1 Tema Nr. 2: Analiza și Compararea a două metode de construire a structurii de date Heap: "De jos în sus" (Bottom-up) vs. "De sus în jos" (Top-down)

Timp de lucru: 2 ore

1.1 Implementare

Se cere implementarea **corectă** și **eficientă** a două metode de construire a structurii de date Heap i.e., "de jos în sus" (*bottom-up*) și "de sus în jos" (*top-down*). De asemenea, se cere implementarea algoritmului *heapsort.*

Informații utile și pseudo-cod găsiți în notițele de curs sau în bibliografie[1]:

- "*De jos în sus*": secțiunea 6.3 (Building a heap)
- *Heapsort:* sectiunea 6.4 (The heapsort algorithm)
- "*De sus în jos*": secțiunea 6.5 (Priority queues) și problema 6-1 (Building a heap using insertion)

1.2 Cerințe minimale pentru notare

Lipsa oricărei cerințe minimale (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo*: Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.
- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu prelăum teme unde tot codul este pus în main).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerinței, calitatea interpretărilor din comentariul bloc și* ***răspunsul corect dat de dumeavostră la întrebările puse de către profesor.***

1.3 Cerințe

1.3.1 Analiza comparativă a *unuia* din algoritmii de sortare din L1 (la alegere) în versiune *iterativă* vs *recursivă*. Analiza se va efectua atât din *perspectiva numărului de operații, cât și a timpului de rulare* (2p)

Demo: Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

Pentru analiza comparativă a versiunii iterative vs recursive, alegeți oricare din cei 3 algoritmi din laboratorul 1 (bubble sort, insertion sau selection). Folosiți varianta iterativă pe care ați implementat-o și predat-o în cadrul laboratorului (corectată, dacă este nevoie, în funcție de feedback-ul pe care l-ați primit) și implementați același algoritm de sortare în mod recursiv.

Trebuie să măsurați atât efortul total, cât și timpul de rulare al celor două versiuni (iterativ și recursiv) => două grafice, fiecare comparând cele două versiuni de algoritm.

1.3.2 Implementarea metodei bottom-up de construire a unui heap (2p)

Demo: Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

1.3.3 Implementarea metodei top-down de construire a unui heap (2p)

Demo: Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

1.3.4 Analiza comparativă a celor două metode de construire în cazul mediu statistic (2p)

Se cere compararea celor două metode de construcție a structurii heap 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.

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).

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 curbă nu poate fi vizualizată corect din cauza că celelalte curbe au o rată

mai mare de creștere, atunci plasați noua curbă și pe un alt grafic. Denumiți adecvat graficele și curbele.

1.3.5 Analiza comparativă a metodelor de construcție în cazul defavorabil (1p)

1.3.6 Implementarea și exemplificarea corectitudinii algoritmului heapsort (1p)

Demo: Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

1.4 Informații adiționale

> **!** Doar atribuirile (=) și comparațiile (<, ==, >, !=) care se fac pe datele de intrare și pe datele auxiliare corespunzătoare se iau în considerare.

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);
```

Numărul de teste (*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.

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

> **!** *Observație.* Pentru a evalua cât mai corect timpul, Profiler nu va face numărătoarea operațiilor. Astfel, evaluarea de timp și de operații trebuie făcute separat.

References

[1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.