PROJEKTOVANJE I ANALIZA ALGORITAMA

Laboratorijska vežba 2

Ristovski Nikola 19347

Tehnologija : C#

KORIŠĆENI RESURSI

Za formiranje fajlova sa random integer elementima koristio sam python skriptu:

```
import random


def generate_and_save_random_integers(file_name, range_max, count):

with open(file_name, 'w') as file:

for _ in range(count):

number = random.randint(0, range_max)

file.write(f"{number}\n")


print(f"{count} random integers (with repeats) written to {file_name}")


generate_and_save_random_integers("Nums10MRepeat.txt", 10000, 10000000)
```

VAŽNO

Dok sam radio predašnju i ovu lab vežbu uočio sam da pri merenju vremena izvršenja, prvi poziv funkcije uvek zahteva izuzetno puno vremena u odnosu na očekivano. Kao primer, ako pokrenemo sorting za 100, 1000, 10.000 itd elemenata bubble sort algoritmom, i ako dobijemo redom 0.2ms, 2ms, 200ms ... uočavamo pattern da se svaki put povećava vreme sortiranja 100 puta, što i očekujemo jer je niz 10 puta veći, a pošto je bubble sort $O(n^2)$ shvatamo da će izvršenje trajati baš 100 puta duže. Tu nam smeta jedino činjenica da je za 100 elemenata potrebno 0.2ms, a očekivano po pattern-u je 0.02ms. Shvatio sam tražeći rešenje ovog problema da je u pitanju neko keširanje funkcije, ili tome slično, pa je za prvi poziv jedne te iste f-je potrebno više vremena. Upravo zbog tog razloga dalje u projektu videćete da sam pre pozivanja funkcije za 100 elemenata pozivao test „dummy“ funkciju za 22 elementa čisto da bismo nakon njenog izvršenja dobijali korektne proporcionalne veličine vremena izvršenja za ostale nizove od 100, 1000, 10.000 itd elemenata.

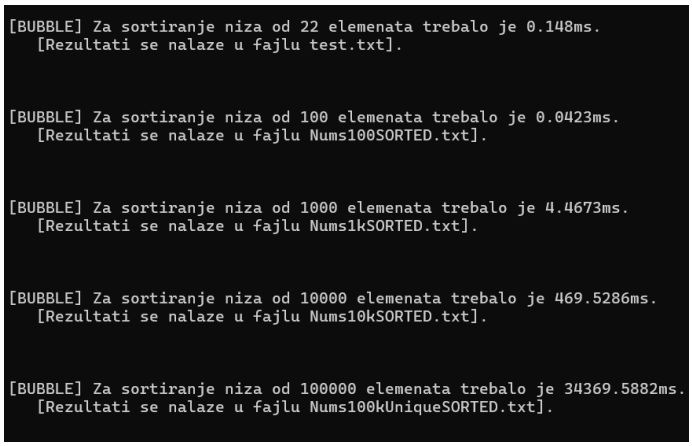
```
[BUBBLE] Za sortiranje niza od 100 elemenata trebalo je 0.201ms.
   [Rezultati se nalaze u fajlu Nums100SORTED.txt].


[BUBBLE] Za sortiranje niza od 1000 elemenata trebalo je 4.1304ms.
   [Rezultati se nalaze u fajlu Nums1kSORTED.txt].


[BUBBLE] Za sortiranje niza od 10000 elemenata trebalo je 464.2391ms.
   [Rezultati se nalaze u fajlu Nums10kSORTED.txt].


[BUBBLE] Za sortiranje niza od 100000 elemenata trebalo je 34238.2862ms.
   [Rezultati se nalaze u fajlu Nums100kUniqueSORTED.txt].
```

```
[BUBBLE] Za sortiranje niza od 22 elemenata trebalo je 0.148ms.
   [Rezultati se nalaze u fajlu test.txt].


[BUBBLE] Za sortiranje niza od 100 elemenata trebalo je 0.0423ms.
   [Rezultati se nalaze u fajlu Nums100SORTED.txt].


[BUBBLE] Za sortiranje niza od 1000 elemenata trebalo je 4.4673ms.
   [Rezultati se nalaze u fajlu Nums1kSORTED.txt].


[BUBBLE] Za sortiranje niza od 10000 elemenata trebalo je 469.5286ms.
   [Rezultati se nalaze u fajlu Nums10kSORTED.txt].


[BUBBLE] Za sortiranje niza od 100000 elemenata trebalo je 34369.5882ms.
   [Rezultati se nalaze u fajlu Nums100kUniqueSORTED.txt].
```

KAKO SE POREDE SORTING ALGORITMI

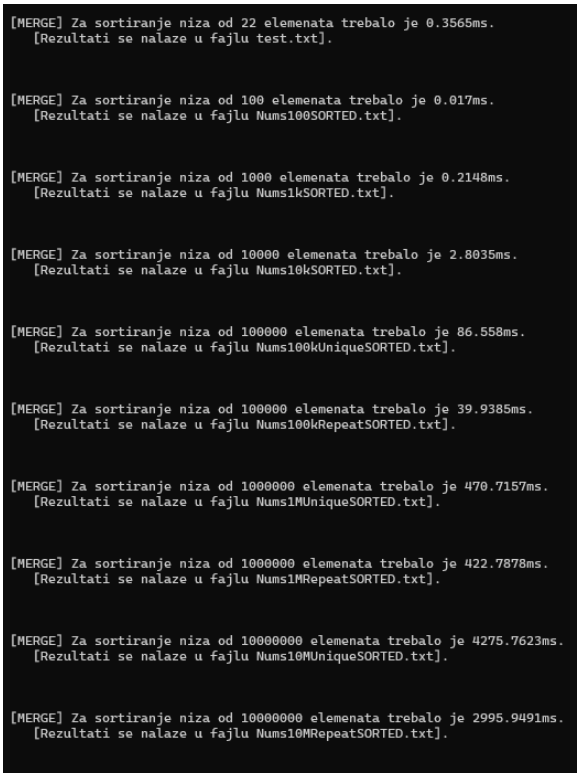
BUBBLE SORT

Bubble sort algoritam je složenosti $O(n^2)$. Kako smo imali situaciju da nam svaki od nizova koje testiramo ima 10 puta više elemenata nego pređašnji, jasno nam je da je očekivano da svaki od njih međusobno traje $10^2 = 100$ puta duže odnosno kraće. To možemo i uočiti na sledećoj slici (zanemarimo niz od 22 elementa zbog razloga koji su malopre navedeni).



Jasno uočavamo da vreme potrebno za izvršenje raste sa 0.04ms na oko 4.4ms što jeste oko 100 puta duže, pa opet na 469ms, opet prati naše pravilo. Vidimo da ipak nije tačno 100 puta svakog puta, u poslednjem uzorku od 100.000 elemenata bilo je potrebno oko 34 hiljade ms, to je malo manje od povećanja od 100 puta. To je sasvim korektno jer trajanje algoritma zavisi i od same strukture niza koji je sortiran, a i od drugih faktora, pa je ovo ponašanje u redu. Nisam nastavio sa sortiranjem nizova od milion i 10 miliona elemenata, jer bi to po našoj kalkulaciji trajalo, za milion oko 56 minuta, a za 10 miliona onda 100 puta duže, oko 5600 minuta.

MERGE SORT



Merge sort je algoritam složenosti O(nlogn). Uzmimo na primer nizove od 1000 i 10.000 elemenata, i ustanovimo koliko bi trebalo da duže traje ovaj od 10.000.

(10.000 * lg10.000 / 1000 * lg1000) = oko 13.3.

Dakle, očekivano trajanje za 10k elemenata je vreme za 1k elemenata pomnoženo sa 13.3, i to 0.215 * 13.3 = 2.86.

Jasno nam je da to i uočavamo u konzoli.

U tekstu zadatka je glasilo da koristimo cele brojeve od 0 do 10.000 u našim nasumično generisanim fajlovima, ali je mene interesovalo da li postoji neka razlika i kada koristimo samo jedinstvene brojeve, zato ovde imamo 2 rezultata za 100k, 1M i 10M elemenata. Vidimo da je otprilike slično izvršenje osim kod 10M gde je za 1000ms duže ako su jedistveni.

Još nešto što smeta je rezultat za jedinsveni fajl od 100k elemenata za koji je potrebno čak 86ms. Nemam objašnjenje za ovu pojavu, i ne mislim da je do algoritma već do mašine ili visual studia.

RADIX SORT

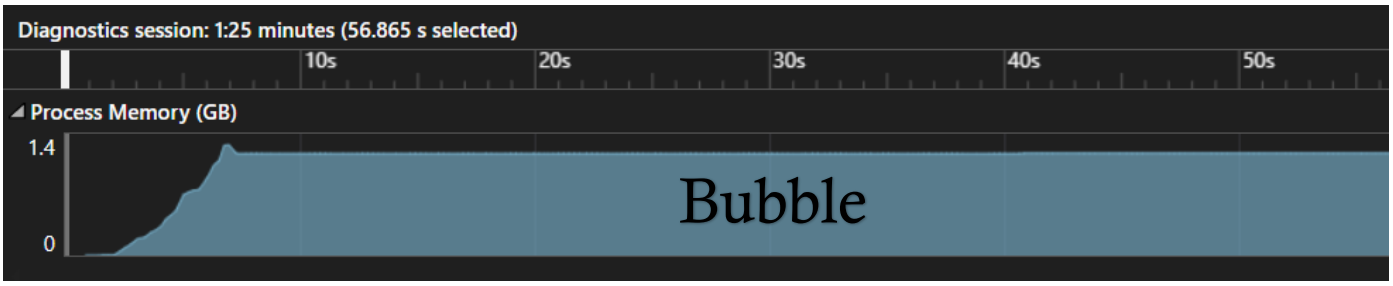
Radix sort je linearni sort algoritam, pa očekujemo da uvek trajanje sortinga bude 10 puta duže jer su nam nizovi 10 puta veći.

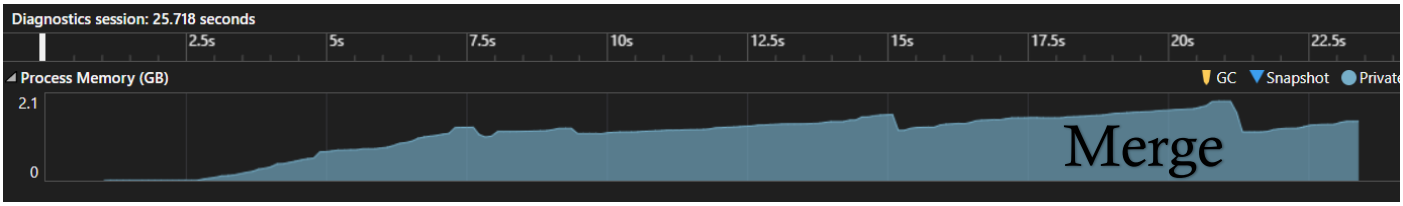
On koristi neki stabilan algoritam koji održava redosled sortiranih elemenata koji imaju istu cifru, a ja sam konkretno koristio Counting sort.

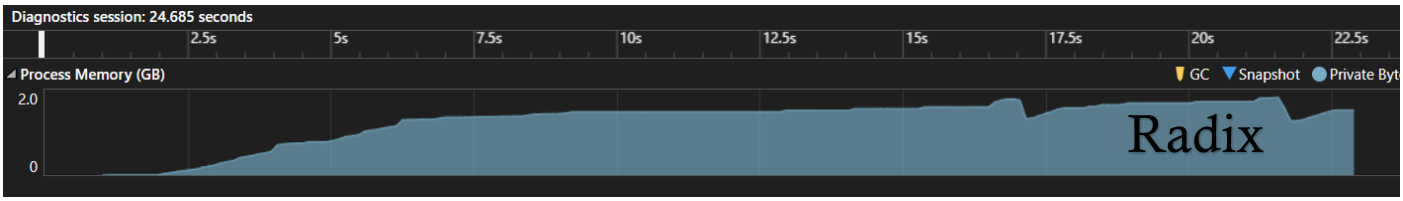
Rezultat iz konzole potvrdjuje ovo očekivanje i vidimo da optrilike važi za svaki slučaj testiranja. Jasni izuzeci su nizovi sa jedinstvenim elementima. To je i logično ako poznajemo kako radix sort radi. On sortira cifru po cifru, a kako kod unikatnih elemenata je neophodno da brojevi budu u opsegu od 0 do veličine niza, kod recimo elemenata sa 1M elemenata naići ćemo na elemente tipa 999.999, oni imaju više cifara nego elemeti niza od 1M elemenata kod kog se elementi ponajvljaju, jer su u opsegu 0-10.000, pa je najveći mogući broj tu 9.999. Ovaj niz unikatnih elemenata će zato imati za 2 više sortiranja od ovog gde se elemeti ponavljaju, i zato i vreme izvršenja raste.

```
[RADIX] Za sortiranje niza od 22 elemenata trebalo je 29.3466ms.
   [Rezultati se nalaze u fajlu test.txt].


[RADIX] Za sortiranje niza od 100 elemenata trebalo je 0.0476ms.
   [Rezultati se nalaze u fajlu Nums100SORTED.txt].


[RADIX] Za sortiranje niza od 1000 elemenata trebalo je 0.3179ms.
   [Rezultati se nalaze u fajlu Nums1kSORTED.txt].


[RADIX] Za sortiranje niza od 10000 elemenata trebalo je 3.4374ms.
   [Rezultati se nalaze u fajlu Nums10kSORTED.txt].


[RADIX] Za sortiranje niza od 100000 elemenata trebalo je 46.9327ms.
   [Rezultati se nalaze u fajlu Nums100kUniqueSORTED.txt].


[RADIX] Za sortiranje niza od 100000 elemenata trebalo je 30.8514ms.
   [Rezultati se nalaze u fajlu Nums100kRepeatSORTED.txt].


[RADIX] Za sortiranje niza od 1000000 elemenata trebalo je 620.105ms.
   [Rezultati se nalaze u fajlu Nums1MUniqueSORTED.txt].


[RADIX] Za sortiranje niza od 1000000 elemenata trebalo je 389.9857ms.
   [Rezultati se nalaze u fajlu Nums1MRepeatSORTED.txt].


[RADIX] Za sortiranje niza od 10000000 elemenata trebalo je 7122.8789ms.
   [Rezultati se nalaze u fajlu Nums10MUniqueSORTED.txt].


[RADIX] Za sortiranje niza od 10000000 elemenata trebalo je 3325.0104ms.
   [Rezultati se nalaze u fajlu Nums10MRepeatSORTED.txt].
```

MEMORIJA







Ako uporedimo iskorišćenost memorije ova tri algoritma, vidimo da je najbolja iskorišćenost kod Bubble sorta. Primećujemo da s povećanjem veličine niza raste i iskorišćenost memorije, što je i očekivano. Bubble sort je in-place algoritam koji ne zahteva dodatnu memoriju za izvršenje. Merge sort koristi u mojoj implementaciji dodatni niz samo u Combine funkciji koja spaja 2 sortirana podniza u jedan veliki niz, ali je velika mana po memoriju što je algoritam rekurzivni, što utiče na iskorišćenost stack-a. Kod Radix sorta, imamo 2 dodatna niza, jedan koji broji ponavljanje odredjenog elementa u nizu, a drugi koji je rezultat niz. Zato kod njih primećujemo veću potrošnju memorije u odnosu na Bubble sort.

PROBLEM 2 – PRODAVNICA SLATKIŠA

Kako ste rekli da su slatkiši jedinstveni u nizu, tj pojavljivanje njihovih cena koja ih reprezentuje, nije moguće kreirati niz od 100k, 1M i 10M elemenata koji koriste cele brojeve iz opsega 0-10.000 a da elementi budu unikatni. Zato sam za niz od 100k elemenata koristio opseg 0-100.000, za 1M sam koristio opseg od 0 do milion itd. kako bi slatkiši bili unikatni. Ako ipak mora da bude zadovoljen uslov da elementi smeju biti korišćeni samo iz opsega 0-10k, dovoljno je u programu parametar koji predstavlja niz zameniti, i umesto recimo Nums100kUnique staviti Nums100kRepeat.

Zadatak prodavnice slatkiša koristi sortiranje niza kako bi pronašao koliko je najmanje novca potrebno potrošiti kako bismo probali sve slatkiše. Ako nam prodavac uz kupovinu jednog slatkiša da k slatkiša besplatno, k je 20% od dužine niza tj broja slatkiša, normalno je da ćemo kao besplatne uzimati one najskuplje slatkiše prvo. Tako ćemo svaki put kupiti najjeftiniji slatkiš koji već nismo kupili, da bismo odabrali besplatno k najskupljih slatkiša koje opet do tada nismo probali. Ključna stvar u ovog algoritmu jeste upravo sortiranje. Ako pogledamo bubble sort, vidimo da je otprilike zadovoljeno povećanje složenosti. Nizovi od 100 i 1000 elemenata pokazuju to svojstvo, dok niz od 10k elemenata traje skoro duplo kraće nego očekivana gornja granica, a to vidimo i kod niza od 100k elemenata. Opet, gornja granica je samo gornja granica i može se desiti da je potrebno manje vremena. Kod nizova od 100 i 1000 elemenata pretpostavljam da je odredjivanje najniže cene imalo veći uticaj na vreme izvršenja nego kod 10k i 100k, pa je zato malo duže nego sam sort. Opet kao i kod sortiranja nisam pustio da se izvrše nizovi od 1M i 10M elemenata jer bi trajalo izuzetno dugo.

```
[PROBLEM2 BUBBLE] Novac potreban za sve slatkise: 12
Za PROBLEM2 nizom od 22 elemenata trebalo je 0.2209ms.


[PROBLEM2 BUBBLE] Novac potreban za sve slatkise: 1979
Za PROBLEM2 nizom od 100 elemenata trebalo je 0.0781ms.


[PROBLEM2 BUBBLE] Novac potreban za sve slatkise: 283
Za PROBLEM2 nizom od 1000 elemenata trebalo je 6.8038ms.


[PROBLEM2 BUBBLE] Novac potreban za sve slatkise: 15
Za PROBLEM2 nizom od 10000 elemenata trebalo je 380.7182ms.


[PROBLEM2 BUBBLE] Novac potreban za sve slatkise: 15
Za PROBLEM2 nizom od 100000 elemenata trebalo je 34896.023ms.
```

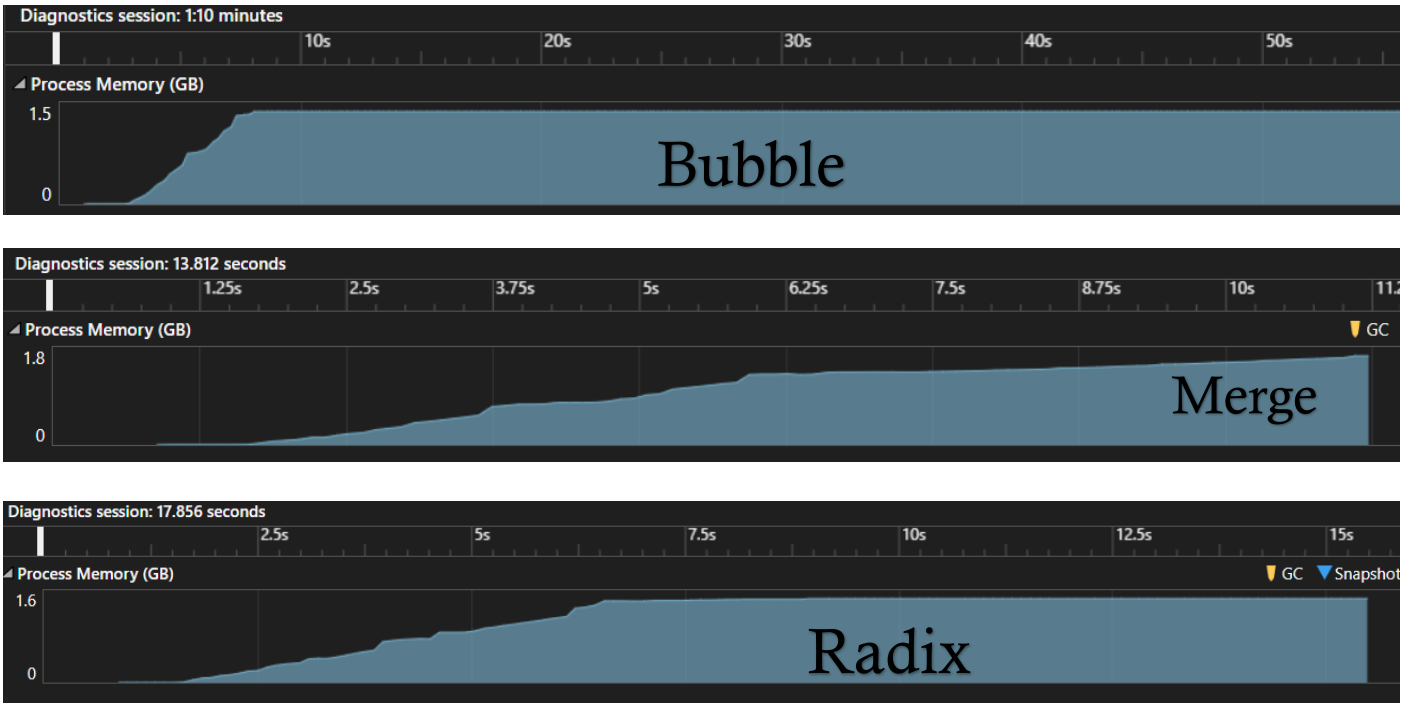
```
[PROBLEM2 MERGE] Novac potreban za sve slatkise: 12
Za PROBLEM2 nizom od 22 elemenata trebalo je 0.471ms.


[PROBLEM2 MERGE] Novac potreban za sve slatkise: 1979
Za PROBLEM2 nizom od 100 elemenata trebalo je 0.0462ms.


[PROBLEM2 MERGE] Novac potreban za sve slatkise: 283
Za PROBLEM2 nizom od 1000 elemenata trebalo je 0.2347ms.


[PROBLEM2 MERGE] Novac potreban za sve slatkise: 15
Za PROBLEM2 nizom od 10000 elemenata trebalo je 6.1971ms.


[PROBLEM2 MERGE] Novac potreban za sve slatkise: 15
Za PROBLEM2 nizom od 100000 elemenata trebalo je 85.3791ms.


[PROBLEM2 MERGE] Novac potreban za sve slatkise: 15
Za PROBLEM2 nizom od 1000000 elemenata trebalo je 493.4493ms.


[PROBLEM2 MERGE] Novac potreban za sve slatkise: 15
Za PROBLEM2 nizom od 10000000 elemenata trebalo je 4222.3242ms.
```

Korišćenjem MERGE sorta za potrebe rešavanja ovog problema uočavamo slično ponašanje. Primećujemo da je zaista ispraćeno pravilo rasta složenosti, uz opet činjenicu da su za nizove manjih dužine kalkulacije najniže cene uticale na ukupno vreme izvršenja nego na nizove većih dužina.

I najzad korišćenjem radix sort algoritma za potrebe rešavanja problema prodavnice slatkiša primećujemo isto pravilo. Složenost raste linearno s tim da odredjivanje same potrebne ukupne cene svih slatkiša koje moramo kupiti da bismo uz onih k besplatnih svaki put probali svih N slatkiša utiče na ukupno vreme izvršenja više kod malih nizova nego kod onih većih.

```
[PROBLEM2 RADIX] Novac potreban za sve slatkise: 12
Za PROBLEM2 nizom od 22 elemenata trebalo je 49.5589ms.


[PROBLEM2 RADIX] Novac potreban za sve slatkise: 1979
Za PROBLEM2 nizom od 100 elemenata trebalo je 0.146ms.


[PROBLEM2 RADIX] Novac potreban za sve slatkise: 283
Za PROBLEM2 nizom od 1000 elemenata trebalo je 0.8166ms.


[PROBLEM2 RADIX] Novac potreban za sve slatkise: 10
Za PROBLEM2 nizom od 10000 elemenata trebalo je 10.3151ms.


[PROBLEM2 RADIX] Novac potreban za sve slatkise: 100010
Za PROBLEM2 nizom od 100000 elemenata trebalo je 124.5045ms.


[PROBLEM2 RADIX] Novac potreban za sve slatkise: 10
Za PROBLEM2 nizom od 1000000 elemenata trebalo je 1016.7703ms.


[PROBLEM2 RADIX] Novac potreban za sve slatkise: 10
Za PROBLEM2 nizom od 10000000 elemenata trebalo je 8539.2416ms.
```

MEMORIJA U PROBLEMU SLATKIŠA



Kao i malopre, uočavamo povećanje iskorišćenosti memorije kod radix i merge sorta u odnosu na bubble sort. Razlog je isti kao malopre naveden. Ovde u problemu slatkiša nemamo nikakvo znatno zauzeće memorije pored sort algoritma, osim par integer promenljivih koji nam čuvaju podatke za while petlju i ukupnu sumu novca.

ZAKLJUČAK

Zaključak je da zaista ovi testirani algoritmi prate složenosti koje smo odredili na časovima. Neki su bolji u pogledu vremena, dok su istovremeno gori u pogledu memorijskog zauzeća. To je čest problem u određivanju algoritma koji je potrebno koristiti u našem konkretnom slučaju. Nekad treba praviti balans između memorije (resursa) i vremena, a nekad kompromis.

Ristovski Nikola 19347, Elektronski fakultet Niš, 2024