1. Íhugið eftirfarandi lykkjumynstur:

```
// F
while( C1 ) {
    // I1
    S1
    // I2
    if( C2 ) continue;
    S2
}
// E
```

   Hvaða samband þarf að gilda milli F, C1, I1, I2, S1, C2, S2 og E til að lykkja þessi sé rökrétt?

   Svar:

   (a) $F \Rightarrow I1$

   (b) $\neg C1 \wedge I1 \Rightarrow E$

   (c) $\{I1\}\, S1\, \{I2\}$

   (d) $I2 \wedge C2 \Rightarrow I1$

   (e) $\{I2 \wedge \neg C2\}\, S2\, \{I1\}$

2. Hvaða lágmarkssamband þarf að gilda milli lýsinga boða í undirklasa og yfirklasa til að tryggt sé að forrit sem nota þessa klasa séu rökrétt (miðað við að röksemdafærsla sé að öðru leyti rétt)?

   Svar: Ef við gerum ráð fyrir að klasi B erfi frá klasa A og að báðir klasarnir innihaldi sama tilviksboð f, með eftirfarandi lýsingum:

```
// ** Lýsing í klasa A **
// Notkun:  x.f(...);
// Fyrir:   FA
// Eftir:   EA

// ** Lýsing í klasa B **
// Notkun:  x.f(...);
// Fyrir:   FB
// Eftir:   EB
```

   Þá þarf að gilda:

(a) $FA \Rightarrow FB$

(b) $EB \Rightarrow EA$

3. Íhugið eftirfarandi lykkju, sem hefur þann tilgang að reikna x^y (`x^y`) þar sem x er fleytitala og y er jákvæð heiltala:

```
// y >= 0
double p=1.0, q=x;
int r=y;
while( r != 0 ) {
    // x^y == p*q^r, r>=0
    ... (breytum hvorki x né y)
}
// ???
```

- Hvert er rökrétt eftirskilyrði lykkjunnar?
  Svar: `p==x^y`.
- Hvernig getum við forritað stofninn í lykkjuna á hraðvirkan hátt þannig að fjöldi umferða verði $O(\log y)$?
  Svar:

```
// y >= 0
double p=1.0, q=x;
int r=y;
while( r != 0 ) {
    // x^y == p*q^r, r>=0
    if( r%2==1 ) p*=q;
    q *= q;
    r /= 2;
}
// ???
```

4. Fyllið inn þar sem spurningarmerkin eru, þ.e. skrifið hvað á að koma í stað `?1?`, o.s.frv. Þetta eru samtals sjö svör.

```
// Notkun: k = leita(f,i,j,x);
// Fyrir:  f[i..j-1] er í vaxandi röð og inniheldur
//         a.m.k. eitt gildi sem er stærra en x.
// Eftir:  k vísar á fremsta gildi í f[i..j-1] sem
//         er stærra en x.
int leita( double[] f, int i, int j, double x ) {
```

```
  int p=?1?, q=?2?;
  while( ?3? ) {
    // | <=x | óþekkt/unknown | >x |
    //  i     p                q    j
    int m = (?4?)/2;
    if( f[m] ?5? x )
      p = ?6?;
    else
      q = ?7?;
  }
  return p;
}
```

Svar:

```
// Notkun: k = leita(f,i,j,x);
// Fyrir:  f[i..j-1] er í vaxandi röð og inniheldur
//         a.m.k. eitt gildi sem er stærra en x.
// Eftir:  k vísar á fremsta gildi í f[i..j-1] sem
//         er stærra en x.
int leita( double[] f, int i, int j, double x ) {
  int p=i, q=j;
  while( p!=q ) {
    // | <=x | óþekkt/unknown | >x |
    //  i     p                q    j
    int m = (p+q)/2;
    if( f[m] <= x )
      p = m+1;
    else
      q = m;
  }
  return p;
}
```

5. Gefið er eftirfarandi stef:

```
// Notkun:  rolldown(f,i,j);
// Fyrir:   Svæðið f[i+1..j] uppfyllir hrúguskilyrði
//          fyrir hrúgur með stærsta gildi efst.
// Eftir:   Búið er að víxla gildum í svæðinu f[i..j]
//          þannig að það svæði uppfyllir nú
```

```
//          hrúguskilyrðið.
void rolldown( double[] f, int i, int j );
```

Skrifið heapsort stef (með lýsingu - notkun, fyrir og eftir) með hjálp þessa stefs. Ekki þarf að forrita rolldown stefið.

Ef þið viljið megið þið gera ráð fyrir að til sé `rollup` stef, svipað og `rolldown`, nema í hina áttina, en þið verðið þá að lýsa því stefi (notkun/fyrir/eftir). Þið þurfið ekki að forrita það stef.

Athugið: Við reiknum með því að rót hrúgunnar sé `f[0]`.

Svar:

```
// Notkun: heapsort(f,n);
// Fyrir:  f[0..n] er svæði í f, n>0.
// Eftir:  Búið er að raða svæðinu í vaxandi röð.
void heapsort( double[] f, int n )
{
    int i=n/2+1;
    while( i!=0 )
    {
        // f[i+1..n] uppfyllir hrúguskilyrði.
        rolldown(f,i--,n);
    }
    i=n;
    while( i!=0 )
    {
        // f[0..i] er hrúga.
        // f[i+1..n] inniheldur stærstu gildi í vaxandi röð.
        double x = f[0];
        f[0] = f[i];
        f[i] = x;
        rolldown(f,0,--i);
    }
}
```

6. Tilgreinið tímaflækju eftirfarandi röðunaraðferða miðað við að raðað sé n slembitölum. Tilgreinið einnig hvort tímaflækjan sé tími í versta tilfelli, meðaltími eða eitthvað annað.

   (a) Heapsort

   (b) Merge-sort

(c) Quicksort

(d) Insertion-sort

(e) Radix-sort

(f) Röðum með því að moka gildunum í AVL tré (sem upphaflega er tómt) og moka þeim síðan út í vaxandi röð
Sort by shovelling the numbers into an AVL tree (which is originally empty) and then shovelling them out in ascending order

(g) Röðum með því að moka gildunum í Splay tré (sem upphaflega er tómt) og moka þeim síðan út í vaxandi röð
Sort by shovelling the numbers into a Splay tree (which is originally empty) and then shovelling them out in ascending order

Svar:

(a) Heapsort: $O(l \log n)$ í versta tilfelli.

(b) Merge-sort: $O(l \log n)$ í versta tilfelli.

(c) Quicksort: $O(l \log n)$ að meðaltali.

(d) Insertion-sort: $O(n^2)$ í versta tilfelli (og að meðaltali).

(e) Radix-sort: $O(nk)$ í versta tilfelli, þar sem n er fjöldi gilda og k er fjöldi stafa í hverju gildi.

(f) Röðum með því að moka gildunum í AVL tré (sem upphaflega er tómt) og moka þeim síðan út í vaxandi röð: $O(n \log n)$ í versta tilfelli.

(g) Röðum með því að moka gildunum í Splay tré (sem upphaflega er tómt) og moka þeim síðan út í vaxandi röð: $O(n \log n)$ í versta tilfelli.

7. Lýsið quicksort. Tilgreinið tímaflækju quicksort og rökstyðjið hana.

Svar: Quicksort gengur þannig fyrir sig að byrjað er með svæði `f[i..j-1]` í einhverju fylki `f`. Til að raða svæðinu í vaxandi röð er valið eitthvert vendistak `p`, sem yfirleitt er eitthvert gildi í svæðinu, og gildunum er síðan víxlað þannig að þeim er skipt í þrjú svæði, `f[i..k-1]` sem inniheldur gildi sem öll eru $\leq$ `p`, `f[k..n-1]` inniheldur gildi jöfn `p`, og `f[n..j-1]` inniheldur gildi $\geq$ `p`. Það er nauðsynlegt að tryggt sé að í mesta lagi eitt þessara svæða sé tómt. Þegar þessari skipringu er lokið er svæðunum `f[i..k-1]` og `f[n..j-1]` raðað (yfirleitt með quicksort, á endurkvæman hátt). Að sjálfsögðu þarf ekki að fara í gegnum þetta ferli ef fjöldi gilda í svæðinu `f[i..j-1]` er minni en 2.

Tímaflækja quicksort er að meðaltali $O(n \log n)$. Tímaflækjuna má rökstyðja með því að benda á að tímaflækja skiptingarinnar er línuleg, og heildartímaflækja skiptingarinnar í hverri hæð í endurkvæminni er því í mesta

lagi $O(n)$. Afgangurinn af tímaflækjunni ræðst af dýpt endurkvæmninnar og heildartímaflækjan verður margfeldið af tímaflækju skiptingarinnar og dýptinni. En það er nokkuð ljóst að ef við erum ekki kerfisbundið óheppin í skiptingunni mun dýpt endurkvæminnar verða $O(\log n)$. Heildartímaflækjan er því $O(n \log n)$.

8. Hver er fastayrðing ytri lykkju í insertion-sort? Tilgreinið tímaflækju insertion-sort i versta tilfelli og rökstyðjið hana.

   Svar: Fastayrðingin í ytri lykkju sem raðar svæði `f[i..j-1]` er eftirfarandi (eða eitthvað álíka):

   ```
   while( k!=j )
   {
       // f: | í vaxandi röð | óþekkt |
       //     i               k        j
       ...
   }
   ```

   Tímaflækja insertion sort, í versta tilfelli, er $O(n^2)$, þar sem n er fjöldi sæta í svæðinu sem raða skal. Fjöldi umferða í ytri lykkjunni er n og ef við erum óheppin í innri umferð munum við þurfa að bera saman nýtt gildi við sérhvert gildanna í svæðinu sem er í vaxandi röð, eða a.m.k. hliðra helmingnum af því svæði. Tímaflækja innri umferðarinnar er því í réttu hlutfalli við stærðina á svæðinu í vaxandi röð, sem er að meðaltali af stærð $\frac{n}{2}$, sem er $O(n)$. Heildartímaflækjan er því $O(n^2)$.

9. Íhugið eftirfarandi myndir af trjám. Segið til fyrir hverja mynd hvort hún getur staðið fyrir eitt eða fleiri af eftirfarandi:

   - Tvíleitartré
   - AVL-tré
   - Splay-tré
   - Rautt-svart tré (ef svo er, tilgreinið þá einnig hvaða hnúta má mála rauða til að það gangi upp)
   - 2-3 tré
   - Hrúga með hæsta gildi efst
   - Hrúga með minnsta gildi efst

   Athugið að hér er ætlast til að öll tvíleitartré séu með gildin í vaxandi *in-order* röð.

Athugið einnig að sama mynd af tré getur vel staðið fyrir fleiri en eina af þessum upp töldu gerðum af trjám.

Svar:

Tré A: Tvíleitartré, splay-tré.

Tré B: Tvíleitartré, AVL-tré, splay-tré, rautt-svart tré (málum 2 rautt, allt hitt svart, eða 1,3 rauð, allt hitt svart).

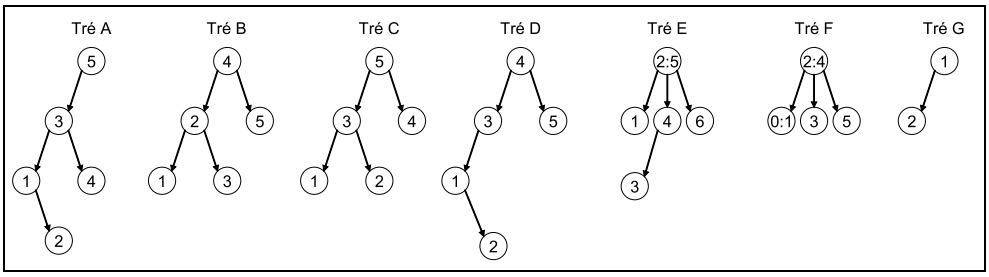
Tré C: Hrúga með stærsta efst.

Tré D: Tvíleitartré, splay-tré.

Tré E: Ekkert.

Tré F: 2-3 tré.

Tré G: Hrúga með minnsta efst.



10. Hver er fastayrðing gagna fyrir rauð-svört tré?

    Svar:

    - Sérhver hnútur í rauð-svörtu tré er annaðhvort rauður eða svartur.
    - Rótin er svört.
    - Foreldri rauðs hnútar er svartur.
    - Sérhver leið frá rót til hnútar sem hefur annaðhvort ekkert vinstra undirtré eða ekkert hægra undirtré (eða vantar bæði) hefur sama fjölda svartra hnúta.

11. Í tætitöflum og í fleiri vandamálum eru notuð stækkanleg fylki sem hafa sömu aðgerðir og venjuleg fylki en auk þess aðgerð til að bæta einu sæti aftast á fylkið. Hver er tímaflækja þessarar aðgerðar og hvers konar tímaflækja er það (versta tilfelli, meðaltal eða innistæðubungin)? Lýsið því hvernig aðgerðin er útfærð til að ná þessari tímaflækju.

    Svar: Innistæðubundin tímaflækja aðgerðarinnar er $O(1)$. Útfærslan er þannig að undirliggjandi fylki er tvöfaldað að stærð þegar þörf krefur.

12. Gerið grein fyrir einni eða fleiri aðgerðum sem AVL-tré gera kleift að gera á hraðvirkari hátt en tætitöflur gera kleift. Gerið einnig grein fyrir einni eða fleiri aðgerðum sem tætitöflur gera kleift að gera á hraðvirkari hátt en AVL tré.

    Svar: Í AVL-tré má leita að næsta gildi á eftir tilteknu gildi eða næsta gildi á undan tilteknu gildi með tímaflækju $O(\log n)$. Slíka aðgerð er ekki hægt að gera á jafn hraðvirkan hátt í tætitöflu. Í tætitöflu má hins vegar bæta gildi við eða eyða gildi með (innistæðubundinni) tímaflækju $O(1)$, sem myndi kosta $O(\log n)$ í AVL-tré.

13. Skrifið klasa í Java eða C++ fyrir forgangsbiðröð heiltalna. Þið megið sleppa því að forrita boðin, nema fyrir aðferðina til að bæta gildi í forgangsbiðröðina, en munið að hafa notkun, forskilyrði og eftirskilyrði fyrir öll boð og munið að hafa skýra fastayrðingu gagna.

    Svar (Java):

```
// Eintök af klasanum FBidrod eru forgangsbiðraðir
// heiltalna.
class FBidrod
{
    int n;
    int[] f;
    // Fastayrðing gagna:
    //   Gildin í forgangsbiðröðinni eru í
    //   f[0..n-1] í engri sérstakri röð.
    //   Hámarksfjöldi gilda er f.length.

    // Notkun: FBidrod fb = new FBidrod(k);
    // Fyrir:  k er heiltala > 0.
    // Eftir:  fb er ný forgangsbiðröð með pláss
    //         fyrir k heiltölur.
    public FBidrod( int k )
    {
        ...
    }

    // Notkun: k = fb.size();
    // Eftir:  k er hámarksfjöldi gilda í fb.
    public int size()
    {
```

```
        ...
    }

    // Notkun: k = fb.count();
    // Eftir:  k er fjöldi gilda í fb.
    public int count()
    {
        ...
    }

    // Notkun: x = fb.deleteMax();
    // Fyrir:  fb er ekki tóm.
    // Eftir:  x er stærst gildanna sem var í fb.
    //         Það hefur verið fjarlægt úr fb (það
    //         gætu samt verið önnur jöfn gildi eftir
    //         í fb).
    public int deleteMax()
    {
        ...
    }

    // Notkun: fb.add(x);
    // Fyrir:  fb er ekki full.
    // Eftir:  x hefur verið bætt í fb.
    public void add( int x )
    {
        f[n++] = x;
    }
}
```

14. Skrifið stef sem skilar minnsta prímtöluþætti tölu sem skal vera viðfang stefsins.

    Svar:

```
// Notkun: p = prim(n);
// Fyrir:  n er heiltala > 1.
// Eftir:  p er minnsti prímþáttur n.
int prim( int n )
{
    int i=2;
    while( i!=n )
```

```
    {
        // n hefur engan prímþátt
        // minni en i, i>=2.
        if( n%i==0 ) return i;
        i++;
    }
    return n;
}
```

15. Skrifið forrit sem les strengi frá aðalinntaki og skrifar strengina í minnkandi röð á aðalúttak.

    Svar:

```
class Svar
{
    // Notkun: sort(f,n);
    // Fyrir:  f[0..n-1] er svæði í f
    //         sem inniheldur strengi.
    // Eftir:  Búið er að raða f[0..n-1]
    //         í vaxandi röð.
    static void sort( String[] f, int n )
    {
        for( int i=0 ; i!=n ; i++ )
        {
            // f[0..i-1] inniheldur minnstu strengi
            // úr f[0..n-1] í vaxandi röð.
            int k=i;
            for( int j=i+1 ; j!=n ; j++ )
            {
                // f[0..i-1] inniheldur minnstu strengi
                // úr f[0..n-1] í vaxandi röð.
                // f[k] er minnstur af f[i..j-1].
                // 0 <= i <= k < j <= n.
                if( f[k].compareTo(f[j]) > 0 ) k=j;
            }
            String tmp = f[k];
            f[k] = f[i];
            f[i] = tmp;
        }
    }
```

```
    // Notkun: main(args);
    // Fyrir:  Aðalinntak inniheldur allt að 1000 línur.
    // Eftir:  Búið er að skrifa línurnar í minnkandi röð
    //         á aðalúttak.
    static public void main( String[] args )
    {
        java.io.BufferedReader r =
            new java.io.BufferedReader
                (
                    new java.io.InputStreamReader(System.in)
                );
        int n=0;
        String[] f = new String[1000];
        for(;;)
        {
            // Búið er að lesa núll eða fleiri línur og þær eru
            // í f[0..n-1].
            String line = r.readLine();
            if( line==null ) break;
            f[n++] = line;
        }
        sort(f,n);
        int i=n;
        while( i!=0 )
        {
            // Búið er að skrifa núll eða fleiri af stærstu
            // lesnu línum í minnkandi röð á aðalúttak.
            // Hinar eru í f[0..i-1] í vaxandi röð.
            System.out.println(f[--i]);
        }
    }
}
```

16. Skrifið stef sem finnur rót samfellds falls með helmingunarleit.

    Svar:

```
// Notkun: x = rot(a,b,eps);
// Fyrir:  Fallið f er samfellt á [a,b],
//         a < b, f(a)*f(b) <= 0, eps > 0.
// Eftir:  Til er z í [a,b] þ.a. f(z)=0 og
//         |z-x| < eps.
```

```
double rot( double a, double b, double eps )
{
    if( b-a < eps ) return a;
    double m = (a+b)/2.0;
    if( f(a)*f(m) <= 0.0 )
        return rot(a,m,eps);
    else
        return rot(m,b,eps);
}
```

17. Skrifið stef sem finnur lággildi samfellds falls með helmingunarleit.

Svar:

```
// Notkun: x = laggildi(a,b,c,eps);
// Fyrir:  Fallið f er samfellt á [a,b],
//         a < b < c, f(b) <= f(a),f(b), eps > 0.
// Eftir:  Til er z sem er staðbundinn lággildispunktur f
//         í [a,c] þ.a. |z-x| < eps.
double laggildi( double a, double b, double c, double eps )
{
    if( c-a < eps ) return b;
    double m1 = (a+b)/2.0;
    double m2 = (b+c)/2.0;
    if( f(m1) <= f(b) )
        return laggildi(a,m1,b,eps);
    if( f(m2) <= f(b) )
        return laggildi(b,m2,c,eps);
    return laggildi(m1,c,m2,eps);
}
```