1 Tema Nr. 3: Analiza și compararea metodelor avansate de sortare – HeapSort și QuickSort / QuickSelect

Timp alocat: 2 ore

1.1 Implementare

Se cere implementarea **corectă** și **eficientă** a Sortării Rapide (*Quicksort*), Sortării Rapide Hibridizate (*Hybrid Quicksort*) și *Quick-Select* (*Randomized-Select*). Se cere și analizarea comparative a complexității Sortării folosind Heapuri *(Heapsort,* implementat în Tema Nr. 2*)* și Sortarea Rapidă *(Quicksort).*

Informații utile și pseudo-cod găsiți în notițele de curs sau în carte(**?**):

- *Heapsort*: capitolul 6 (Heapsort)
- *Quicksort*: capitolul 7 (Quicksort)
- *Hibridizare quicksort utilizând insertion sort iterativ* - in quicksort, pentru dimensiuni de șir $<$ prag, se utilizeaza insertion sort (folosiți implementarea insertion sort din Tema Nr. 1).
- *QuickSelect/Randomized Select:* capitolul 9

1.2 Cerințe minimale pentru notare

- Interpretați graficul și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu nu preluam teme unde tot codul este pus in main)
- ***Punctajele din barem se dau pentru rezolvarea corectă și completă a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de voi la întrebările puse de către profesor.***

1.3 Cerințe

1.3.1 QuickSort: implementare (2p)

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

1.3.2 QuickSort: evaluare în caz mediu statistic, favorabil si defavorabil (3p)

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

Pașii de analiză:

- variați dimensiunea șirului de intrare (n) între [100...10000], cu un increment de maxim 500 (sugerăm 100).

- pentru fiecare dimensiune (n), generați date de intrare adecvate metodei de construcție; rulați metoda numărând operațiile elementare (atribuiri și comparații - pot fi numărate împreună pentru această temă).

! Doar atribuirile și comparație care se fac pe datele de intrare și pe datele auxiliare corespunzătoare se iau în considerare.

1.3.3 QuickSort și HeapSort: analiză comparativă a cazului mediu statistic (2p)

Se cere compararea celor două metode de sortare în cazul **mediu statistic**. Pentru cazul **mediu statistic** va trebui să repetați măsurătorile de m ori (m=5) și să raportați valoarea lor medie; de asemenea, pentru cazul **mediu statistic**, asigurați-vă că folosiți **aceleași** date de intrare pentru cele două metode.

Generați un grafic ce compară cele două metode de construcție în cazul **mediu statistic** pe baza numărului de operații obținut la pasul anterior.

Dacă o curba nu poate fi vizualizată corect din cauza că celelalte curbe au o rată mai mare de creștere, atunci plasați noua curbă pe un alt grafic. Denumiți adecvat graficele și curbele.

1.3.4 Implementarea hibridizării quicksort-ului (1p)

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

1.3.5 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.

1.3.6 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.

1.3.7 Bonus: QuickSelect - Randomized-Select (0.5p)

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

Pentru QuickSelect (Randomized-Select) nu trebuie facuta analiza complexității, doar corectitudinea trebuie exemplificată.