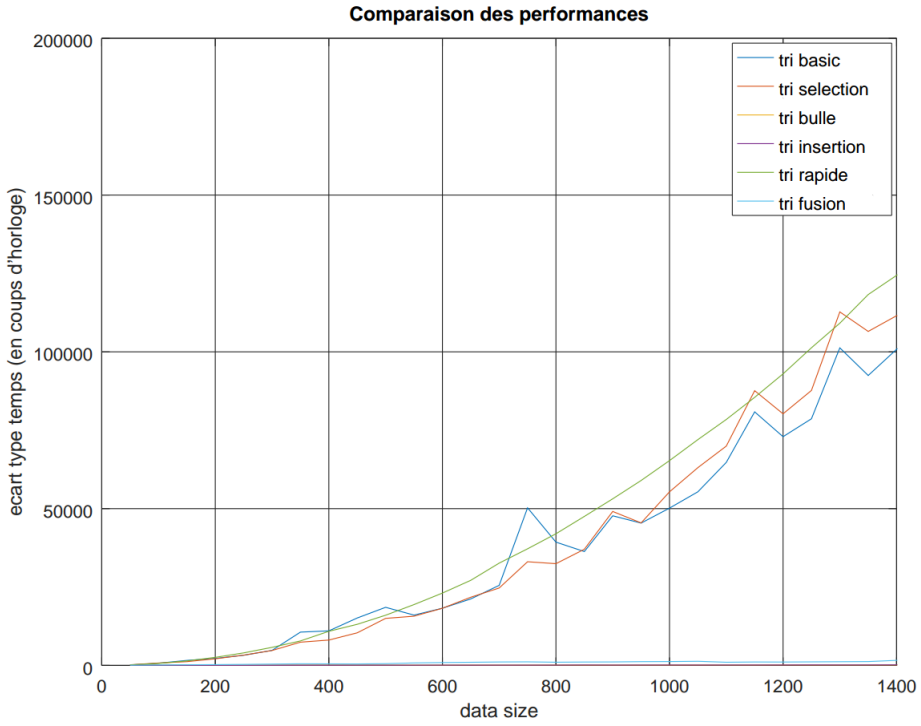

FIGURE 62 – écart type du temps d'exécution

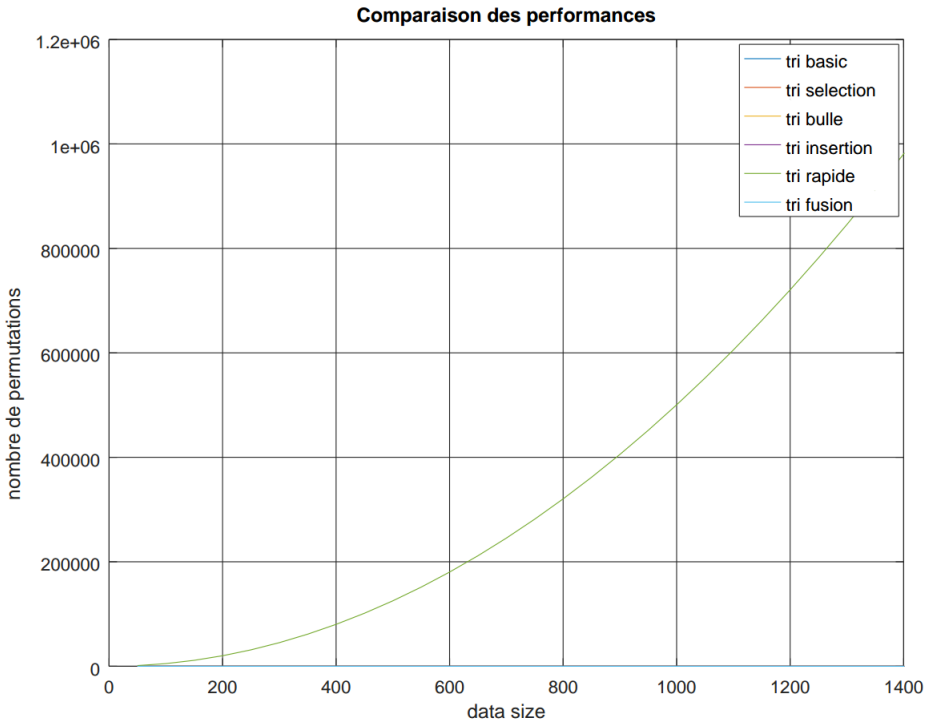

FIGURE 63 – nombre de permutations

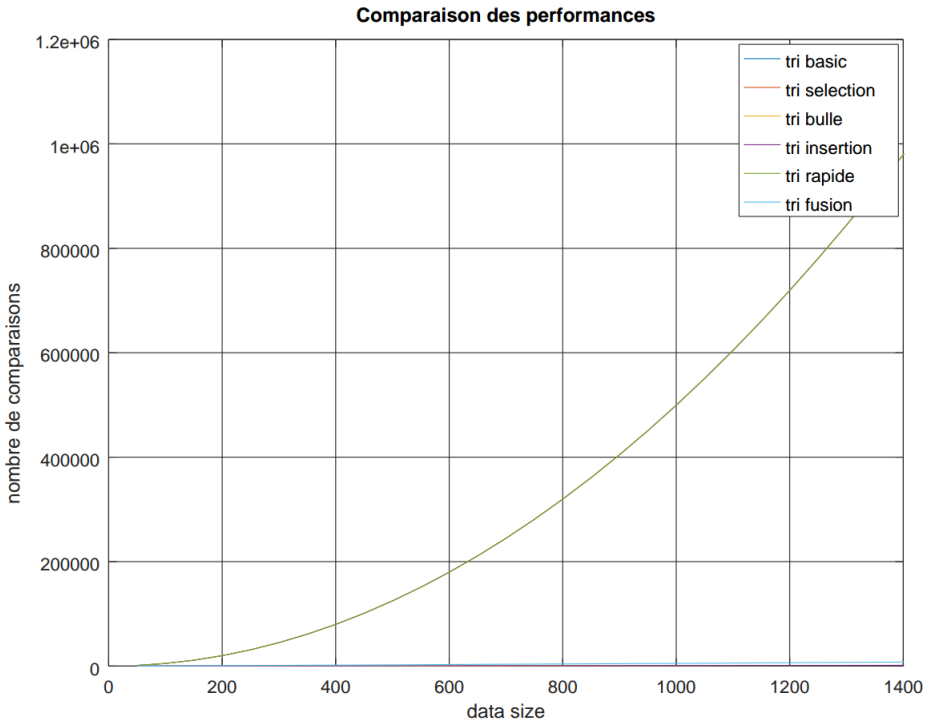

FIGURE 64 – nombre de comparaisons

3.2 Tableaux désordonnés

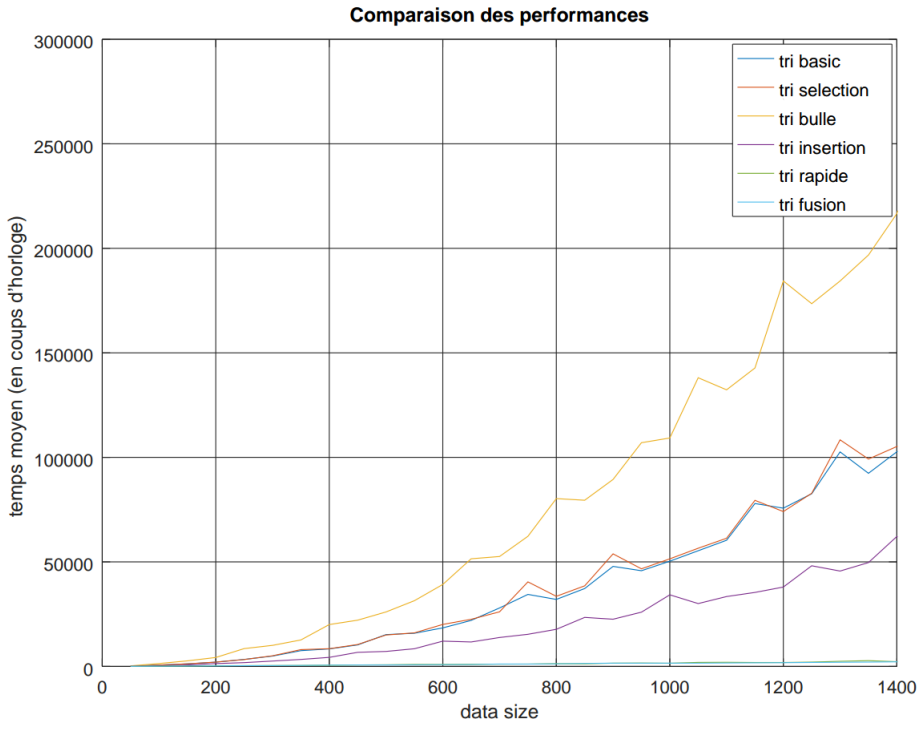

FIGURE 65 – moyenne du temps d'exécution

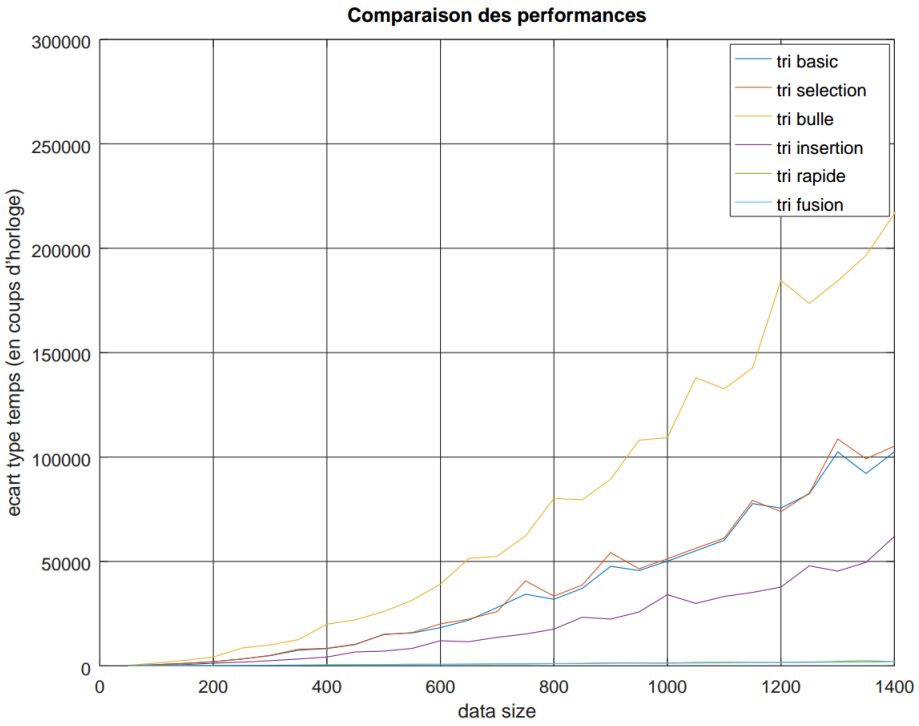

FIGURE 66 – écart type du temps d'exécution

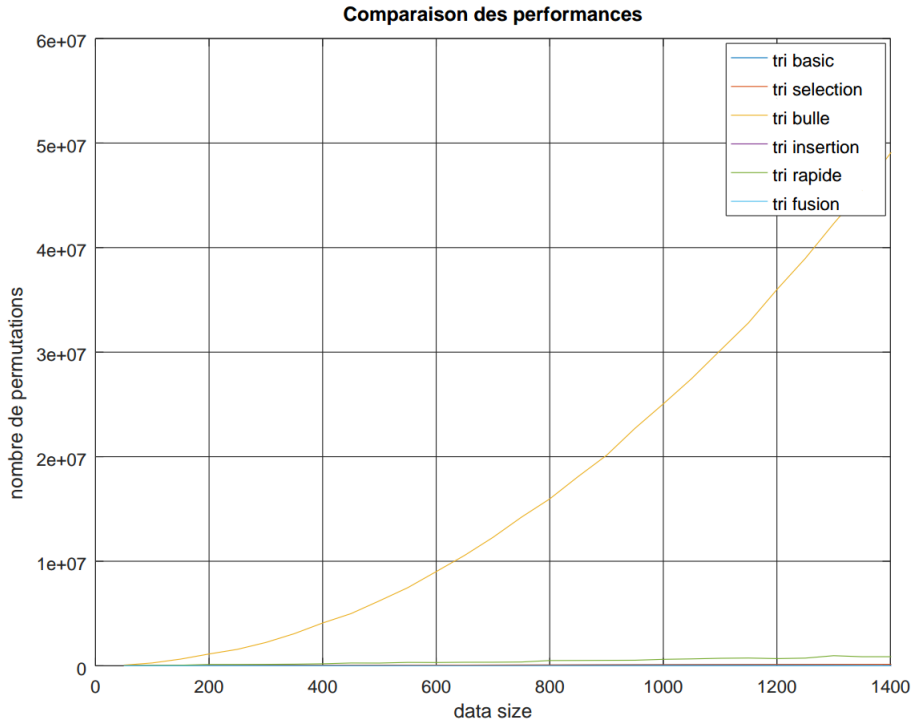

FIGURE 67 – nombre de permutations

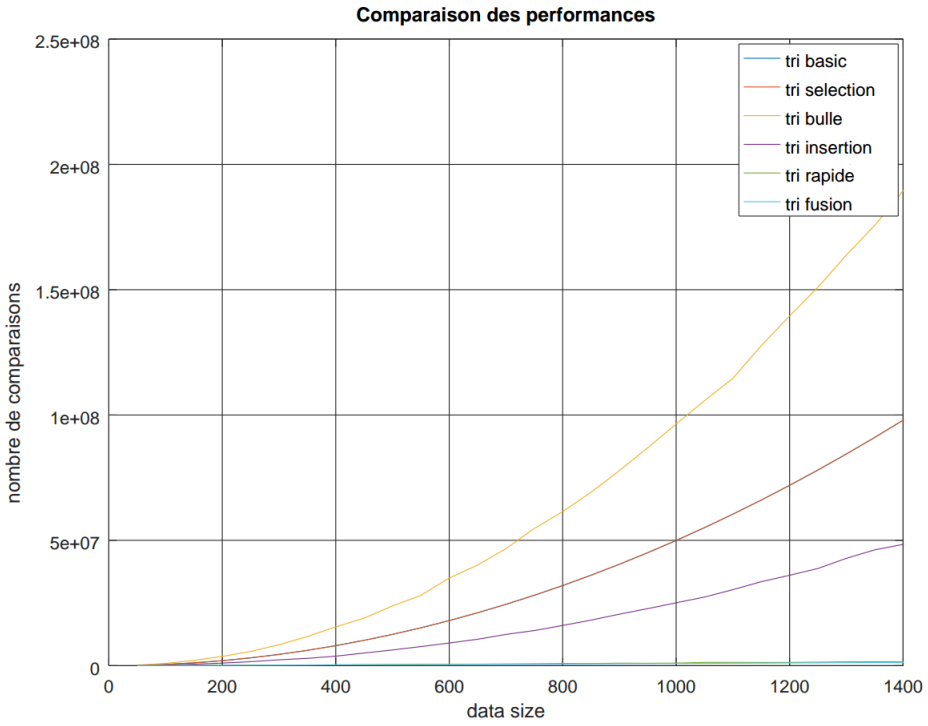

FIGURE 68 – nombre de comparaisons

3.3 Tableaux triés dans l'ordre décroissant

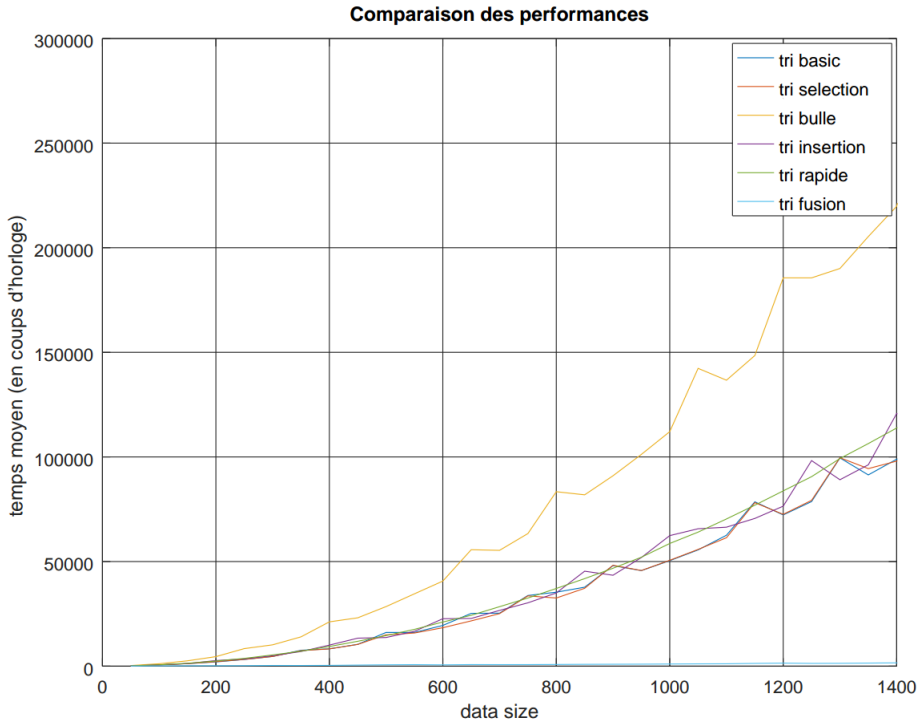

FIGURE 69 – moyenne du temps d'exécution

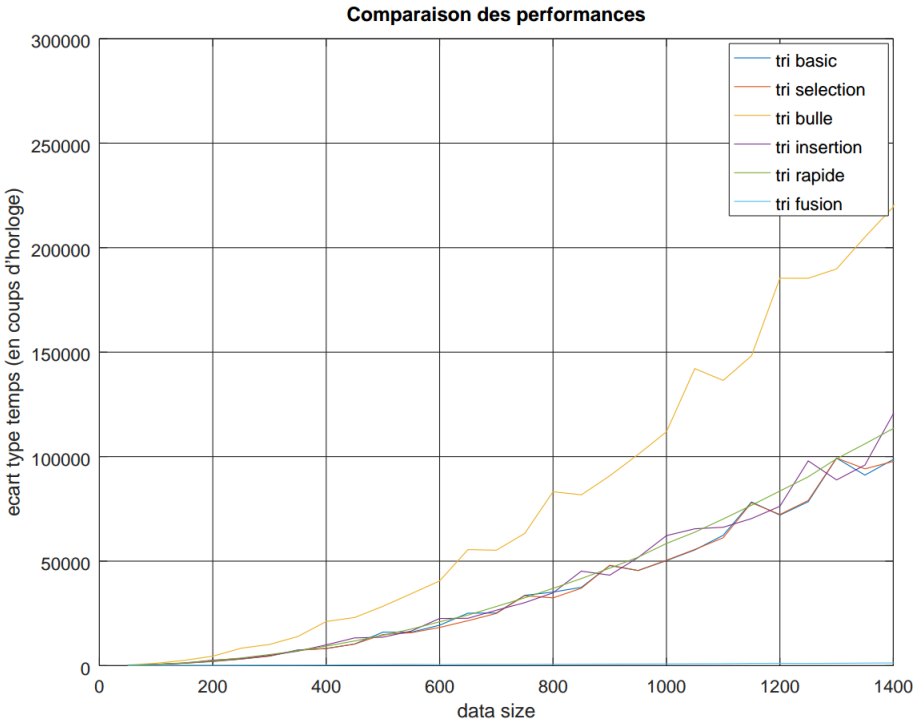

FIGURE 70 – écart type du temps d'exécution

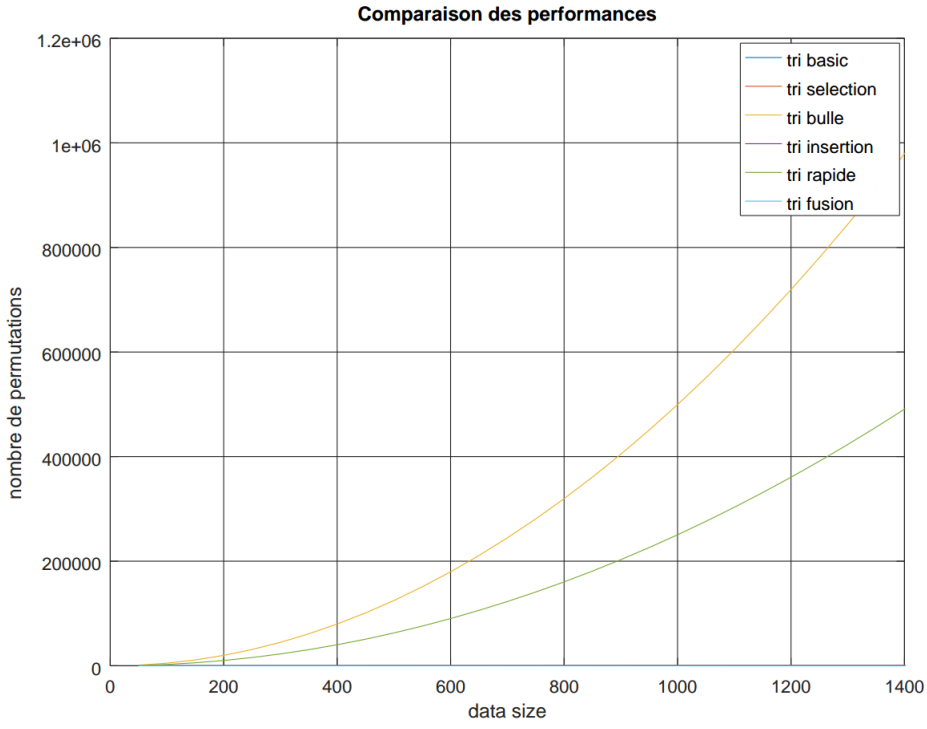

FIGURE 71 – nombre de permutations

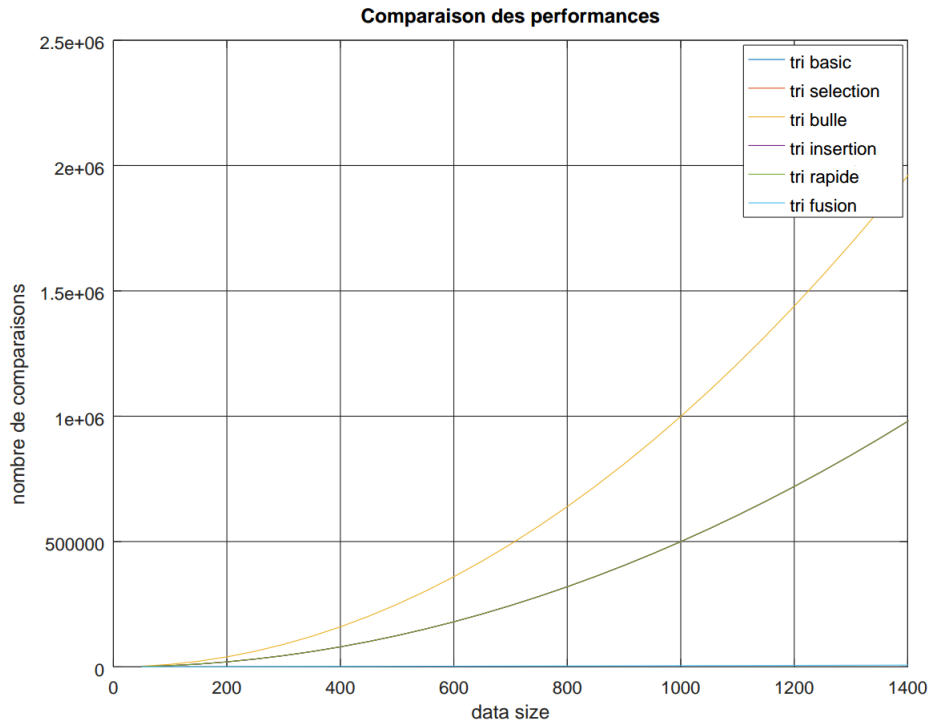

FIGURE 72 – nombre de comparaisons

Remarques

— **Dans le cas des tableaux triés dans l'ordre croissant**
Le tri insertion est le grand gagnant suivi de près par le tri fusion. Ces deux tris se montrent dans ce cas, de loin les plus performants, particulièrement le tri insertion qui affiche alors une complexité linéaire et ressort vaiqueur sur tous les critères. Tous les autres algorithmes de tri sont au coude à coude en terme de temps moyen d'exécution. Enfin on remarque qu'étrangement le tri rapide (d'habitude si performant) est en réalité le pire de tous dans ce cas.

— **Dans le cas des tableaux sont désordonnés**
Le tri rapide et le tri fusion sont de loin les meilleurs sur tous les critères avec leur complexité quasi linéaire, ensuite vient le tri insertion puis le tri selection (ou basique) et l'on remarque que le tri bulle arrive bon dernier et de très de loin.

— **Dans le cas des tableaux sont triés dans l'ordre décroissant**
Le tri fusion se retrouve seul en tête sur tous les critères. En effet, il conserve sa complexité quasi linéaire contrairement au tri rapide qui a encore une fois dégénéré en complexité quadratique et se retrouve ainsi au coude à coude avec les tris insertion et selection en terme de temps d'exécution. Le tri bulle quant à lui arrive en dernière position et de très loin (une fois de plus).

Conclusion

Parvenu au terme de ce TP, nous nous sommes familiarisé avec ces six différents algorithmes de tris, nous avons compris leur principe et nous les avons implémentés puis nous avons, pour chacun d'entre eux fait l'étude de ses performances et enfin nous les avons comparés. Il en ressort donc que généralement notre choix se portera sur le tri rapide ou sur le tri fusion. Toutefois après quelques recherches nous nous sommes aperçus que le tri rapide est celui qui est le plus utilisé car en pratique il est le plus rapide. Il existe des solutions pour palier au danger de le voir dégénerer comme par exemple mélanger le tableau avant de le passer à l'algorithme pour s'assurer qu'il ne soit pas déjà plus ou moins trié. Nous terminons en ajoutant que le tri bulle est les autres tris de complexité quadratique (à l'exception du tri insertion) ne sont plus vraiment utilisés en pratique et font généralement uniquement l'objet d'une étude à des fins pédagogiques.