**Analiza algoritmilor de sortare**

Pentru a testa algoritmii de sortare, am conceput acest input:

T = 9  
N = 1000 Max = 1000  
N = 1000 Max = 1000000  
N = 1000 Max = 1000000000  
N = 10000 Max = 1000000  
N = 10000 Max = 1000000000  
N = 1000000 Max = 1000  
N = 1000000 Max = 1000000000  
N = 10000000 Max = 1000000  
N = 10000000 Max = 1000000000

Pentru generarea efectivă a șirurilor de sortat, am ales două metode pentru a face randomizarea, prin care am reușit să evidențiez anumite diferențe semnificative: funcția rand() din <cstdlib> și generatorul pseudo-aleator Mersenne Twister 19937, pentru o dispunere mai uniformă a numerelor.

Așadar, mai jos sunt câte două rapoarte cu timpii de executare (în secunde) pentru fiecare algoritm de sortare per test, precum și câte un grafic care urmărește evoluția timpilor de executare.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Numere generate prin rand() #1st attempt | | | | | | | | | | |
| Test / Sortare | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| Insertion Sort | 0.007 | 0.007 | 0.007 | 0.752 | 0.735 | - | - | - | - |
| Heap Sort | 0 | 0.001 | 0 | 0.007 | 0.007 | 0.86 | 0.822 | 9.237 | 9.384 |
| Radix Sort (2^4) | 0 | 0.001 | 0.001 | 0.006 | 0.005 | 0.511 | 0.507 | 5.165 | 5.228 |
| Radix Sort (2^8) | 0.001 | 0.002 | 0.001 | 0.005 | 0.005 | 0.258 | 0.27 | 2.611 | 2.641 |
| Intro Sort (3) | 0 | 0 | 0 | 0.002 | 0.002 | 1.014 | 0.848 | 10.771 | 11.066 |
| Intro Sort (5) | 0 | 0 | 0 | 0.003 | 0.003 | 1.027 | 0.417 | 10.83 | 10.882 |
| Tim Sort | 0 | 0.001 | 0.001 | 0.01 | 0.01 | 1.408 | 1.39 | 16.455 | 16.936 |
| STL Sort | 0 | 0 | 0 | 0.002 | 0.002 | 0.185 | 0.293 | 2.954 | 3.019 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Numere generate prin rand() #2nd attempt | | | | | | | | | |
| Test / Sortare | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| Insertion Sort | 0.008 | 0.008 | 0.008 | 0.754 | 0.749 | - | - | - | - |
| Heap Sort | 0 | 0 | 0 | 0.007 | 0.007 | 0.867 | 0.817 | 9.457 | 9.623 |
| Radix Sort (2^4) | 0 | 0 | 0.001 | 0.005 | 0.006 | 0.545 | 0.512 | 5.316 | 5.418 |
| Radix Sort (2^8) | 0.001 | 0.001 | 0.001 | 0.004 | 0.004 | 0.266 | 0.268 | 2.654 | 2.64 |
| Intro Sort (3) | 0 | 0 | 0 | 0.002 | 0.002 | 1.058 | 0.872 | 10.935 | 11.048 |
| Intro Sort (5) | 0 | 0 | 0 | 0.003 | 0.003 | 1.033 | 0.426 | 10.871 | 10.865 |
| Tim  Sort | 0 | 0 | 0.001 | 0.01 | 0.01 | 1.437 | 1.447 | 17.056 | 16.975 |
| STL  Sort | 0 | 0 | 0 | 0.002 | 0.002 | 0.203 | 0.308 | 2.964 | 3.028 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Numere generate prin mt19937 #1st attempt | | | | | | | | | | |
| Test / Sortare | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| Insertion Sort | 0.007 | 0.007 | 0.007 | 0.739 | 0.749 | - | - | - | - |
| Heap Sort | 0 | 0 | 0 | 0.008 | 0.007 | 0.817 | 0.829 | 10.123 | 10.316 |
| Radix Sort (2^4) | 0 | 0 | 0.001 | 0.006 | 0.006 | 0.525 | 0.511 | 5.452 | 5.481 |
| Radix Sort (2^8) | 0.001 | 0.001 | 0.001 | 0.005 | 0.006 | 0.268 | 0.275 | 2.748 | 2.75 |
| Intro Sort (3) | 0 | 0 | 0 | 0.002 | 0.002 | 1.031 | 0.373 | 4.373 | 4.318 |
| Intro Sort (5) | 0 | 0 | 0 | 0.003 | 0.003 | 1.012 | 0.417 | 4.65 | 4.812 |
| Tim  Sort | 0 | 0 | 0 | 0.011 | 0.01 | 1.414 | 1.407 | 17.085 | 17.204 |
| STL  Sort | 0 | 0 | 0 | 0.002 | 0.003 | 0.191 | 0.397 | 4.204 | 4.716 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Numere generate prin mt19937 #2nd attempt | | | | | | | | | |
| Test / Sortare | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| Insertion Sort | 0.007 | 0.007 | 0.007 | 0.743 | 0.737 | - | - | - | - |
| Heap Sort | 0 | 0 | 0 | 0.007 | 0.008 | 0.889 | 0.872 | 10.203 | 10.105 |
| Radix Sort (2^4) | 0 | 0.001 | 0.001 | 0.006 | 0.006 | 0.531 | 0.528 | 5.395 | 5.372 |
| Radix Sort (2^8) | 0.001 | 0.001 | 0.001 | 0.005 | 0.006 | 0.265 | 0.28 | 2.736 | 2.722 |
| Intro Sort (3) | 0 | 0 | 0 | 0.002 | 0.002 | 1.015 | 0.376 | 4.294 | 4.314 |
| Intro Sort (5) | 0 | 0 | 0 | 0.003 | 0.003 | 1.018 | 0.426 | 4.592 | 4.689 |
| Tim  Sort | 0.001 | 0 | 0 | 0.01 | 0.01 | 1.476 | 1.487 | 17.344 | 17.406 |
| STL  Sort | 0 | 0 | 0 | 0.002 | 0.002 | 0.184 | 0.392 | 4.253 | 4.83 |

|  |  |
| --- | --- |
| Sortare | Timp de executare (în medie)(s) |
| Insertion Sort | irelevant (nu merge pe toate testele) |
| Heap Sort | 2.368 |
| Radix Sort (2^4) | 1.306 |
| Radix Sort (2^8) | 0.496 |
| Intro Sort (3) | 1.881 |
| Intro Sort (5) | 1.888 |
| Tim Sort | 4.116 |
| STL Sort | 0.892 |

**Observații**

* Radix Sort în baza 28 se dovedește cel mai eficient pe numere naturale și este de 2.63 ori mai rapid decât același algoritm în baza 24**.** Acesta devine mai rapid chiar și decât sortarea nativă din C++ pe testele mai mari (după testul 6, evidențiat cu albastru în tabele).
* Cele două variante de Intro Sort sunt foarte similiare ca timpi de executare, Intro Sort cu pivotul mediana din 3 nu este semnificativ mai rapid. Tot legat de Intro Sort, putem observa că este mult mai lent în cazul 6 (în care N = 106 și Max = 103), decât în cazul 7 (în care N = 106 și Max = 109). De asemenea, la șirurile generate prin mt19937 se descurcă de aproximativ 2.5 ori mai bine decât la cele generate cu funcția rand(), pe testele mai mari (evidențiat cu roșu în tabele).