Vastaa tentin järjestäjän antamalle paperille (ei kysymyspaperille). Kirjoja, laskinta tms. ei saa olla tentissä. Kukin tehtävä on 1 tai 2 pisteen arvoinen. Muiden kuin koodaa tai piirrä -tehtävien mallivastaukset ovat melko lyhyet, 1, . . . , 4 riviä.

1. Oheisen algoritmin nimi on INSERTIONSORT. Mikä on sen suoritusaika Θ-merkinnällä ilmaistuna, jos kaikki alkiot ovat yhtäsuuret? Perustele.

```
1  for i := 1 to A.koko − 1 do
2    apu := A[i]; j := i
3    while j > 0 && A[j − 1].x > apu.x do
4      A[j] := A[j − 1]; j = j − 1
5    A[j] := apu
```

   $\Theta(n)$. Kun kaikki alkiot ovat yhtäsuuret, tuottaa rivin 3 testi $A[j-1].x > apu.x$ aina epätoden, joten riville 4 ei koskaan mennä.

2. Oletetaan, että A:ssa on ensin $\sqrt{n}$ ykköstä ja loput alkiot ovat nollia. Mikä on INSERTION-SORT:n suoritusaika? Ei tarvitse perustella.

   $\Theta(n\sqrt{n})$.

3. Miten INSERTIONSORT huononee tai paranee, jos rivi 1 muutetaan muotoon **for** $i := 0$ **to** $A.koko - 1$ **do** ?

   Rivit 1, 2 ja 5 sekä rivin 3 alkuosa suoritetaan turhaan, kun $i = 0$. Se ei merkittävästi huononna eikä paranna algoritmia.

4. Miten INSERTIONSORT huononee tai paranee, jos rivi 3 muutetaan muotoon **while** $j \geq 0$ && $A[j-1].x > apu.x$ **do** ?

   Indeksoi laittomasti alkiota $A[-1]$ (paitsi jos pienin alkio on alussa). Ohjelmointikielestä riippuen se voi esimerkiksi kaataa ohjelman.

5. Miten INSERTIONSORT huononee tai paranee, jos rivi 3 muutetaan muotoon **while** $j > 0$ && $A[j-1].x \geq apu.x$ **do** ?

   Vaihtaa yhtäsuurten alkioiden keskinäisen järjestyksen takaperin. Vakaus menetetään ja algoritmi hidastuu.
   Vakauden menetys tuottaa yksinäänkin 1 pisteen, hidastuminen 0,5 p.

6. Anna INSERTIONSORT:n ulommalle silmukalle invariantti.

   Osassa $A[0 \ldots i-1]$ on alkuperäiset alkiot kasvavassa järjestyksessä. Osassa $A[i \ldots n-1]$ on alkuperäiset alkiot alkuperäisessä järjestyksessä.

7. Anna INSERTIONSORT:n sisemmälle silmukalle invariantti.

   Osissa $A[0 \ldots j-1]$ ja $A[j+1 \ldots i]$ on yhteensä osan $A[0 \ldots i-1]$ alkuperäiset alkiot kasvavassa järjestyksessä. Alkuperäinen $A[i]$ on muuttujassa apu. Joko $j = i$ tai $apu.x < A[j+1].x$.

8. Tarkastellaan binäärikekoa (binary heap). Vähintään ja enintään kuinka monella solmulla on täsmälleen yksi lapsi? Mitä solmujen määrästä voidaan päätellä, jos jollakin solmulla on täsmälleen yksi lapsi?

Täsmälleen yksi lapsi on yhdellä tai ei yhdelläkään solmulla. Jos jollakin solmulla on täsmälleen yksi lapsi, niin solmujen määrä on parillinen.

9. Kerro yksi kekojärjestämisen (heapsort) vahvuus ja yksi heikkous.

   Vahvuus: Lisämuistin tarve on vain vakio. Heikkous: Ei ole vakaa.

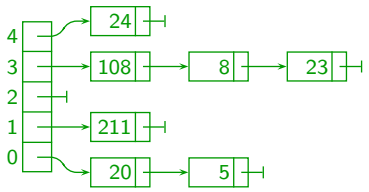
10. Kuinka paljon keosta poistaminen käyttää aikaa hitaimmillaan ja nopeimmillaan?

    $\Theta(\log n)$ ja $\Theta(1)$, missä n on alkioiden määrä keossa.

11. Mikä on lopputulos, kun keosta $[8,6,5,2,4,2]$ poistetaan yksi alkio?

    $[6,4,5,2,2]$

12. Piirrä hajautustaulu, jonka hajautusfunktiona on x mod 5 eli `x % 5`, ja jossa on alkiot 211, 23, 5, 20, 24, 8 ja 108.



13. Kerro yksi hajautustaulujen etu verrattuna punamustiin puihin.

    Alkion löytää avaimen perusteella keskimäärin ajassa $O(1)$.

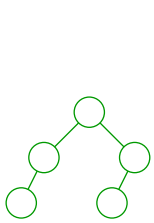
14. Kerro yksi punamustien puiden etu verrattuna hajautustauluihin.

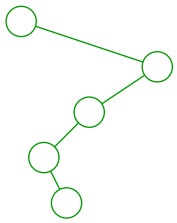
    Alkioita voi selata avainten mukaisessa suuruusjärjestyksessä.

15. Miten käsite ”binäärihakupuu” eroaa käsitteestä ”binääripuu”?

    Binäärihakupuun jokaisessa solmussa on avain. Kunkin solmun vasemman alipuun solmujen avaimet ovat enintään yhtäsuuret ja oikean alipuun solmujen avaimet vähintään yhtäsuuret kuin solmun oma avain.

16. Binääripuun solmun korkeus on mahdollisimman pitkän solmusta johonkin lehteen vievän polun pituus. Piirrä binääripuu, jossa on viisi solmua, ja jonka juuren korkeus on mahdollisimman pieni. Piirrä binääripuu, jossa on viisi solmua, ja jonka juuren korkeus on mahdollisimman suuri.





17&18. Minkä tehtävän Dijkstran algoritmi ratkaisee, ja miten se toimii?

Etsii mahdollisimman lyhyen reitin lähdöstä maaliin. Solmussa on lyhin löydetty etäisyys lähdöstä ja linkki reitillä edellliseen solmuun. Algoritmi laittaa lähdön prioriteettijonoon. Kunnes jono on tyhjä tai sieltä tuli maali, se ottaa jonosta solmun, jolla on lyhin etäisyys, ja tutkii siitä lähtevät kaaret. Se laittaa jonoon solmut, joihin näin löytyi entistä lyhyempi reitti. (Myönnän mokan, on pitempi mallivastaus kuin lupasin paperin alussa.)

Juoksukisan reitin varrelle tarvitaan h huoltopistettä. Koska ne tarvitsevat tilaa yms., niitä ei voi laittaa minne tahansa. Mahdollisia paikkoja on $n-2$ kappaletta, missä $n \geq 2$. Ne on ilmoitettu metreissä juoksureittiä pitkin taulukossa `sijainnit`, missä `sijainnit[0]` $=$ 0 on lähtö, `sijainnit[1]` on ensimmäinen mahdollinen paikka, ..., `sijainnit[`$n-2$`]` on viimeinen mahdollinen ja `sijainnit[`$n-1$`]` on maali. *Osuus* on lähdöstä ensimmäiseen huoltopisteeseen, huoltopisteestä seuraavaan tai viimeisestä huoltopisteestä maaliin. Pisin osuus halutaan minimoida.

19. Olkoon `sijainnit` $=$ $[0,9,18,27,35,44,52]$. Jos huoltopisteitä voidaan perustaa enintään 3, niin mitkä ovat niiden etäisyydet lähdöstä?

    9, 18, 35

20&21. Kirjoita yksinkertainen algoritmi, joka selvittää, onko mahdollista, että pisin osuus on pituudeltaan enintään `pituus`, kun huoltopisteiden määrä on enintään `hpm`.

```
int i = 1;
for( int j = 0; j <= hpm; ++j ){
  int raja = sijainnit[ i - 1 ] + pituus;
  while( i < sijainnit.size() && sijainnit[ i ] <= raja ){ ++i; }
}
return i == sijainnit.size();
```

Oheinen algoritmi valitsee `hpm` huoltopistettä välille lähtö, ..., `paikka`. Aluksi `lyhin[ paikka ][ 0 ]` on `sijainnit[ paikka ]` ja muut `lyhin[ paikka ][ hpm ]` ovat ääretön.

```
1  int paras( int paikka, int hpm ){
2    if( lyhin[ paikka ][ hpm ] == ääretön ){
3      for( int i = 0; i <= paikka; ++i ){
4        int max = sijainnit[ paikka ] - sijainnit[ i ];
5        int alku = paras( i, hpm - 1 );
6        if( alku > max ){ max = alku; }
7        if( max < lyhin[ paikka ][ hpm ] ){
8          lyhin[ paikka ][ hpm ] = max; valittu[ paikka ][ hpm ] = i;
9        }
10     }
11   }
12   return lyhin[ paikka ][ hpm ];
13 }
```

22. Ilmoita algoritmin suoritusaika n:n ja h:n funktiona O-merkinnällä.

    $O(n^2h)$

23. Perustele edellinen vastauksesi.

    Taulukossa `lyhin` on nh alkiota, kullekin niistä `for`-silmukka suoritetaan enintään kerran, `for`-silmukka tekee kerrallaan enintään $n+1$ kierrosta, ja kaikki muu on vakioaikaista paitsi `for`-silmukan suorittava rekursiokutsu.

24. Miten algoritmia voi nopeuttaa puolitushaun avulla? Sanallinen kuvaus riittää, ei tarvita pseudo- eikä ohjelmakoodia.

    Etsitään paras `max` ja `i` puolitushaulla eikä lineaarisesti.

25. Ilmoita edellisen kohdan mukaisesti nopeutettu suoritusaika O-merkinnällä.

    $O(nh\log n)$

26&27. Kirjoita algoritmi, joka tulostaa huoltopisteiden etäisyydet lähdöstä.

```
void tulosta( int paikka, int hpm ){
  int edell = valittu[ paikka ][ hpm ];
  if( hpm ){ tulosta( sijainnit, edell, hpm - 1 ); }
  std::cout << ' ' << sijainnit[ paikka ];
}
```

28. Minkä arvelet kurssilla olleen kaikkein hyödyllisintä tulevalle työurallesi?

    Vastauksista on vaikea löytää yhtenäistä linjaa, mutta eniten esillä oli ehkä kokemuksen kertyminen ohjelmoinnista tai algoritmiasioista.

29&30. Mitä asiaa olisit halunnut kurssiin lisää, ja miksi? Minkä asian voisi jättää kurssista pois, jotta saataisiin tilaa asialle, jota haluaisit lisää?

    Vaikutelmakseni tuli, että eniten toivottiin lisää pieniä ohjelmointitehtäviä sekä suorempaa yhteyttä luentojen ja ohjelmointitehtävien / tentin välille. Mutta tästäkään ei ollut yksimielisyyttä.

loppu