Raport Proiect 1

Implementing heapsort versus selection sort as a single algorithm - with two different data structures

Descrierea codului

Limbajul de programare folosit pentru acest proiect este Python, pentru a profita de cateva avantaje ale limbajului in ceea ce priveste testarea codului. Am implementat doua clase, `Vector` si `Heap`, ce se folosesc la baza de liste pentru a memora date. Principalele diferente dintre cele doua clase sunt:

- `Vector` reprezinta doar un wrapper peste listele din Python, in timp ce `Heap` isi reprezinta intern datele sub forma unui heap stocat intr-o lista;
- In `Vector` poate fi modificat orice element individual al structurii de date (elementele sunt read-write), in timp ce in `Heap` elementele individuale nu pot fi modificate (sunt read-only) deoarece modificarea acestora ar putea strica proprietatea de heap a structurii de date;
- Clasa `Heap` are o metoda suplimentara `heapify()` ce este apelata automat la initializarea unui obiect sau dupa aplicarea `FindMaxAndSwap()` pentru a mentine proprietatea de heap.

Ambele clase contin propria implementare a operatiei `FindMaxAndSwap`: `Vector` o implementeaza in O(n), in timp ce `Heap` o implementeaza in O(1), insa cu adaugarea costului unui apel la `heapify()` de O(log n).

Functia de sortare a celor doua colectii este comuna si apeleaza metoda `FindMaxAndSwap()` corespunzatoare fiecarei clase in parte. La baza, functia de sortare foloseste algoritmul Selection Sort.

In cadrul claselor, functiile si variabilele care incep cu `_` sunt considerate private si nu ar trebui accesate in afara claselor. In limbajul Python nu pot fi declarate explicit atribute si metode private, iar din aceasta cauza am folosit conventia precedenta.

Teste si experimente

Ca experiment, am generat 5 permutari de cate 20000 de elemente, iar apoi am cronometrat cat dureaza sa fie construita si sortata permutarea in ambele structuri de date. Rezultatele obtinute sunt urmatoarele:

```
Test 0:
Generated permutation with 20000 elements.
Sort on vector performed in 23.1581 seconds.
Sort on heap performed in 0.1185 seconds.

Test 1:
Generated permutation with 20000 elements.
Sort on vector performed in 23.8396 seconds.
Sort on heap performed in 0.1183 seconds.

Test 2:
Generated permutation with 20000 elements.
Sort on vector performed in 23.5129 seconds.
Sort on heap performed in 0.1197 seconds.

Test 3:
Generated permutation with 20000 elements.
Sort on vector performed in 23.2833 seconds.
Sort on heap performed in 0.1174 seconds.

Test 4:
Generated permutation with 20000 elements.
Sort on vector performed in 23.9143 seconds.
Sort on heap performed in 0.1204 seconds.
```

Se poate observa ca sortarea pe heap este de aproximativ 200 de ori mai rapida decat sortarea pe un vector obisnuit.