**Raport Sortari-Proiect Structuri de Date**

La proiectul acesta am imlpementat 5 sortari si anume:

* Radix Sort: O(10\*n(numarul de cifre din nmax),pentru baza 10
* Merge Sort: O(n\*logn)
* Shell Sort:O(n\*logn)
* Quick Sort:O(n\*logn)
* Buble Sort:O(n^2)

Pe parcursul acestei prezentari voi atase print-screen-uri cu testele pe care le-am rulat in IDE-ul CodeBlocks.

Pentru fiecare am generat numere random, mai putin atunci cand am sprecificat ca sirul este ordonat crescator sau descrescator; avand maxim un nmax citit de la tastura, ce reprezinta numarul maxim gasit in sir, si n numere generate, n de asemenea citit de la tastura.

Pentru fiecare test am comparat cele 5 sortari implementate, dar si algoritmul de sortare nativ din c++.

1. A picture containing text

   Description automatically generated
2. Text

   Description automatically generated with medium confidence
3. Text

   Description automatically generated with medium confidence
4. Text

   Description automatically generated with medium confidence
5. Text

   Description automatically generated
6. Text

   Description automatically generated with medium confidence
7. Text

   Description automatically generated
8. Text

   Description automatically generated
9. Text

   Description automatically generated with medium confidence
10. Text

    Description automatically generated with low confidence
11. Text

    Description automatically generated with medium confidence
12. Text

    Description automatically generated with medium confidence
13. Text

    Description automatically generated with low confidence

Acestea sunt 12 teste efectuate. Se observa ca bubble-sortul este mult mai ineficient din punct de vedere al timpului pe langa celelalte sortari, exceptie fiind cazul in care sirul este sortat crescator. Toate celelalte reusesc sa duca la capat executia , ca si exemplu de timp , privind la testul 3, avand the worst case buble sortul un sir ordonat descrescator, iar n=10^6, iar procesorul reusind sa efectueze maxim 10^8 calcule pe secunda , sortarea ar dura in jur de 3 ore, iar celalte sortari reusind in mai putin de 0.2 secunde.

Text

Description automatically generated with medium confidence

In celelalte teste se observa cum radix Sortul si Quick Sortul si Merge Sortul sunt apropriate ca timp.

Intre Quick sort si Merge sort aproape de fiecare Quick Sortul are un timp mai bun ( asta si datorita alegerii bune a pivotului, acesta fiind mediana din 3 , vom discuta mai incolo despre Quick Sort).

Intre Quick Sort si Radix Sort, cu n-ul mai mare de 10^5 si n=nmax deja Radix sortul are timpi putin mai buni, dar cu fluctuatii de la caz la caz. Este clar ca radix-sortul nu poate degenera comprativ cu Quick Sortul care poate degenera cu o probabilitate extrem de mica in O(n^2), dar tine mult de alegerea pivotului.

Pentru oricat ar fi n, cu nmax<=10^4 preferam Radix sortul, mai ales cand n este cat mai mare. Cand n este mai mic de 10^5 si avem nmax>=10^6 preferam Quick Sortul.

Sortarea nativa c++, desi are mai multe sortari combinate in implementarea ei, a avut un timp mai prost decat Merge Sort, Quick Sort si Radix Sort.

Mai departe voi afisa cateva teste doar cu Quick Sort , dar cu pivoti diferiti:

1. Text

   Description automatically generated
2. Text

   Description automatically generated
3. Text

   Description automatically generated
4. Text

   Description automatically generated
5. Text

   Description automatically generated
6. Text

   Description automatically generated
7. Text

   Description automatically generated
8. Text

   Description automatically generated
9. Text

   Description automatically generated

Avem 4 quick sorturi cu 4 pivoti diferiti:

* Mediana a 3
* Mediana medianelor
* Generat random
* Elementul de la inceput

Se observa in toate aceste teste cum pivotul care mediana a 3 si mediana medianelor ca au cei mai buni timpi de executie tot timpul.

La numere generate random, pivotul ales random are cei mai slabi timpi, iar o alta problema mare este cand sirul este deja sortat crescator sau descrescator pentru pivotul ales primul sau ultimul. Algoritmul va face foarte multe recursii astfel incat va genera o scurgere de memorie. Asdar este de preferat sa alegem pivotul fie mediana a 3 sau a 5 sau mediana medianelor. La toate testele anterioare am folosit pivotul ca fiind mediana a 3.

Totdata quick Sortul, merge Sortul, shell Sortul se decurca foarte bine atunci cand sirul are acelasi element peste tot.