Primer pitanja za teorijski test iz Algoritama i struktura podataka, I smer, 2024/25.

KOREKNOST

1. Definisati pojam invarijante petlje.
2. Navesti šta je invarijanta spoljašnje petlje algoritma sortiranja umetanjem (Insertion sort).
3. Navesti šta je invarijanta algoritma binarne pretrage.
4. Koliki je otprilike opseg brojeva koji se mogu smestiti u promenljivu tipa int (32 bita), a koliki u promenljivu tipa long long (64 bita)?
5. Koje vrste verifikacije softvera postoje?

SLOŽENOST

6. Kolika je suma vrednosti 2^k za $k = 0$ do n?
7. Kolika je suma vrednosti $(1/2)^k$ za $k = 0$ do n?
8. Kojom rekurentnom jednačinom se može opisati vremenska složenost naredne funkcije?

```
int f(int n) {
    if (n == 0) return 1;
    else return 2*f(n/2)+3n;
}
```

(a) $T(n) = 2T(n/2) + O(1)$ (b) $T(n) = 2T(n/2) + O(n)$ (c) $T(n) = T(n/2) + O(1)$ (d) $T(n) = T(n/2) + O(n)$

9. Navesti while–petlju čija je složenost $O(\log n)$.
10. Kolika je suma vrednosti $(1/2)^k$ za $k = 0$ do n?
11. Navesti primer algoritma složenost $O(n!)$.
12. Koja je složenost narednog koda?

```
for(int i=0; i<n; i++)
    for(int j=0; j<i; j++)
        for(int k=0; k<j; k++)
            O(1)
```

(a) $O(n^2)$ (b) $O(n^3)$ (c) $O(n^2 \log n)$ (d) $T(n \log^2 n)$

13. Kojom brzinom asimptotski u beskonačnost raste suma $1^2 + 2^2 + ... + n^2$?
14. Šta je harmonijski red? Da li je on kovergentan? Kojom brzinom rastu njegove parcijalne sume?
15. Reši rekurentnu jednačinu T(n) = 2T(n-1) + O(1), T(0) = O(1).
16. Reši rekurentnu jednačinu T(n) = 3T(n/3) + O(n), T(0) = O(1).
17. Reši rekurentnu jednačinu T(n) = 5T(n/5) + O(1), T(0) = O(1).

18. Reši rekurentnu jednačinu T(n) = 3T(n/2) + O(n^2), T(0) = O(1).

19. Navedi bar dva algoritma tipa podeli pa vladaj koji imaju različite jednačine kojima se opisuje njihova složenost. Postavi i reši te jednačine.

20. Navesti bar dva primera algoritama koji imaju rekurentnu jednačinu T(n) = 2T(n/2) + O(n), T(0) = O(1). Koja je njihova složenost?

21. Navesti primer petlje for koja se izvršava u a) linearnoj b) logaritamskoj c) konstantnoj vremenskoj složenosti.

22. Navesti primer algoritma složenosti O($\log n$) i primer algoritma složenosti O($\sqrt{n}$). Koji se od njih smatra efikasnijim? Koliki je otprilike red veličine broja n da bi se algoritam mogao sprovesti za oko jedne sekunde za svaku od ove dve složenosti?

23. Koliki je otprilike maksimalni red veličine ulaza n koji se može obraivati algoritmima složenosti 2n?

24. Koliki je otprilike red veličine ulaza koji se na prosečnom današnjem računaru može rešiti algoritmom kvadratne složenosti za jednu sekundu?

25. Koliko operacija sabiranja celih brojeva otprilike može da izvrši prosečan današnji računar u toku jedne sekunde?

26. Koliko otprilike operacija izvršava algoritam složenosti n$\log n$ za ulaz veličine n=10^6?

OSNOVNE TEHNIKE ZA POBOLJŠANJE SLOŽENOSTI

27. Na koliko načina iz skupa od n elemenata možemo izabrati 2 elementa.

28. Kolika je složenost Euklidovog algortima koji se zasniva na oduzimanju?

29. Navesti kako izgleda rekurzivni poziv kod Euklidovog algoritma zasnovanom na ostacima:

```
int f(int a, int b)
{
    if (b == 0)
        return a;
    else return ..................;
}
```

30. Neka je dat red od n elemenata: $\frac{1}{x} - \frac{2}{x^2} + \frac{4}{x^3} + \ldots + \frac{(-2)^n}{x^n}$. Napisati petlju koja na efikasan način računa sumu ovog reda.

SORTIRANJE, BINARNA PRETRAGA

31. Koji je vremenski složenost QuickSort-a u proseku i u najgorem slučaju?

32. Koji je glavni preduslov da bi Binarna pretraga mogla da se koristi na datom nizu?

33. Koji je vremenski složenost MergeSort-a i kolika je prostorna složenost?

34. Kako QuickSort bira pivot element i zašto je izbor pivota važan za njegovu efikasnost?

35. Kojom rekurentnom jednačinom se može opisati vreme izvršavanja algoritma MergeSort?

36. Ukoliko je opseg vrednosti niza koji se sortira mali, kojom tehnikom može da se izvrši sortiranje i koja je složenost u tom slučaju?

37. Šta je kanonski oblik niza i koja je njegova svrha?

38. Navesti barem jednu funkciju u C++ koja koristi binarnu pretragu ili njenu varijaciju.

39. Šta radi funkcija lower_bound? Koju vrednost vraća ako se element ne nalazi u nizu?

40. Šta radi funkcija upper_bound? Koju vrednost vraća ako se element ne nalazi u nizu?

41. Neka se u rešenju nekog problema koristi tehnika binarne pretrage po rešenju i to uključuje ispitivanje opsega vrednosti $[1, m]$ i za svaku od tih vrednosti koju pretražujemo potrebno je ispitati i sve elemente niza duže n. Kolika je složenost ovakvog rešenja?

42. Opisati korak particionisanja kod QuickSort algoritma. Gde se može naći pivot nakon particionisanja? Koji uslov je zadovoljen nakon particionisanja?

43. Kod MergeSort algoritma šta prethodi koraku objedinjavanja i koji uslov mora da bude ispunjen? Kolika je složenost objedinjavanja?

KONSTRUKCIJA ALGORITMA INDUKCIJOM

44. Ukratko definisati šta je *rekurzivna funkcija* i kakav je to "problem manje dimenzije".

45. Napisati primer najjednostavnije rekurzivne funkcije i ilustrovati pozivni dijagram (rekurzivno stablo poziva).

46. Koja je razlika izmedju *rekurzivne* i *iterativne* implementacije, ako obe slede istu induktivnu ideju?

47. Definisati princip *matematičke indukcije* i objasniti osnovne korake (baza indukcije, induktivni korak).

48. Zašto rekurzija uvek podrazumeva da u nekom trenutku imamo *bazu*, odnosno granični slučaj?

49. Kako se matematička indukcija koristi da se dokaže *korektnost* rekurzivnog algoritma?

50. Objasniti šta je *induktivna pretpostavka* (pretpostavimo da su svi manji problemi već rešeni).

51. Da li se uvek može eliminisati rekurzija i napisati *petlju* (iterativnu verziju)? Koje su prednosti i mane jednog i drugog pristupa?

52. Dati primer rekurzivnog algoritma gde se problem ne svodi samo na $n-1$, već npr. na $n-2$ ili se deli na dva podproblema dimenzije $n/2$.

53. U kojoj meri se koncept indukcije razlikuje u tim slučajevima?

54. Objasniti postupak "osloboditi rekurziju" (tj. pretvoriti rekurzivni kod u iterativni) uz primer.

OSNOVNE STRUKTURE PODATAKA

55. Ako hip sadrži n elemenata onda je njegova visina reda: (a) $\log n$ (b) 2^n (c) n (d) $n \log n$

56. Ako su poznati pokazivači na početak i na kraj jednostruko povezane liste, koja je složenost operacija dodavanja i brisanja sa početka i sa kraja liste.

57. Kako je implementirana neuređena mapa, a kako uređena mapa?

58. Navesti bar tri operacije koje se efikasno mogu izvršiti nad uređenim skupom, a ne mogu nad neuređenim.

59. Koje su osnovne operacije koje podržava tip podataka skup?

60. Koja je složenost provere da li se element nalazi u skupu ako je skup implementiran preko a) pretraživačkog drveta koje ne mora biti balansirano b) pretraživačkog drveta koje je uvek balansirano?

61. Koja su dva osnovna načina implementacije asocijativnih struktura podataka (skupova i mapa)?

62. Kada se koristi heš-tabela kako se dobija pozicija u tabeli na kojoj bi trebalo da se smesti dati element?

63. Šta je kolizija heš-funkcije? Opisati neki način razrešavanja kolizija.

64. Koja je složenost operacija dodavanja i brisanja elemenata sa početka i sa kraja i koja je složenost indeksnog pristupa kod reda sa dva kraja ako je on implementiran preko a) dvostruko povezane liste i b) preko strukture podataka dek (engl. deque)?

65. Koje su osnovne tri operacije reda sa prioritetom?

66. Kako se u memoriji predstavlja struktura podataka hip?

67. Koji je odnos roditelja i njegova dva deteta u min-hipu?

68. Koji je odnos roditelja i njegova dva deteta u pretraživačkom binarnom drvetu?

69. U programskom jeziku C++, definisati čvor jednostruko povezane liste, binarnog stabla i dvostruko povezane liste.

70. Napisati funkciju koja dodaje novi čvor na početak dvostruko povezane liste.

71. Šta je balansirano binarno stablo? Zašto je važno da stablo u implemtaciji mape/skupa bude balansirano?

PODELI PA VLADAJ

72. Ukratko definisati šta je tehnika *podeli pa vladaj*. Koji su glavni koraci ove tehnike?

73. Napisati rekurentnu jednačinu koja opisuje vremensku složenost algoritma *merge sort*. Koje je rešenje te jednačine?

74. Uporediti *merge sort* i *quick sort*:

    (a) Po kojoj rekurentnoj jednačini se može opisati *merge sort*?

    (b) Koja rekurentna jednačina opisuje *quick sort* u prosečnom slučaju, a koja u najgorem slučaju?

    (c) Zašto *quick sort* može da degeneriše u $O(n^2)$?

75. Detaljno opisati fazu *spajanja* (merge) kod algoritma *merge sort*. Koliko traje spajanje dva podniza od po $n/2$ elemenata?

76. Objasniti ulogu *rekurzije* i *indukcije* u konstrukciji algoritma *merge sort*. Zbog čega se ova tehnika naziva *induktivno-rekurzivnom* konstrukcijom?

77. Kod balansiranih struktura (npr. kod binarnog stabla), podproblemi imaju skoro duplo manju dimenziju. Navedite konkretan primer gde se ovaj princip pojavljuje i navesti rekurentnu jednačinu koja opisuje njegovo vreme izvršavanja.

78. Posmatrati varijantu problema *maksimalni zbir podniza* koja se rešava tehnikom podeli-pa-vladaj. Koja se rekurentna jednačina tom prilikom dobija i kako se rešava?

    (a) Koja je vremenska složenost osnovne (klasične) podeli-pa-vladaj varijante za maksimalni zbir podniza?

    (b) Kako se može dodatnim *ojačanjem* induktivne hipoteze postići bolja ($O(n)$) složenost?

79. Koje su tipične greške ili poteškoće koje se javljaju kod algoritama koji se zasnivaju na *podeli-pa-vladaj*? Navedite dve situacije u kojima ova tehnika može dati neefikasno rešenje ($O(n^2)$ ili gore).

80. Navedite barem dva algoritma tipa *podeli-pa-vladaj* (pored *merge sorta* i *quick sorta*) sa različitim rekurentnim jednačinama. Postavite i rešite te jednačine, ili makar opišite njihovo približno rešenje (npr. pomoću *master teoreme*).

81. Navedite situaciju u kojoj algoritam *podeli-pa-vladaj* ne bi bio prikladan ili ne bi doneo značajno ubrzanje u odnosu na jednostavnije iterativne metode. Objasniti zašto.

GENERISANJE KOMBINATORNIH OBJEKATA

82. Ukratko navesti šta podrazumevamo pod: (a) podskupovima, (b) varijacijama (sa i bez ponavljanja), (c) kombinacijama (sa i bez ponavljanja) i (d) permutacijama. Zašto se kaže da sve ove strukture u najgorem slučaju imaju eksponencijalno mnogo elemenata u odnosu na n?

83. Rekurzivno generisanje podskupova:

    (a) Navesti rekurzivnu funkciju (obilaženjem u dubinu) koja enumeriše sve podskupove skupa $\{1, 2, \ldots, n\}$.

    (b) Koliko ukupno čvorova ima odgovarajuće rekurzivno stablo, a koliko listova?

    (c) Zašto se rekurzija često naziva "obilaskom implicitnog stabla" u ovom kontekstu?

84. 0–1 reprezentacija podskupova.

    (a) Objasniti kako se skup $\{a_0, a_1, \ldots, a_{n-1}\}$ može predstaviti pomoću n-torke koja sadrži samo 0 i 1 (tj. bit-vrednosti).

    (b) Zašto se u ovoj reprezentaciji samo listovi u rekurzivnom stablu odnose na validne podskupove?

    (c) Napisati (u pseudo-kodu ili C++-u) funkciju koja takve 0–1 nizove *iterativno* generiše u leksikografskom poretku.

85. Leksikografski poredak i "sledeći" objekat:

    (a) Definisati leksikografski poredak nad n-torakama prirodnih brojeva.

    (b) Napisati algoritam (ili opisati postupak) koji za dati podskup $S \subseteq \{1, 2, \ldots, n\}$ (zadat kao strogo rastuća lista elemenata) pronalazi *sledeći* podskup u leksikografskom poretku.

    (c) Kolika je vremenska složenost ovog pristupa i od čega zavisi?

    (d) Posmatrano leksikografski, za dati skup $\{0, 1, 2, 3\}$ koji je sledeći podskup podskupa: a) $\{0, 2\}$, b) $\{0, 3\}$?

86. Varijacije i kombinacije:

    (a) Koliko ima varijacija od k elemenata skupa od n elemenata (sa i bez ponavljanja)?

    (b) Napisati rekurzivnu funkciju za nabrajanje svih *kombinacija* od k elemenata (bez ponavljanja) iz skupa $\{1, \ldots, n\}$, tako da se generišu u *leksikografskom* poretku.

    (c) Koja je prosečna (ili očekivana) dužina putanje u takvom rekurzivnom stablu?

87. Koliko ima permutacija n različitih elemenata?

88. Zašto se kaže da je generisanje svih kombinatornih objekata često neizvodljivo za veće n (eksponencijalna složenost)?

89. Razmotriti generisanje svih podskupova *pre-order*, *in-order* i *post-order* obilaskom odgovarajućeg implicitnog stabla. U kom od ta tri obilaska se prirodno generišu podskupovi u strogo rastućem (leksikografskom) poretku?

90. Kako biste izmenili rekurzivni kod za generisanje svih kombinacija da biste dobili *kombinacije sa ponavljanjem*?

91. Navedite (bar) jednu standardnu biblioteku ili funkciju u C++ koja pomaže pri generisanju permutacija ili kombinacija.

92. Opiši algoritam za nabrajanje svih varijacija sa ponavljanjem dužine n skupa $\{1, \ldots, k\}$.

BEKTREKING I GRUBA SILA

93. Definisati pojam "grube sile" i ukratko opisati zašto je složenost takvih pristupa uglavnom eksponencijalna.

94. Navedite primer problema (u vidu matrice) koji se može rešiti *potpunom* enumeracijom svih mogućih rešenja.

95. U kom slučaju, i pored eksponencijalne složenosti, može biti prihvatljivo koristiti brute force?

96. U čemu se suštinski razlikuje backtracking od proste *grube sile*?

97. Šta znači "odsecanje" u kontekstu backtracking pretrage i kako se njime postiže ubrzanje?

98. Navesti jedan konkretan primer kriterijuma za odsecanje koji bi se mogao primeniti pri rešavanju problema "n dama" (n-queens) na šahovskoj tabli.

99. Ukratko objasniti razliku izmedjuu DFS i BFS.

100. Navedite primer lavirinta predstavljenog kao matrica gde želimo da utvrdimo da li postoji put od gornjeg levog ćoška do donjeg desnog. Koje prednosti ima BFS u odnosu na DFS, a koje nedostatke, za ovaj tip zadatka?

101. Zašto BFS nužno nalazi najkraći put u matrici (ako sve "cene" ili "težine" koraka imaju istu vrednost)?

102. U slucaju pronalaska puta kroz lavirint (zadat matricom) – kako obezbediti da se ne "vrti" u beskonačnoj petlji ukoliko je dozvoljeno kretati se u svim pravcima (gore, dole, levo, desno)?

103. U slucaju pronalaska puta kroz lavirint (zadat matricom) – kolika je vremenska složenost u najgorem slučaju, i kako se može smanjiti preuranjenim odsecanjem?

104. Zašto je rasporedjivanje n dama (n-queens problem) tipičan primer backtracking-a?

105. Koji jednostavni kriterijum odsecanja možemo primeniti već pri postavljanju svake nove dame?

106. Koja razlika u implementaciji backtracking-a nastaje ako samo želimo *prvo* validno rešenje, za razliku od slučaja kada hoćemo *sva* validna rešenja?

107. Ilustrovati to na primeru "n dama" ili "lavirinta" – gde biste prekinuli pretragu?

108. Dati primer situacije u rasporedjivanju n dama ili bojenju matrice, gde se može *zaključiti* vrednost neke promenljive bez enumeracije svih kandidata.

109. Zašto ovakvi heuristički koraci ne utiču na korektnost rešenja (pod uslovom da su ispravno implementirani)?

110. Zašto i backtracking i brute force, u najgorem slučaju, ostaju eksponencijalni algoritmi?

REKURZIJA I DINAMIČKO PROGRAMIRANJE

111. Navesti primer problema preklapanja rekurzivnih poziva (ponovljenih rekurzivnih poziva) i opisati kako se on rešava.

112. Da li se prilikom tehnike memoizacije eliminiše rekurzija?

113. Zašto se dešava da rekurzivna definicija funkcije (poput `fibonaci(n)`) pravi identične rekurzivne pozive više puta?

114. Navedite još jedan primer (pored Fibonacija) gde se javlja problem preklapanja poziva.

115. Definisati tehniku *memoizacije* i pokazati kako se ona primenjuje na primeru `fibonaci(n)`.

116. Kako se mogu čuvati već izračunate vrednosti?

117. U kom trenutku se vrši provera da li je vrednost za dati parametar već izračunata?

118. Koja je vremenska i memorijska složenost ovako memoizovanog izračunavanja `fibonaci(n)`?

119. U čemu se razlikuje programiranje naviše DP pristup od memoizacije?

120. Zašto se često dešava da u praksi i memoizacija i bottom-up DP popune vrednosti za sve moguće parametre?

121. Na primeru Fibonačijevog niza, zašto ne moramo čuvati sve izračunate vrednosti?

122. Definisati *problem ranca* sa ograničenjem težine. Koji su ulazni parametri?

123. Izvršiti memoizaciju sledeće funkcije:

```
int minBrojNovcica(int v[], int n, int s) {
    if (s == 0) return 0;
    if (n == 0) return INF;
    int br = minBrojNovcica(v, n-1, s);
    if (s >= v[n-1])
        br = min(br, minBrojNovcica(v, n, s - v[n-1]) + 1);
    return br;
}
```

124. Navesti četiri koraka koja su tipična za DP?

125. Zašto je *rekurzija* i *indukcija* (odnosno iteracija) u osnovi svih DP rešenja?

126. U kojim slučajevima DP može biti memorijski previše zahtevan, iako je vremenski prihvatljiv?

GRAMZIVI ALGORITMI

127. Definisati šta su *gramzivi (greedy) algoritmi.*

128. U čemu se razlikuju od algoritama zasnovanih na pretrazi (npr. backtracking) i od dinamičkog programiranja?

129. Navedite bar jedan klasičan problem koji se tipično uspešno rešava gramzivim pristupom.

130. Objasniti pojam "lokalno optimalni izbor" i zašto se veruje da on vodi do globalno optimalnog rešenja u gramzivim algoritmima.

131. Dati primer gde lokalno optimalni izbor u svakom koraku garantuje optimizaciju.

132. Navedite primer problema kod koga lokalno optimalni izbor *ne* vodi do globalnog optimuma.

133. Zašto je, pored ispravnosti (da rešenje zadovoljava uslove zadatka), potrebno dokazati i *optimalnost*?

134. Objasniti osnovne tehnike dokazivanja optimalnosti.

135. Navedite primer problema kod koga gramzivi algoritam može dati rešenje koje nije optimalno.

136. Zašto se, uprkos tome, gramzivi algoritam ponekad ipak koristi kao *heuristika*?