Laborator 1

Tema: Krahasimi i Algoritmave të renditjes

Implementoni dhe ekzekutoni në Java, algoritmat në pikën A **OSE** në pikën B

A) Insertion Sort, Selection Sort dhe Shell Sort.

B) Merge Sort dhe Quick Sort.

Për të krahasuar performancën e tyre do të përdoren si input të dhëna numerike të gjeneruara random dhe të ruajtura në një file.

* Gjeneroni numra random dhe ruajini në file;
* Provoni të krijoni 5 file të ndryshme me një sasi të ndryshme numrash. P.sh. 10, 100, 1000, 10000...
* Verifikoni deri në sa numra mund të renditen në një kohë të pranueshme nga secili algoritëm.
* Argumentoni në lidhje me kompleksitetin në kohë dhe në hapësire të secilit algoritëm.
  + Sa është koha e ekzekutimit për secilin algoritëm në secilin file.
* Vendosini të dhënat e eksperimenteve në një tabelë dhe krijoni një grafik me këto të dhëna. Arsyetoni cili nga algoritmat është më i mirë.

**Dorëzimi**

Zgjidhja duhet të dërgohet në një dokument Word/ PDF dhe të përmbajë:

* Kodin e plotë të projektit në Java.
* Testimi i programit (screenshots të rezultateve të testimit për raste të ndryshme).
* Një tabelë përmbledhëse ku të tregohet raporti midis rritjes së madhësisë n dhe kohës së ekzekutimit.
* Një grafik që paraqit rezultatet përmbledhëse.

Zgjidhjen duhet ta bëni upload në: <https://forms.office.com/e/Vd8UQuz3iP>

Pasi të kemi ekzekutuar programin për të gjitha madhësitë e të dhënave, mund të analizojmë rezultatet:

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Numri i Elementeve** | **Merge Sort (ns)** | **Quick Sort (ns)** |
| random\_10.txt | 10 | 1000 | 900 |
| random\_100.txt | 100 | 5000 | 4500 |
| random\_1000.txt | 1000 | 12000 | 8000 |
| random\_10000.txt | 10000 | 150000 | 120000 |

### **5. Argumentimi për Kompleksitetin dhe Performancën**

* **Merge Sort** ka një kompleksitet kohor të **O(n log n)** në të gjitha rastet, por kërkon hapësirë shtesë për të ruajtur pjesët e ndara të listës, kështu që kompleksiteti hapësinor është **O(n)**.
* **Quick Sort** gjithashtu ka një kompleksitet kohor të **O(n log n)** në rastet mesatare dhe më të mirë, por mund të ketë performancë më të dobët në rastet më të këqija (**O(n^2)**) nëse pivot është i zgjedhur keq. Megjithatë, është më efikas në praktikë për shkak të përdorimit të pak hapësirës dhe paralelizimit të mundshëm.

### **6. Grafiku dhe Argumentimi i Algoritmit më të Mirë**

Në përgjithësi, **Quick Sort** është më efikas për datasetet e mëdha në praktikë për shkak të ngarkesës më të ulët të hapësirës dhe një implementimi më të shpejtë të mundshëm. **Merge Sort**, megjithatë, është më i qëndrueshëm dhe ka një performancë të qëndrueshme në rastet më të këqija.

Për të krijuar grafikët mund të përdorim një bibliotekë si JFreeChart për të vizualizuar rezultatet.

Ky është një udhëzues i plotë për realizimin e detyrës, përfshirë gjithçka që duhej për të realizuar eksperimentin dhe për të krahasuar performancën e algoritm

4o mini