Tema Nr. 8: Parcurgere iterativă vs recursivă arbore binar. Hibridizare quicksort. Analiza comparativă a timpului de execuție.

Timp alocat: 2 ore

Implementare

Se cere implementarea **corectă** și **eficientă** a *parcurgerii iterative si recursive* al unui arbore binar, respectiv *hibridizare pentru quicksort*

Informații utile și pseudocod găsiți în notițele de la curs și seminar:

- *Parcurgere recursivă și iterativă a unui arbore binar în O(n)*
- *Hibridizare quicksort utilizând insertion sort iterativ - in quicksort, pentru dimensiuni de șir < prag, se utilizeaza insertion sort (folosiți implementare quicksort din tema 3 și insertion sort din tema 1).*

Cerințe

1. **Implementare a parcurgerii iterative și recursive a unui arbore binar în *O(n)* și cu *memorie aditională constantă* (5p)**

Corectitudinea algoritmilor va trebui exemplificată pe date de intrare de dimensiuni mici.

2. **Analiza comparativă a parcurgerii *iterative și recursive* din perspectiva numărului de operații (2p)**

! Înainte de a începe să lucrați pe partea de evaluare, asigurați-vă că aveți un **algoritm corect**!

Pentru analiza comparativă a versiunii iterative vs recursive, trebuie sa *numărați doar operațiile de afișare a cheii* variind numărul de noduri din arbore în intervalul [100…10000] cu un pas de maxim 500 (recomandăm 100).

Pentru partea de construcție al arborelui, puteti sa porniti de la un tablou a cărui dimensiune variază și să alegeți în mod aleatoriu rădăcina.

3. **Implementarea hibridizării quicksort-ului (1p)**

Corectitudinea algoritmilor va trebui exemplificată pe date de intrare de dimensiuni mici.

4. **Analiză comparativă (între *quicksort și quicksort hibridizat*) din punct de vedere a numărului de operații și a timpului efectiv de execuție (1p)**

! Înainte de a începe să lucrați pe partea de evaluare, asigurați-vă că aveți un **algoritm corect**!

Corectitudinea algoritmilor va trebui exemplificată pe date de intrare de dimensiuni mici.

Pentru hibridizare quicksort, trebuie să utilizați versiunea iterativă de insertion sort din prima temă în cazul în care dimensiunea vectorului este mică (sugerăm utilizarea insertion sort dacă vectorul are sub 30 de elemente). Comparați *timpul de rulare și numărul de operații* (asignări + comparații) pentru quicksort implementat în tema 3 cu cel hibridizat.

Pentru a măsura timpul de execuție puteți folosi Profiler similar cu exemplul de mai jos.

```
profiler.startTimer("your_function", current_size);
for(int test=0; test<nr_tests; ++test) {
   your_function(array, current_size);
}
profiler.stopTimer("your_function", current_size);
```

În momentul în care măsurați timpul de execuție, asigurați-vă că opriți orice alte procese care nu sunt necesare.

5. **Determinare a unui prag optim în hibridizare + motivație (*grafice/măsuratori*) (1p)**

Determinarea optimului din perspectiva pragului utilizat se realizează prin varierea valorii de prag pentru care se aplică insertion sort.

Comparați rezultatele obținute din perspectiva performanței (*timpului de execuție și a numărului de operații*) pentru a determina o valoare optima a pragului. Puteți folosi 10,000 ca dimensiune fixă a vectorului ce urmează sa fie sortat și variați pragul între [5,50] cu un increment de 1 până la 5.

Numărul de teste care trebuie sa fie repetate (*nr_tests* din exemplul de mai sus) trebuie ales în funcție de procesor și modul de compilare. Sugerăm valori mai mari, precum 100 sau 1000.