SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKT IZ PREDMETA BIOINFORMATIKA

Najdulji rastući podniz O(NlogN) - implemetacija

Vedrana Briševac

Ankica Gogić

Filip Grilec

Nikolina Očić

Voditelji:

Dr. sc. Mirjana Domazet - Lošo

Doc. dr. sc. Mile Šikić

Zagreb, siječanj, 2014.

Sadržaj

1. Teoretski opis problema

Problem najduljeg rastućeg podniza jest za zadani niz pronaći najdulji mogući podniz u kojem će svi elementi biti sortirani u rastućem poretku (od najmanjeg ka najvećemu). Navedeni podniz najčešće se označava sa LIS i nije nužno jedinstven. Primjerice, za niz A = {2, 5, 3} postoje dva jednako dobra LIS-a, LIS1 = {2, 5} i LIS2 = {2,3}.

Postoji nekoliko poznatih rješenja problema:

- rekurzivnim izvedbama postigla se složenost $O(2^N)$ na način da se niz rekurzivno pretražuje u potrazi za najdužim nizom.
- rješenje izvedeno dinamičkim programiranjem složenosti je $O(N^2)$, a ideja mu je da traži najdulji zajednički podniz nizova S i T, gdje je S zadani niz, a T sortirani niz S. Poznata je još i činjenica da dotično rješenje u specijalnom slučaju kada je niz permutacija brojeva 1, … , n postiže nešto bolje rezultate i složenosti je O(N log log N).
- trenutno najučinkovitija verzija algoritma jest ona složenosti O(NlogN) koja će biti promatrana i izvedena u ovom projektu.

Najdulji rastući podniz LIS niza S koristi se kod poravnavanja dvaju nizova, primjerice u metodi MUMer[1] koja se zasniva na globalnom poravnanju sljedova. Njegova primjena nije samo u bioinformatici, te se tako koristi u takozvanim *diff*[2]algoritmima, točnije u njegovoj verziji *PatienceDiff*[3] gdje se uspoređuju sličnosti dviju datoteka.

[1] MUM = **M**aximal**U**nique**M**atch, Mirjana Domazet – Lošo, Poravnavanje više slijedova odjednom, FER-ZPR, 6. predavanje, Bioinformatika, http://www.fer.unizg.hr/_download/repository/Bioinformatika_-_6._predavanje.pdf

[2] http://en.wikipedia.org/wiki/Diff

[3] http://git.661346.n2.nabble.com/Bram-Cohen-speaks-up-about-patience-diff-td2277041.html

2. Teoretski opis algoritma O(NlogN)

Algoritam složenosti O(NlogN) zasniva se na binarnom pretraživanju složenosti O(logN) te korištenju nekoliko pomoćnih nizova. Po ulaznom nizu prolazi se samo jedanput, što daje složenost O(N). Kako se za svakog člana niza radi binarno pretraživanje, dobivamo složenost O(NlogN).

Ideja je da se procesira jedan po jedan član ulaznog niza X, te se nakon procesiranja člana X[i] spreme potrebne vrijednosti u pomoćne nizove M[j] i P[k]. Pritom nizovi M i P imaju slijedeće značenje:

- M[j] – sprema poziciju *p* najmanje vrijednosti *X[p]* (najmanje vrijednosti ulaznog niza) takve da postoji rastući podniz duljine *j* koji završava na poziciji *X[p]*, tako da je j ≤ p≤ i.
- P[p] – sprema poziciju prethodnika od *X[p]* u najduljem rastućem podnizu koji završava u *X[p]*

Dodatno se sprema i varijabla L koja označava duljinu najdužeg dotad pronađenog podniza.

Laički rečeno, prolazimo po ulaznom nizu X, gledajući jedan po jedan znak. Pomoću binarnog pretraživanja gledamo u kakvom je odnosu znak sa krajnjim elementima svih prethodno zapamćenih listi.

On može biti takav da je veći od svih krajnjih znakova, te ćemo ga tada staviti na kraj najduljeg niza, prethodno ga duplicirajući kako bismo ostavili prostora za moguće bolje rješenje iste duljine. Tim postupkom dobivamo podniz duljine *j*, te u pomoćni niz M na poziciju *j* zapisujemo poziciju *p* koju zadnji član ovog podniza(novododani ulazni znak) ima u ulaznom nizu X. Također, u niz P na poziciju *p* zapisujemo poziciju *p'* koja odgovara poziciji predzadnjeg člana podniza u koji smo smjestili trenutni znak (pamtimo njegovog prethodnika u novoizgrađenom podnizu).

Ako je pak znak takav da je manji od svih krajnjih elemenata svih poznatih podnizova, binarnim sortiranjem pronaći ćemo takav podniz koji će za krajnji element imati najveću moguću vrijednost još uvijek manju od vrijednosti trenutnog znaka. Taj niz također ćemo duplicirati i dodati mu na kraj ulazni znak. No, ovoga puta još odbacujemo sve postojeće podnizove čije su duljine jednake duljini novoizgrađenog podniza, čime omogućavamo daljnju nadogradnju. To ostvarujemo tako da u pomoćnom nizu M ovaj puta na poziciji *j* prepišemo postojeću vrijednost pozicije sa pozicijom *p* ulaznog znaka. U pomoćni niz P još uvijek spremamo vrijednost prethodnika ulaznog znaka.

Algoritam počiva na činjenici da ako postoji rastući podniz duljine *i* koji završava na *X[M[i]]*, onda također postoji podniz duljine *i-1* koji završava na manjoj vrijednosti nego onaj duljine *i* – točno onaj koji završava na *X[P[M[i]]]*. Prema tome, moguće je provoditi binarna pretraživanja u logaritamskom vremenu.

Pseudokod algoritma je slijedeći:

```
L=0
for i = 1, 2, ..., n:
        binarno potraži najmanji pozitivni j ≤ L takav da je X[M[j]] < X[i] ili j=0 ako takva
        vrijednost ne postoji
        P[i] = M[j]
        if j == L or X[i] < X[M[j+1]] :
                M[j+1] = i
                L = max (L, j+1)
```

Konačna duljina najduljeg podniza spremljena je u varijabli L. Sam podniz iz pomoćnih nizova i ulaznog niza iščitava se počevši od zadnjeg člana. Njega se iščita kao X[M[L]], slijedeći je X[P[M[L]]], zatim X[P[P[M[L]]]], itd.

Iščitavanje je dano slijedećim pseudokodom:

```
p = M[L]
for i = L, ..., 0:
        LIS[i] = X[p]
        p = P[p]
```

Kao što je već navedeno, za pretragu niza koristi se binarno pretraživanje. Pritom se misli na klasičnu iterativnu primjenu pretraživanja danu slijedećim pseudokodom:

```
while (imin<imax):
        intimid = midpoint (imin, imax)
        if A[imid] <key:
                imin = imid + 1
        else:
                imax = imid
```

3. Objašnjenje algoritma na primjeru

Kao što je to slučaj s većinom algoritama, tako je i ovaj mnogo razumljivije shvatiti na primjeru.

Uzmimo stoga slijedeći niz: {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15}[4]

Algoritam provodimo redom nad svim članovima niza na slijedeći način:

Korak 1 Inicijaliziramo sve nizove i varijable:

i	j	M	P	L
0	-	{0}	{-1}	0

U ovom slučaju binarnim pretraživanjem ustvari uspoređujemo 0 s 0 te kao rezultat dobijemo j = -1. Kako je j != L i uspoređujemo 0 s 0, ostali nizovi se ne mijenjaju. Ovo je slučaj inicijalizacije kada pamtimo prvi niz, duljine 1 koji nema prethodnika (P={-1}). Važno je uočiti da se u ovoj izvedbi u pomoćni niz M upisuju duljine krenuvši od 0, te ih se uvijek treba interpretirati kao da su za 1 veće.

i	j	M	P	L
0	-1	{0}	{-1}	0

Korak 2 Promatramo broj 8. Binarnim sortiranjem saznajemo da je on veći od svih krajnjih elemenata postojećih podnizova, te kloniramo najdulji podniz {0} i na kraj mu dodajemo ulazni znak. Tako sada imamo podnizove {0} i {0,8}. U niz M na poziciji 1 koja predstavlja podniz duljine 2 zapisujemo poziciju ulaznog znaka, a u niz P dodajemo poziciju prethodnika ulaznog znaka. Status je na kraju ovog koraka slijedeći:

i	j	M	P	L
1	0	{0,1}	{-1,0}	1

Korak 3 Promatramo broj 4. Binarnim sortiranjem saznajemo da on nije veći od svih krajnjih znakova postojećih podnizova. Također dobivamo informaciju da je veći od niza kojemu je krajnji znak {0}, te taj niz dupliciramo i na kraj mu dodajemo znak 4. Dobivamo niz {0,4} koji je iste duljine kao i niz {0,8}, ali je njegov krajnji element manji od krajnjeg elementa niza {0,8}, te stoga odbacujemo niz {0,8}. To radimo na način da u pomoćni niz M na poziciju 1 koja predstavlja niz duljine 2 zapisujemo poziciju novopristiglog znaka koja iznosi 2. Sada imamo podnizove {0} i {0,4}. U niz P još dodajemo poziciju njegovog prethodnika (znaka 0), te je situacija slijedeća:

[4]Preuzeto sa http://en.wikipedia.org/wiki/Longest_increasing_subsequence

i	j	M	P	L
2	0	{0,2}	{-1,0,0}	1

Korak 4 Promatramo broj 12. Binarnim sortiranjem saznajemo da je taj ulazni znak veći od svih krajnjih znakova postojećih listi, pa odabiremo najdulju listu, dupliciramo ju i dodamo joj 12 na kraj. Ovime smo našu listu nizova proširili za novi niz {0.4.12}, te sada imamo tri mogućnosti {0}, {0,4} i {0,4,12}.Pomoćni niz M proširujemo dodajući mu na poziciju 2 koja označava niz duljine 3 indeks pozicije novododanog znaka 12. U niz P dodajemo indeks prethodnika dodanog znaka, a varijablu duljine L povećavamo za 1 (i nju je potrebno promatrati tek nakon što ju se uveća za 1 jer je i njeno indeksiranje krenulo od 0). Situacija je stoga slijedeća:

i	j	M	P	L
3	1	{0,2,3}	{-1,0,0,2}	2

Ako promotrimo prethodna 4 koraka, evidentno je da postoje 3 situacije:

1. započinjemo novu listu duljine 1
2. X[i] je najveći od svih krajnjih elemenata svih listi. Odabiremo najdulju aktivnu listu, kloniramo ju i dodamo X[i] na kraj (korak 4)
3. X[i] je manji od svih krajnjih elemenata svih listi, ali veći od njihovih početaka. Pronalazimo listu s najvećim krajnjim elementom koji je još uvijek manji od X[i], kloniramo tu listu i proširimo ju s X[i] te odbacimo sve liste iste dužine kao novo dobivena (korak 3).

Stoga ćemo ostatak algoritma provesti pozivajući se na jednu od 3 situacije.

Korak 5 Promatramo broj 2.
Trenutni nizovi su: {0}
{0,4}
{0,4,12}
U situaciji smo (3), te odabiremo niz {0}, kloniramo ga i dodamo 2 na kraj, dobivši {0,2}. Nakon što još odbacimo niz {0,4}, dobivamo slijedeće nizove : {0}, {0,2} i {0,4,12}, te

i	j	M	P	L
4	0	{0,4,3}	{-1,0,0,2,0}	2

Korak 6 Promatramo broj 10. Opet smo u situaciji (3), odabiremo niz {0,2}, kloniramo ga i dodamo 10 na kraj, dobivši {0,2,10}. Odbacujemo niz {0,4,12} i dobivamo nizove: {0}, {0,2}, {0,2,10} te

i	j	M	P	L
5	1	{0,4,6}	{-1,0,0,2,0,4}	2

Korak 7 Promatramo broj 6. Ponovno situacija (3), odabiremo niz {0,2}, kloniramo ga i dodamo 6 na kraj, dobivši {0,2,6}. Odbacujemo niz {0,2,10} i dobivamo nizove {0}, {0,2} i {0,2,6} te

i	J	M	P	L
6	1	{0,4,6}	{-1,0,0,2,0,4,4}	2

Korak 8 Promatramo broj 14. Ovaj put smo u situaciji (2), jer je 4 najveći od svih krajnjih elemenata. Stoga odabiremo najdulji niz {0,2,6} i dodajemo 14 na kraj, te dobijemo {0,2,6,14}. Imamo nizove {0}, {0,2}, {0,2,6} i {0,2,6,14} te

i	J	M	P	L
7	2	{0,4,6,7}	{-1,0,0,2,0,4,4,0}	3

Korak 9 Promatramo broj 1. U situaciji smo (3), kloniramo niz {0} i dodajemo 1 na kraj. Odbacujemo niz {0,2} i imamo nizove {0}, {0,1}, {0,2,6,14} te

i	J	M	P	L
8	0	{0,8,6,7}	{-1,0,0,2,0,4,4,0,0}	3

Korak 10 Promatramo broj 9. U situaciji smo (3), kloniramo niz {0,2,6} i dodamo 9 na kraj, te odbacimo niz {0,2,6}. Imamo {0}, {0,1}, {0,2,6}, {0,2,6,9} te

i	J	M	P	L
9	2	{0,8,6,9}	{-1,0,0,2,0,4,4,0,0,6}	3

Korak 11 Promatramo broj 5. U situaciji smo (3), kloniramo niz {0,1} i dodamo 5 te dobijemo {0,1,5}. Odbacujemo niz {0,2,6} i imamo nizove {0}, {0,1}, {0,1,5}, {0,2,6,9} te

i	J	M	P	L
10	1	{0,8,10,9}	{-1,0,0,2,0,4,4,0,0,6,8}	3

Korak 12 Promatramo broj 13. U situaciji smo (2), odabiremo najdulji niz, kloniramo ga i dodamo 13 na kraj. Na kraju imamo nizove {0}, {0,1}, {0,1,5}, {0,2,6,9}, {0,2,6,9,13} te

i	J	M	P	L
11	3	{0,8,10,9,11}	{-1,0,0,2,0,4,4,0,0,6,8,9}	4

Korak 13 Promatramo broj 3. U situaciji smo (3), kloniramo niz {0,1}, dodamo 3 na kraj te odbacimo niz {0,1,5}. Na kraju imamo nizove {0}, {0,1}, {0,1,3}, {0,2,6,9} i {0,2,6,9,13} te

i	J	M	P	L
12	1	{0,8,12,9,11}	{-1,0,0,2,0,4,4,0,0,6,8,9,8}	4

Korak 14 Promatramo broj 11. U situaciji smo (3), kloniramo niz {0,2,6,9}, dodajemo 11 na kraj te odbacujemo niz {0,2,6,9,13}. Na kraju imamo nizove {0}, {0,1}, {0,1,3}, {0,2,6,9} i {0,2,6,9,11} te

i	J	M	P	L
13	3	{0,8,12,9,13}	{-1,0,0,2,0,4,4,0,0,6,8,9,8,9}	4

Korak 15 Promatramo broj 7. U situaciji smo (3), kloniramo niz {0,1,3}, dodamo 7 na kraj i imamo {0,1,3,7}. Odbacujemo niz {0,2,6,9} i imamo nizove {0}, {0,1}, {0,1,3}, {0,1,3,7} i {0,2,6,9,11} te

i	J	M	P	L
14	2	{0,8,12,14,11}	{-1,0,0,2,0,4,4,0,0,6,8,9,8,9,12}	4

Korak 16 Promatramo broj 15. U situaciji smo (2). Uzimamo najdulji niz i dodajemo mu 15. Na kraju imamo nizove {0}, {0,1}, {0,1,3}, {0,1,3,7}, {0,2,6,9,11} i {0,2,6,9,11,15} te

i	J	M	P	L
15	4	{0,8,12,9,11,15}	{-1,0,0,2,0,4,4,0,0,6,8,9,8,9,12,13}	5

U konačnici odabiremo najdulji niz, što je u ovom slučaju niz {0,2,6,9,11,15}.

4. Programske izvedbe

U sklopu ovog projekta problem pronalaska najduljeg rastućeg podniza pri složenosti O(NlogN) izveden je u nekoliko programskih jezika. Jezici su Java (2 verzije), Python i C.

Radi testiranja svih verzija izgeneriran je niz slučajnih brojeva u rasponu od 1 do 10000 duljine 1000 znakova. Navedeni niz moguće je pronaći u dodatku A zajedno sa najduljim rastućim podnizom koji je moguće pronaći u njemu.

Testiranje je izvedeno na Biolinux[5] platformi. Platforma je pokrenuta kao virtualna mašina pomoću programa VMware Player[6] na 64-bitnoj verziji Windowsa 7 Professional, SP1. Hardverska potpora bazira se na uređaju sa 4GB RAM-a te 2.80 GHz Pentium Dual-Core procesoru. Samoj mašini dodijeljeno je 1GB RAM-a za izvođenje.

Važna napomena: radi omogućavanja jednostavnijeg izvođenja krajnjem korisniku, čitanje ulaznih podataka ostvareno je izravnim pozivom ulazne datoteke unutar koda. Korisnik je onda, ukoliko želi promijeniti ulazne podatke, dužan iste izmijeniti otvaranjem datoteke *u.txt* koja se nalazi u svakom od datoteka projekata i izmjenom podataka u njoj.

U narednim poglavljima biti će ukratko analizirano memorijsko zauzeće kao i vremensko izvođenje svakog pojedinog koda. Kodove je moguće pronaći u dodatku B.

4.1 C izvedba (Filip Grilec)

Time [s]: 0.002
Memory [B]:4407296

4.2 Python izvedba

4.2.1 Verzija 1 (Vedrana Briševac)

Time [s]: 0.001
Memory [B]:3324592

[5] http://nebc.nerc.ac.uk/tools/bio-linux
[6] https://my.vmware.com/web/vmware/downloads

4.2.2 Verzija 2 (Nikolina Očić)

Time [s]: 0.001 Memory [B]:3321552

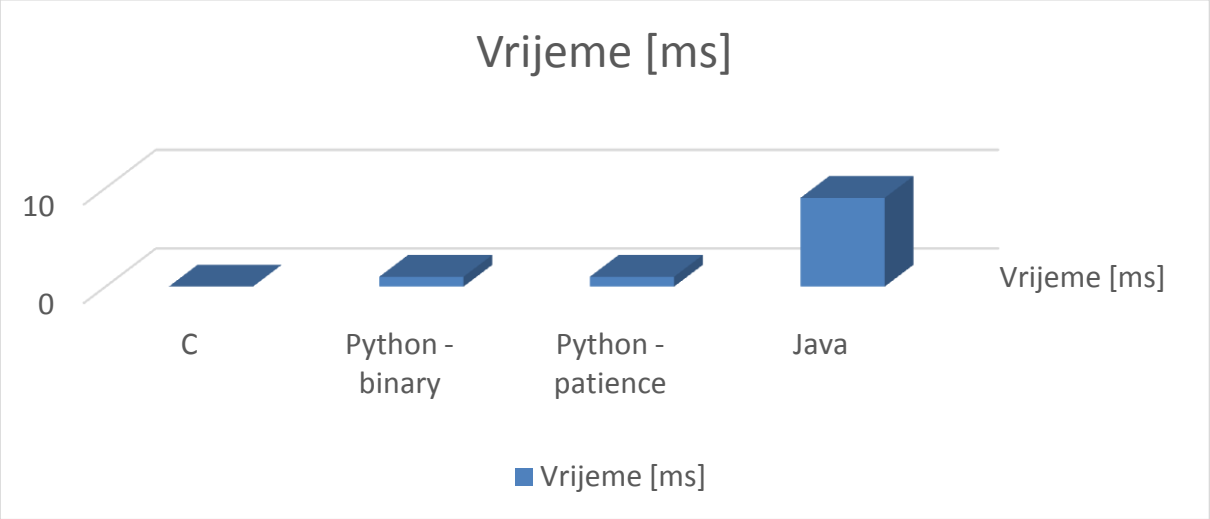
4.3 Java izvedba (Ankica Gogić)

Time [s]: 0.008 Memory [B]:333824

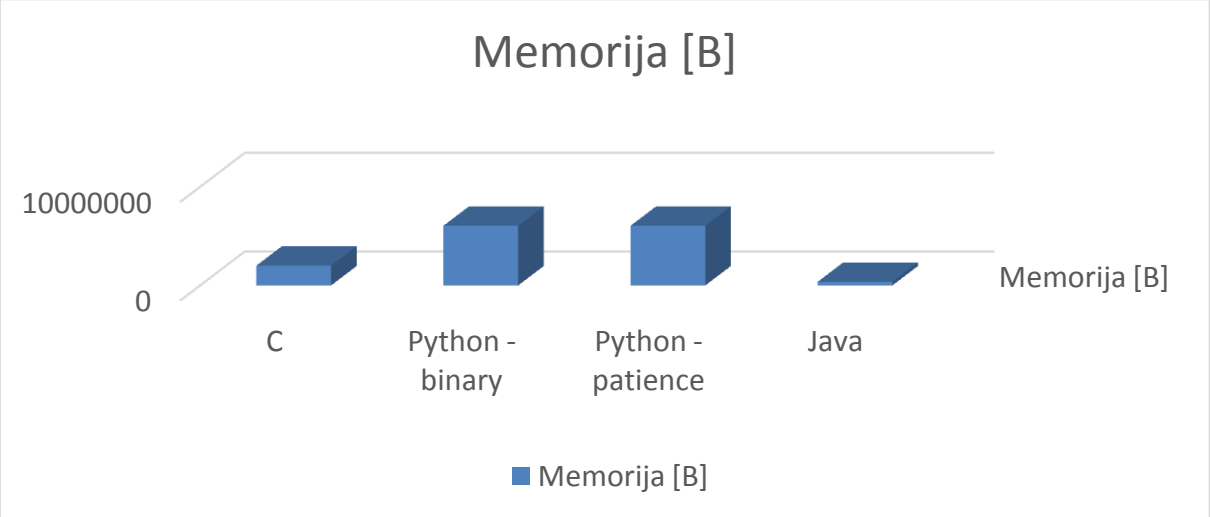
5. Usporedba izvedbi

Izvođenjem programa je ustanovljeno vrijeme izvođenja i memorijsko zauzeće svakog programa. Rezultati su prikazani na slici 1 i 2.

Kao zaključak Java koristi najmanje memorije, ali je također i najsporija izvedba. Brzine izvedaba u Pythonu i C-u zauzimaju puno više memorije, ali su brže sa relativno jednakim brzinama izvedbe.



Slika 1: Vremensko trajanje programa



Slika 2: Memorijsko zauzeće programa

Dodatak A – ulazni i izlazni nizovi

Ulazni niz:

1618, 5140, 8776, 5285, 5164, 4470, 4433, 5070, 2728, 4100, 3395, 3337, 8215, 786, 3473, 8299, 390, 2366, 2282, 6488, 4011, 5017, 8502, 8742, 5756, 1502, 9077, 7918, 3036, 3513, 6954, 7356, 4696, 1548, 6876, 2747, 60, 5196, 1300, 7763, 6828, 5208, 4508, 5574, 8917, 6239, 661, 4226, 2497, 5803, 717, 3702, 2198, 631, 3860, 6506, 9754, 6358, 1456, 1561, 4167, 275, 5372, 3350, 5923, 9734, 7972, 4330, 7731, 730, 8098, 8876, 803, 2152, 5887, 1273, 8783, 1258, 9431, 8607, 1102, 5033, 7970, 1788, 3737, 281, 9532, 4417, 8155, 2494, 5795, 9688, 8566, 9904, 3285, 145, 3495, 4418, 7977, 3708, 186, 5619, 9870, 2126, 9366, 7458, 5180, 4757, 3485, 9835, 6387, 1416, 3730, 8874, 4608, 2925, 9818, 361, 4493, 6951, 1840, 570, 9147, 9735, 5311, 7475, 1490, 1401, 8961, 3230, 8878, 3667, 9767, 5715, 9809, 3992, 1300, 2920, 5145, 9459, 9504, 9018, 1331, 7027, 4717, 5050, 9873, 4174, 9958, 353, 5503, 5394, 3283, 3080, 5337, 5573, 6351, 9463, 2680, 789, 2271, 1839, 2068, 770, 3254, 4864, 6654, 4936, 2627, 7857, 5103, 5698, 6372, 8725, 2790, 8554, 6512, 1578, 1266, 8385, 2358, 5767, 750, 5316, 5974, 8853, 2674, 1545, 4983, 9808, 3216, 2578, 9324, 1576, 4176, 2508, 8483, 4098, 4577, 3128, 2283, 9860, 4900, 972, 1405, 4058, 6025, 4700, 3476, 638, 6624, 7605, 6766, 3576, 5993, 6538, 6342, 58, 4151, 8995, 6848, 8479, 7390, 4741, 2018, 9563, 5394, 6962, 3454, 3321, 5582, 658, 295, 8048, 1353, 2733, 6939, 120, 3555, 9100, 8182, 4953, 8702, 8989, 9837, 162, 3201, 5605, 8177, 1700, 152, 5304, 3413, 2537, 4198, 4539, 798, 1343, 309, 4961, 7944, 5501, 4394, 2267, 8452, 8817, 7213, 3671, 66, 9324, 8168, 2170, 210, 6663, 2254, 3205, 2996, 1752, 1702, 756, 9560, 3628, 3656, 5422, 7956, 9705, 6605, 6628, 9705, 6002, 1234, 4737, 7616, 3355, 5850, 6561, 3479, 4093, 2615, 9504, 4873, 1213, 8446, 5451, 7449, 8707, 3609, 5341, 2797, 7354, 1818, 9393, 2280, 5034, 7943, 5901, 4055, 4425, 3312, 3645, 3925, 3798, 4038, 9090, 631, 8547, 4795, 9732, 2710, 2083, 9035, 9253, 7175, 4773, 6777, 8453, 9893, 8696, 2755, 8319, 3788, 4205, 1014, 8056, 3761, 1455, 6799, 1428, 6573, 6427, 3023, 9312, 9838, 6288, 6473, 5039, 6762, 3368, 8473, 1782, 2266, 531, 5746, 8030, 1754, 3245, 9874, 8006, 5944, 5291, 7919, 8797, 3869, 5702, 2536, 4888, 1486, 4275, 7222, 3860, 5144, 1794, 1216, 4024, 7565, 8956, 9008, 5983, 9003, 4933, 1420, 5241, 7723, 2463, 8770, 2318, 2096, 1442, 1795, 99, 1208, 7079, 2645, 5580, 2154, 1846, 6558, 4612, 9921, 2729, 4624, 3755, 162, 6523, 9406, 2952, 6021, 6338, 8158, 6361, 7679, 1864, 2856, 6390, 7291, 4444, 2337, 7760, 2621, 5564, 9457, 4229, 9794, 3951, 5373, 530, 9333, 2181, 3820, 5995, 9935, 8402, 541, 4407, 1262, 2593, 6917, 6633, 460, 9651, 2310, 8808, 7060, 3485, 6578, 8979, 7483, 7331, 2838, 9434, 3428, 5509, 654, 249, 8112, 981, 1523, 1969, 535, 1233, 1774, 1825, 731, 2231, 2831, 8378, 8314, 6527, 5146, 4069, 2523, 6917, 62, 3814, 4472, 1509, 3353, 448, 1295, 9520, 5278, 4644, 7526, 9690, 4580, 7709, 1175, 4799, 8075, 594, 685, 6447, 3224, 1378, 4264, 4011, 8387, 6531, 4900, 457, 4333, 8322, 2793, 5594, 1042, 1881, 1907, 4226, 5809, 651, 8872, 2702, 2237, 8197, 571, 2351, 4838, 7705, 1394, 2470, 642, 7879, 2369, 8633, 7471, 5388, 2166, 4391, 8337, 9566, 4309, 5690, 2827, 4607, 5836, 7698, 9249, 8184, 134, 3196, 5830, 3073, 690, 112, 5317, 190, 3104, 3377, 4963, 7224, 9880, 2870, 3061, 6987, 6193, 4896, 3954, 3522, 5171, 9383, 5750, 2047, 8018, 8769, 8853, 893, 7756, 7435, 8873, 9725, 5314, 2435, 4445, 8553, 3222, 7285, 9105, 6292, 5285, 612, 6441, 5987, 241, 565,

9464, 6176, 3648, 4085, 8393, 4345, 5039, 7356, 5842, 8943, 6680, 43, 4784, 994, 6240, 1326, 2633, 4997, 8105, 106, 4226, 1502, 7119, 5304, 7624, 527, 4132, 4673, 9220, 5801, 4880, 2186, 8861, 4710, 2467, 4727, 1432, 7796, 8978, 8544, 8680, 4656, 3610, 4726, 3903, 167, 3108, 831, 2603, 6134, 5279, 2709, 4174, 8627, 7880, 6553, 3286, 559, 3255, 6263, 4736, 2397, 3797, 5307, 2222, 1262, 4487, 5467, 9780, 155, 6570, 7046, 4389, 4555, 773, 6463, 9674, 3207, 9363, 7481, 3544, 3579, 3832, 1770, 7117, 150, 9023, 2971, 8353, 5472, 5092, 5986, 1422, 9783, 5885, 9821, 2056, 4831, 6013, 8007, 9855, 1516, 9192, 5328, 853, 9010, 7267, 3396, 7506, 5043, 2738, 500, 8731, 3235, 7594, 3029, 9602, 3529, 7767, 6802, 8887, 6055, 5650, 6663, 2733, 7084, 2730, 1817, 4489, 8139, 678, 2550, 3929, 5756, 1944, 1223, 3927, 9926, 313, 6728, 9027, 3833, 7747, 6938, 8287, 6124, 6339, 8132, 4281, 9679, 3198, 3845, 1207, 7585, 1734, 8145, 7658, 7889, 4909, 9959, 3846, 3550, 4804, 5732, 8764, 3007, 8069, 7437, 135, 9775, 8737, 1786, 8845, 6561, 8231, 4291, 10, 550, 7401, 8819, 4815, 1690, 6947, 9211, 844, 5113, 690, 4626, 3743, 2101, 791, 4361, 9734, 1469, 5261, 1067, 6761, 4625, 5877, 3549, 1877, 3127, 8962, 1587, 3819, 7132, 2409, 5634, 2707, 6750, 9781, 3613, 3255, 6054, 6361, 5571, 3996, 1240, 2637, 4462, 2603, 276, 7832, 7962, 6613, 1258, 9835, 1887, 9599, 664, 8173, 510, 5137, 690, 3489, 2902, 4826, 6968, 426, 7243, 2018, 4059, 760, 7011, 9944, 6377, 180, 3910, 1484, 6408, 5207, 2479, 7942, 474, 6378, 409, 5888, 4639, 82, 6775, 4379, 9842, 4175, 7649, 9686, 5835, 1887, 4818, 1028, 8101, 4403, 8726, 8751, 565, 5011, 1172, 5901, 9609, 6710, 6777, 5204, 1680, 776, 4795, 6072, 7601, 5961, 4426, 479, 1603, 6693, 7512, 2266, 2016, 3944, 2176, 1787, 3634, 4276, 9809, 6641, 8774, 982, 5647, 9711, 5266, 6210, 106, 7723, 4757, 4495, 4040, 7087, 4020, 6205, 6127, 8930, 5259, 4098, 4700, 1906, 8820, 7354, 9513, 9633, 3152, 5301, 1178, 9400, 2393, 7317, 7293, 200, 1532, 6380, 4656, 1029, 2729, 1145, 6367, 4968, 4110, 5491, 1133, 549, 4083, 5076, 9612, 9592, 87, 7592, 9250, 9870, 547, 7111, 4765, 1699, 2791, 3257, 6126, 9888, 7685, 2696, 9701, 6328, 3821, 3936, 1111, 3994, 252, 6331, 237, 56, 6958, 742, 9814, 4449, 6493, 2427, 9115, 295, 8609, 6328, 3307, 4328, 2034, 1226, 7270, 8680, 4028, 1933, 9876, 219, 8881, 8110, 1038, 7, 1080, 4158, 9211, 3379, 7517, 7118, 1420, 655, 5815

LIS:

60, 661, 717, 730, 803, 1102, 1300, 1331, 1545, 1576, 1700, 1702, 1754, 1794, 1795, 1846, 1864, 1881, 1907, 2237, 2351, 2369, 2827, 3073, 3104, 3377, 3522, 3648, 4085, 4132, 4673, 4710, 4726, 4736, 5307, 5467, 5472, 5885, 6013, 6055, 6124, 6339, 6361, 6377, 6378, 6775, 7649, 8101, 8726, 8751, 8774, 8820, 9400, 9592, 9701, 9814, 9876

Dodatak B – Programski kodovi

C izvedba (Filip Grilec)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>


int binary_rep(int *arr, int left, int right, int key) {
        //Standard binary search and insertion of number into the
result array
        int mid=(left+right)/2;
        for (mid; left <= right; mid = (left+right)/2)
        {
                if (arr[mid] > key)
                        right = mid - 1;
                else if (arr[mid] == key)
                        return mid;
                else if (mid+1 <= right && arr[mid+1] >= key)
                    {
                        arr[mid+1] = key;
                        return mid+1;
                    }
                else
                        left = mid + 1;
        }
        if (mid == left)
        {
            arr[mid] = key;
            return mid;
        }
        arr[mid+1] = key;
        return mid+1;
}

int parseLine(char* line){
        int i = strlen(line);
        while (*line < '0' || *line > '9') line++;
        line[i-3] = '\0';
        i = atoi(line);
        return i;
    }


int getValue(){
      //Note: this value is in KB!
      //works only in Linux
        FILE* file = fopen("/proc/self/status", "r");
        int result = -1;
        char line[128];
```

```
        while (fgets(line, 128, file) != NULL){
            if (strncmp(line, "VmSize:", 7) == 0){
                result = parseLine(line);
                break;
            }
        }
        fclose(file);
        return result;
    }

int main(void) {
        //Variables for time measurement
        clock_t begin, end;
        begin = clock();
        double time_spent;

        int i=0, tmp,length = -1;
        //Allocation of memory for the array(first member)
        int *arr_0 =(int*) malloc(1*sizeof(int));

        //Reading input file
        FILE *fp = fopen("u.txt","r");
        if(fp==NULL)
        {
            printf("Error!");
            return 0;
        }
        while(fscanf(fp,"%d, ",&tmp)>0)
         {
             arr_0[i]=tmp;
             i++;
             //Reallocation for every new member added, uknown input
size
             arr_0=(int*)realloc(arr_0,(i+1)*sizeof(int));
         }
        fclose(fp);
        // after reading the input is done, i represents number of
members in the array
        // memory allocation follows accordingly
        int size=i;
        int *arr_n = (int*) malloc(i*sizeof(int));    //lista brojeva
        int *index = (int*) malloc(i*sizeof(int));  //lista indeksa


        for (i = 0; i < size; i++)
        {
                //for every member, do a binary search and replacement
                index[i] = binary_rep(arr_n, 0, i, arr_0[i]);
                if (length < index[i])
                {
                    length = index[i];
                }
        }
        //reading output array backwards to recieve the longest
increasing substring
        int *res = (int*) malloc((length+1) * sizeof(int));
```

```
        for (i = size-1, tmp = length; i >= 0; i--)
        {
                if (index[i] == tmp)
                {
                     res[tmp] = arr_0[i];
                     tmp--;
                }
        }
        //writing in output file

        fp = fopen("output.txt","w");
        //if it fails
        if(fp==NULL)
        {
            printf("Error!");
            return 0;
        }
        //output formatting
        fprintf(fp,"[");
        for (i = 0; i < length; ++i) {
                fprintf(fp,"%d, ", res[i]);
        }
        fprintf(fp,"%d]",res[length]);
        fclose(fp);

        //Calculate required time
        end = clock();
        time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
        printf("Time [s]: %f\n",time_spent);
      //Memory allocation calculation in function
        printf("Memory [B]: %d\n",getValue()*1024);

        return 0;
}
```

Python izvedba (Vedrana Briševac)

```
# Program implements patience sorting algorithm for finding longest
increasing subsequence
# Algorithm taken from http://en.wikipedia.org/wiki/Patience_sorting

import bisect
import time

def patienceSorting(dataList):
    # patienceSorting(dataList) creates heapCurrent and heaps list
    # heapCurrent is the number of heap on which the current integer
from the dataList is being stored
    # every integer is being placed on the top of one of the heaps,
like in solitaire card game
    heaps = list()
    for x in dataList:
        heapCurrent = bisect.bisect_left(heaps, x)
```

```
        if heapCurrent == len(heaps):
            heaps.append(x)
        else:
            heaps[heapCurrent] = x
        yield x, heapCurrent

def findLongestIncreasingSubsequence(dataList):
    heaps = [[]]
    # patience sorting of dataList and storing entrance data on heaps
    # indexing elements on the top of preceding heap
    for x, y in patienceSorting(dataList):
        if y + 1 == len(heaps):
            heaps.append([])
        heaps[y + 1].append((x, len(heaps[y]) - 1))
    # following sequence from the top card in the last pile (in program
heap) towards the first pile
    # its reverse is an answer to the longest increasing subsequence
algorithm.
    lis = list()
    pred = 0
    for heap in range((len(heaps) - 1), 0, -1):
        x, pred = heaps[heap][pred]
        lis.append(x)
    lis.reverse()
    # longest increasing subsequence is being stored in output file in
same format as the input file
    # integers are stored in .txt file, separated with commas
    # findLongestIncreasingSubsequence(dataList) returns only one
correct answer
    foutput = open ('output.txt','w')
    for i in range(0,len(lis)-1):
        foutput.write('%s,'%lis[i])
    foutput.write('%s'%lis[len(lis)-1])
    foutput.close()
    return 0

if __name__ == '__main__':
    #
    begin = time.clock()
    fInput = open ('u.txt','r')
    dataString = fInput.read()
    fInput.close()
    dataList=[int(i) for i in dataString.split(",")]
    findLongestIncreasingSubsequence(dataList)
    stop = time.clock()
    print ("Time[ms]:%.3f " %((stop-begin)*1000))

    from guppy import hpy
    h = hpy()
    print 'Memory[B]:%d' %h.heap().size
```

Python izvedba (Nikolina Očić)

```
import time

#Function that finds longest increasing subsequence
def longest_increasing_subsequence(input):

            """
            Algorithm is based on simple binary search
            It processes each member from input array one by one
            After processing member input[i], in auxiliary arrays M[]
            and P[] appropriate values are stored:
                  M[j] - stores position(index) p of smallest value
                         from input array such that value input[p] is
                         ending value of increasing subsequence of
                         length j, where j<=p<=i (i is index of a value
                         currently in analysis)
                   P[p] - stores position of predecessor of member
                         input[i] which is located in longest
                         subsequence that ends with value input[i]
            L - length of longest subsequence found at some point of
                algorithm
            """

        n = len(input)
        input = [None] + input
        M = [None]*(n+1)
        P = [None]*(n+1)
        L = 0

        for i in range(1,n+1):
                if L == 0 or input[M[1]] >= input[i]:
                        j = 0
                else:
                        lower = 1
                        higher = L+1
                        while lower < higher - 1:
                                middle = (lower + higher)/2
                                if input[M[middle]] < input[i]:
                                        lower = middle
                                else :
                                        higher = middle
                        j = lower

                P[i] = M[j]
                if j == L or input[i] < input[M[j+1]]:
                        M[j+1] = i
                        L = max(L,j+1)

            """
            Reading found members of longest increasing subsequence
            backwards using arrays M[] and P[]
            """

        output = []
        position = M[L]
```

```
        while L > 0:
                output.append(input[position])
                position = P[position]
                L -= 1

        output.reverse()
        return output

#Main function
if __name__ == '__main__':
        f1 = open ('u.txt','r')
        f2 = open ('izlaz.txt','w')

           """
           Reading input file as a string and transforming it into an
           array of integers for further processing
           """

        str = f1.read()
        array = [int(x) for x in str.split(',')]
        f1.close()

           """
           Start measuring time required for running of algorithm
           Call for algorithm function
           """

        start = time.clock()
        lis = longest_increasing_subsequence(array)
        end = time.clock()

           """
           Writing result into output file
           """

        f2.write('%s'%lis[0])
        for i in range(1,len(lis)):
        f2.write(', %s'%lis[i])
        f2.close()

           """
           Print measured running time
           Estimate and print used memory
           """

        print "Time[s]:%.3f " %(end-start)

        from guppy import hpy
        h = hpy()
        print 'Memory[B]:%d' %h.heap().size
```

Java izvedba verzija (Ankica Gogić)

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class LongestIncreasingSubsequence{

	//This is a standard version of binary search done iteratively

	//The input is array A[] on which the search is made; help array
M[]
	// with indexes of last members of each sub array of length l,
where l
	// is in range of 0 to M.length()-1; variable position which
indicates
	// which member of array A[] to observe and variable length which
	// indicates the length of currently longest found subsequence

	static int binary_search(int A[], int[] M, int position, int
length)
	{
		// variable initialisation, x must be nonnegative number
		int x = 0;
		int y = length;
		int middle;

		// binary search is done within boundaries x and y
		while (x <= y){
			middle = (x+y)/2;
			// comparing all last members of known sub sequences
with
			// current number on which the search is done
			if (A[M[middle]]<A[position]){
				x=middle+1;
			}
			else{
				y = middle-1;
			}
		}
		return y;
	}
	// A function that finds the longest sub sequence of given input
array

	private static int[] findLongestSubsequence(int[] inputArray){

		// Initialising variables
		int L = 0;
		int j;
		int[] M = new int[inputArray.length];
		int[] P = new int[inputArray.length];
		M[0]=0;
```

```
            P[0]=-1;

            // The main loop goes through all the elements of input
array
            for (int i=0; i<inputArray.length; i++){

                  // First the binary search is done to find where to put
                  // the new element
                  j = binary_search(inputArray, M, i, L);

                  // For all situations but the first element inspection
                  // remember the position of the predecessor
                  if ( j !=-1){
                        P[i]=M[j];
                  }

                  // If the conditions are met, remember the new sub
sequence
                  // and if necessary discard all existing sub sequences
of
                  // the same length
                  if (j==L || (inputArray[i]<inputArray[M[j+1]])){
                        M[j+1]=i;

                        // Also remember the new largest length if
necessary
                        if (j+1>L){
                              L = j+1;
                        }
                  }
            }

            // Make new array LIS in which the solution will be
            int[] LIS = new int[L+1];

            // Start reading backwards using help arrays P and M
            int p = M[L];
            for (int i=L; i>=0; i--){
                        LIS[i] = inputArray[p];
                        p = P[p];
            }

            return LIS;
      }

      // The main function
      public static void main(String[] args) throws IOException{

            // Start measuring time
            long startTime = System.currentTimeMillis();
            // Use string builder and buffer to read input file
            StringBuilder stringBuilder;

            // File opening
            BufferedReader bufferedReader = new BufferedReader(new
                                                FileReader("u.txt"));
```

```
            try{ // File reading
                  stringBuilder = new StringBuilder();
                  String line = bufferedReader.readLine();

                  while (line != null){
                        stringBuilder.append(line);
                        stringBuilder.append(",");
                        line = bufferedReader.readLine();
                  }
            }
            finally{
                  bufferedReader.close();
            }

            // Input data transforming (String -> Integer)
            String s = stringBuilder.toString();
            String[] help = s.split(",");
            int[] input = new int[help.length];

            for (int i=0; i<help.length; i++){
                  if (help[i].length()>0){
                        input[i]=Integer.parseInt(help[i]);
                  }
            }

            // Call the function to find LIS
            int[] subsequence = findLongestSubsequence(input);

            // Print out the LIS
            BufferedWriter outputWriter = null;
            outputWriter = new BufferedWriter(new
FileWriter("out.txt"));
            for (int i = 0; i < subsequence.length; i++) {
                outputWriter.write(subsequence[i]+", ");
      }
            outputWriter.flush();
            outputWriter.close();

            // Finish estimating time
            long estimatedTime = System.currentTimeMillis() - startTime;
            System.out.println("Time [s]: " +
(float)estimatedTime/1000);

            // Estimate used memory
            Runtime runtime = Runtime.getRuntime();
            runtime.gc();
            long total = Runtime.getRuntime().totalMemory();
            long free = Runtime.getRuntime().freeMemory();
            long used = total - free;
            System.out.println("Memory [B]: " + used);
      }
}
```