1. Output of non-deterministic algorithm may be different for different runs with the same input data
   Mittedetermineeritud algoritmi tulemus samade lähteandmete korral võib erinevatel lahenduskordadel olla erinev.

   Tõene
   Väär

2. Partial algorithm terminates for any set of input data.
   Osaline algoritm peatub mistahes sisendandmete korral.

   Tõene
   Väär

3. Average time complexity of binary search is O(log n).
   Kahendotsimise keskmine ajaline keerukus on O(log n).

   Tõene
   Väär

4. Worst case time complexity of merge sort is O(n).
   Ühildusmeetodi (merge sort) halvima juhu ajaline keerukus on O(n).

   Tõene
   Väär

5. Sorting method is quick if it has average time complexity O(n lon n).
   Järjestamismeetod on kiire, kui selle keskmine ajaline keerukus on O(n log n).

   Tõene
   Väär

6. Last element added to the stack is removed first.
   Magasini (stack) viimati lisatud element eemaldatakse esimesena.

   Tõene
   Väär

7. Set of possible values is a component of an abstract data type.
   Väärtusvaru on üks abstraktse andmetüübi komponent.

   Tõene
   Väär

8. Each element of a doubly linked list contains a pointer to the previous element and a pointer to the next element of the list.
   Topeltseotud ahela iga element sisaldab viita nii eelmisele kui ka järgmisele elemendile.

   Tõene
   Väär

9. Right parenthetic expression becomes Reverse Polish Notation after removing parentheses and commas.
   Avaldise pööratud poola kuju (RPN) saadakse parempoolsest suluesitusest sulgude ja komade ärajätmise teel.

   Tõene
   Väär

10. Full graph is a simple graph.
    Iga täisgraaf on lihtgraaf.

Tõene
Väär

11. Each weakly connected digraph is strongly connected.
Iga nõrgalt sidus graaf on tugevalt sidus.

Tõene
Väär

12. If there is a cycle in a graph it is impossible to find the topological order of vertices.
Kui graafis esineb tsükkel, siis ei saa graafi tippe topoloogiliselt järjestada.

Tõene
Väär

13. It is possible to convert recursion to loops using stack.
Rekursiooni saab magasini abil teisendada tsükliteks.

Tõene
Väär

14. Exhaustive search algorithms tend to have exponential time complexity.
Ammendava otsingu algoritmid on üldjuhul eksponentsiaalse ajalise keerukusega.

Tõene
Väär

15. Smaller height of the binary search tree leads to more effective search.
Mida väiksem on kahendotsimise puu kõrgus, seda efektiivsem on otsimine.

Tõene
Väär

16. It is possible to express the prefix code using code tree.
Koodipuu abil saab kirjeldada prefikskoodi.

Tõene
Väär

17. Set of edges of the null graph is empty.
Nullgraafi servade hulk on tühi.

Tõene
Väär

18. Self-loops are allowed in a simple graph.
Lihtgraafis võivad esineda silmused.

Tõene
Väär

19. If there exists a path from vertex a to vertex b in a graph, then the transitive closure of the graph contains edge (a,b).
Kui graafis leidub tee tipust a tipuni b, siis selle graafi transitiivne sulund sisaldab kaart (a,b).

Tõene
Väär

20. Spanning tree is acyclic.
Toesepuu (spanning tree) on atsükliline.

Tõene
Väär

21. Complexity class of the function $10000n^6+8nlogn+5^n$ is
Funktsiooni $10000n^6+8nlogn+5^n$ keerukusklass on

Vali üks:
n^5
n^6
5^n
nlogn

22. Which of the relationships between functions f and g is defined below
Millist seost funktsioonide f ja g vahel väljendab järgmine definitsioon

$$\exists c > 0,\ \exists n_0 \in N:\ \forall n > n_0\ \ |f(n)| \leq c \cdot |g(n)|$$

Vali üks:
f ~ ω(g)
f ~ O(g)
f ~ Ω(g)
f ~ o(g)
f ~ Θ(g)

23. Which of the relations between functions f and g is defined below
Millist seost funktsioonide f ja g vahel väljendab järgmine definitsioon

$$\forall \epsilon > 0,\ \exists n_0 \in N:\ \forall n > n_0 :\ |f(n)| < \epsilon \cdot |g(n)|$$

Vali üks:
f ~ O(g)
f ~ Θ(g)
f ~ ω(g)
f ~ Ω(g)
f ~ o(g)

24. Average time complexity of the merge sort is
Järjestamise ühildamismeetodi keskmine ajaline keerukus on

Vali üks:
O(n log n)
O (n2)
O(n)
O(log n)
O(1)

25. Average time complexity of simple insertion sort is
Järjestamise lihtsa pistemeetodi keskmine ajaline keerukus on

Vali üks:
O (n2)
O(log n)
O(n log n)
O(1)
O(n)

26. Average time complexity of radix sort is
Järjestamise positsioonismeetodi keskmine ajaline keerukus on

Vali üks:
O (n2)
O(1)
O(n)
O(n log n)
O(log n)

27. Average time complexity of hashtable search is
Paisktabelist otsimise keskmine ajaline keerukus on

Vali üks:
O(n log n)
O(log n)
O(1)
O (n2)
O(n)

28. Average time complexity of binary search is
Kahendotsimise keskmine ajaline keerukus on

Vali üks:
O (n2)
O(log n)
O(n)
O(n log n)
O(1)

29. Average time complexity of heapsort is
Järjestamise kuhjameetodi keskmine ajaline keerukus on

Vali üks:
O(n log n)
O(1)
O (n2)
O(n)
O(log n)

30. Worst case time complexity of quicksort is
Järjestamise kiirmeetodi halvima juhu ajaline keerukus on

Vali üks:
O(n log n)
O (n2)
O(1)
O(n) - Vale
O(log n)

31. Leaves of a tree are
Puu lehed on

Vali üks:
all nodes / kõik puu tipud
nodes without parent / ülemuseta tipud
nodes with children / alluvaid omavad tipud
nodes without children / alluvateta tipud

32. Dijkstra algorithm on graphs is for finding
Dijkstra algoritmiga arvutatakse graafis

Vali üks:

shortest paths from a given vertex to all reachable vertices
antud tipust algavaid lühimaid teid kõigisse saavutatavatesse tippudesse
lengths of shortest paths between all pairs of vertices
lühimate teede pikkusi kõigi tipupaaride vahel
Euler cycle
Euleri tsüklit
Hamilton cycle
Hamiltoni tsüklit
topological order of vertices
tippude topoloogilist järjestust

33. Floyd-Warshall algorithm on graphs is for finding
Floyd-Warshalli algoritmiga arvutatakse graafis

Vali üks:
Hamilton cycle
Hamiltoni tsüklit
lengths of shortest paths between all pairs of vertices
lühimate teede pikkusi kõigi tipupaaride vahel
topoogical order of vertices
tippude topoloogilist järjestust
Euler cycle
Euleri tsüklit
shortest paths from a given vertex to all reachable vertices
antud tipust algavaid lühimaid teid kõigisse saavutatavatesse tippudesse - Vale

34. Kruskal algorithm on graphs is for finding
Kruskali algoritmiga arvutatakse graafis

Vali üks:
Euler cycle
Euleri tsüklit
minimal spanning tree
minimaalset toesepuud
Hamilton cycle
Hamiltoni tsüklit
topological order of vertices
tippude topoloogilist järjestust
lengths of shortest paths between all pairs of vertices
lühimate teede pikkusi kõigi tipupaaride vahel

35. Prim algorithm on graphs is for finding
Primi algoritmiga arvutatakse graafis

Vali üks:
topological order of vertices
tippude topoloogilist järjestust
minimal spanning tree
minimaalset toesepuud
Euler cycle
Euleri tsüklit
Hamilton cycle
Hamiltoni tsüklit
lengths of shortest paths between all pairs of vertices
lühimate teede pikkusi kõigi tipupaaride vahel

36. Which algorithm uses cyclic hash functions for pattern matching
Milline algoritm kasutab tsükliliste räsifunktsioonide arvutamist alamsõne otsimiseks

Vali üks:
Boyer-Moore

Huffman
Rabin-Karp
Shannon-Fano
Knuth-Morris-Pratt

37. Which algorithm uses prefix functions (failure functions) for pattern matching
Milline algoritm kasutab prefiksfunktsiooni arvutamist alamsõne otsimiseks

Vali üks:
Rabin-Karp
Huffman
Knuth-Morris-Pratt
Shannon-Fano
Boyer-Moore

38. Which algorithm uses looking glass heuristic and character jump heuristic for pattern matching
Milline algoritm kasutab sufiksfunktsiooni ja ebasobiva sümboli heuristika arvutamist alamsõne otsimiseks

Vali üks:
Boyer-Moore
Knuth-Morris-Pratt
Huffman
Rabin-Karp
Shannon-Fano

39. Which algorithm uses bisection of sets of symbols to calculate the codes of symbols
Milline algoritm kasutab sümbolihulkade poolitamist sümbolite koodide arvutamiseks

Vali üks:
Shannon-Fano
Rabin-Karp
Boyer-Moore
Knuth-Morris-Pratt
Huffman

40. Which algorithm builds a code tree to calculate the codes of symbols
Milline algoritm kasutab koodipuu moodustamist sümbolite koodide arvutamiseks

Vali üks:
Knuth-Morris-Pratt
Shannon-Fano
Huffman
Boyer-Moore
Rabin-Karp

41. If recursive call is the last command in an algorithm it is called
Kui rekursiivne pöördumine on algoritmi viimane käsk, siis on see

Vali üks:
tail recursion
sabarekursioon
nose recursion
ninarekursioon
head recursion
pearekursioon
indirect recursion
kaudne rekursioon
paw recursion
käparekursioon

42. Returning to the choice made earlier and choosing an unused path in exhaustive search algorithms is known as
Tagasipöördumist varem kõrvale jäetud lahendusvariandi juurde ammendava otsingu ülesannetes nim. inglise keeles:

Vali üks:
greedy choice
sorting
backtracking
dynamic programming
searching

43. Problem solution method that uses pre-calculated answers to sub-tasks is known as
Alamülesannete vastuste meeldejätmisel põhinevat iteratiivset lahendusmeetodit nim. inglise keeles

Vali üks:
sorting
returning - Vale
backtracking
searching
dynamic programming

44. Kruskal algorithm and Huffman algorithm are examples of
Kruskali algoritm (samuti Huffmani algoritm) on

Vali üks:
divide and conquer algorithm / jaga ja valitse algoritm
exhaustive search algorithm / ammendava otsingu algoritm - Vale
quicksort algorithm / kiirsorteerimise algoritm
greedy algorithm / ahne algoritm
dynamic programming algorithm / dünaamilise kavandamise algoritm

45. Calculating the Fibonacci sequence and finding the longest common subsequence (LCS) of two sequences are examples of
Pikima ühise osasõne leidmine (samuti Fibonacci jada moodustamine) on

Vali üks:
greedy algorithm / ahne algoritm
dynamic programming algorithm / dünaamilise kavandamise algoritm
exhaustive search algorithm / ammendava otsingu algoritm
divide and conquer algorithm / jaga ja valitse algoritm - Vale
quicksort algorithm / kiirsorteerimise algoritm

46. Eight queens problem and knapsack problem are examples of
Lippude paigutamine malelauale, samuti seljakotiülesanne on

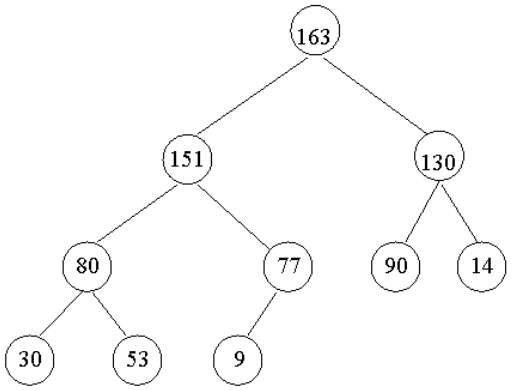
Vali üks:
divide and conquer algorithm / jaga ja valitse algoritm - Vale
greedy algorithm / ahne algoritm
exhaustive search algorithm / ammendava otsingu algoritm
quicksort algorithm / kiirsorteerimise algoritm
dynamic programming algorithm / dünaamilise kavandamise algoritm

47. Quicksort and merge sort are examples of
Järjestamise kiirmeetod, samuti ühildamismeetod on

Vali üks:
divide and conquer algorithm / jaga ja valitse algoritm
dynamic programming algorithm / dünaamilise kavandamise algoritm
exhaustive search algorithm / ammendava otsingu algoritm - Vale

backtracking algorithm / tagasivõtu (tagurdamise) algoritm
greedy algorithm / ahne algoritm

48. Which data structure is on the picture
Millise andmestruktuuriga on tegemist



Vali üks:
stack / magasin (pinu)
binary heap / kahendkuhi
binary search tree / kahendotsimise puu
B-tree / B-puu
queue / järjekord

49. Which order of nodes of a binary tree is generated by the following algorithm:
1) process the root node;
2) apply this algorithm to the left subtree;
3) apply this algorithm to the right subtree.

Milline tippude järjestus saadakse läbides kahendpuud algoritmiga:
1) töödelda juur;
2) läbida vasak alampuu;
3) läbida parem alampuu.

Vali üks:
post-order / tagajärjestus
in-order / keskjärjestus - Vale
end-order / lõppjärjestus
pre-order / eesjärjestus

50. Which property is described as: all keys in left subtree are not greater than the key of the root node and all keys in right subtree are not less than the key of the root node.
Millist omadust kirjeldab lause: kõik võtmed vasakus alampuus ei ole suuremad juure võtmest ning kõik võtmed paremas alampuus ei ole väiksemad juure võtmest.

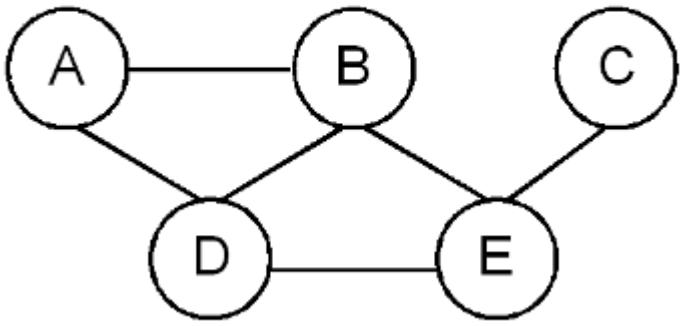
Vali üks:
property of keys in heap / kuhjaomadus - Vale
property of keys in binary search tree / kahendotsimise puu võtmete omadus
property of keys in reverse heap / pöördkuhja omadus
property of keys in binomial tree / binomiaalpuu võtmete omadus

51. Match notations with asymptotic properties. f and g are functions.
Leia vastavus tähistuste ja tähenduste vahel, f ja g on funktsioonid, mille asümptootilist käitumist võrreldakse.

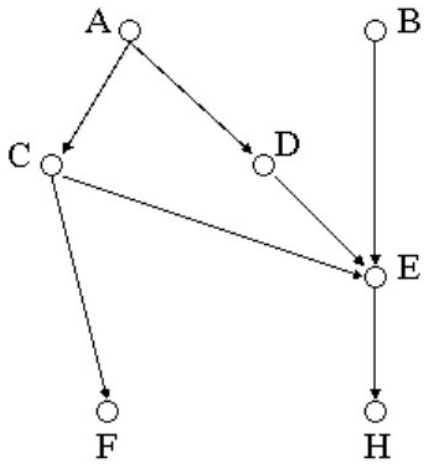
f~ω(g)	f grows faster than g / f kasvab kiiremini kui g ✓
f~Ω(g)	f grows not slower than g / f kasvab mitte aeglasemalt kui g ✓
f~Θ(g)	f and g grow equally fast / f kasvab niisama kiiresti kui g ✓
f~O(g)	f grows not faster than g / f kasvab mitte kiiremini kui g ✓
f~o(g)	f grows slower than g / f kasvab aeglasemalt kui g ✓

52. Fill in the blanks with numbers to express the matrix of shortest pathlengths for the following graph. Each edge is of length 1.
Moodusta selle graafi lühimate teepikkuste maatriks täites lüngad arvudega (iga serva pikkus on üks ühik):

	A	B	C	D	E
A	0 ✓	1 ✓	3 ✓	1 ✓	2 ✓
B	1 ✓	0 ✓	2 ✓	1 ✓	1 ✓
C	3 ✓	2 ✓	0 ✓	2 ✓	1 ✓
D	1 ✓	1 ✓	2 ✓	0 ✓	1 ✓
E	2 ✓	1 ✓	1 ✓	1 ✓	0 ✓

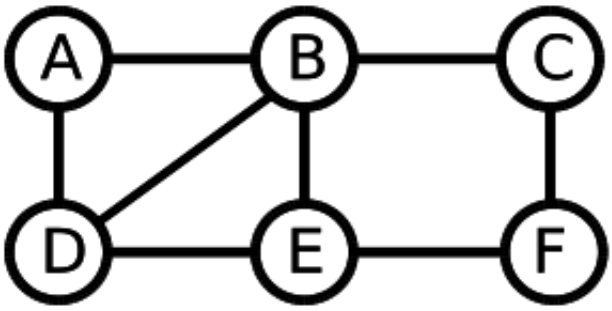


53. Choose three sequences that express topological order of vertices of the following graph:
Leia antud järjestuste hulgast kolm, mis sobivad selle graafi tippude topoloogiliseks järjestuseks.



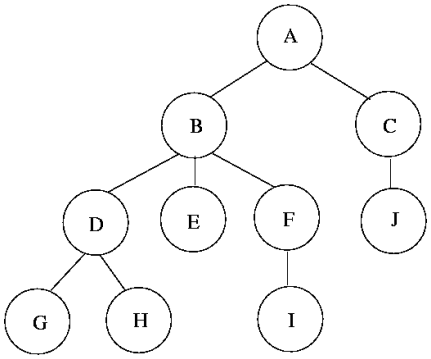
Vali üks või enam:
ACFDBEH
ABEDCFH
ACDEFBH
BACFDEH
ADBCEHF
FHECDAB

54. Choose three sequences that match the breadth first traversal order of the following graph starting from vertex A.
Leia allpool kolm järjestust, mis vastavad selle graafi tippude laiuti läbimise strateegiale alates tipust A.



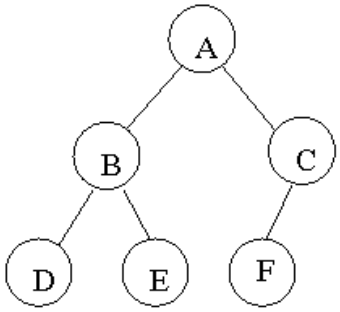
Vali üks või enam:
ABDECF
ADBCEF
ABDCEF
ADBEFC
ABCDEF
ABDCFE

55. Find the post-order sequence of nodes of the following tree and write it as a string without spaces.
Leia selle puu tippude lõppjärjestus ning esita see ilma tühikuteta sõnena.



Vastus: **GHDEIFBJCA**

56. Find the in-order sequence of nodes of the following tree and write it as a string without spaces.
Leia selle kahendpuu tippude keskjärjestus ning esita see ilma tühikuteta sõnena



Vastus: **DBEFCA**

57. Find the reverse polish notation (RPN) of the following expression and write it as a string where elements are separated by one space exactly.
Leia selle avaldise pööratud poola kuju ning esita see sõnena, milles avaldise elemendid on eraldatud täpselt ühe tühikuga:

4 /(5-3) + 2*6

Vastus: **4 5 3 - / 2 6 * +**

58. Which of the following is the weakest precondition to the program:
Milline loetletutest on alljärgneva programmi nõrgim eeltingimus:

```
a := a + 1;
b := a + 2;
```

if postcondition is:
kui järeltingimus on:

{ b == 6 }

Vali üks:
{ b == 6 }
{ a == 5 }
{ a == 4 }
{ a == 3 }

59. Write a Java method maxWidth() in class Answer to calculate the width of the tree with root this, where width is defined as the greatest number of children under one node in this tree. Do not forget to test the tree that consists of root node only (width 0).

Koostage klassi Answer isendimeetod maxWidth(), mis leiaks puu juurega this "laiuse", kus puu laius on defineeritud kui suurim ühe tipu vahetute alluvate arv antud puus. Testige muuhulgas ka ühetipuline puu (laius 0).

public int maxWidth()

Vastus:

```
import java.util.*;

public class Answer {

   private String   name;
   private Answer   firstChild;
   private Answer   nextSibling;

   Answer (String n, Answer d, Answer r) {
      setName (n);
      setFirstChild (d);
      setNextSibling (r);
   }

   public void   setName (String n)        { name = n; }
   public String getName()                 { return name; }
   public void   setFirstChild (Answer d)  { firstChild = d; }
   public Answer getFirstChild()           { return firstChild; }
   public void   setNextSibling (Answer r) { nextSibling = r; }
   public Answer getNextSibling()          { return nextSibling; }

   @Override
   public String toString() {
      return leftParentheticRepresentation();
   }

   public String leftParentheticRepresentation() {
      StringBuffer b = new StringBuffer();
      b.append (getName());
      if (getFirstChild() != null) {
         b.append ("(");
         b.append (getFirstChild().leftParentheticRepresentation());
         Answer right = getFirstChild().getNextSibling();
         while (right != null) {
            b.append ("," + right.leftParentheticRepresentation());
            right = right.getNextSibling();
         }
         b.append (")");
      }
      return b.toString();
   }

   public static Answer parseTree (String s) {
      if (s == null) return null;
      if (s.length() == 0) return null;
      Answer root = null;
      Answer curr = null;
      Answer last = null;
      int state = 0; // begin
      Stack<Answer> stk = new Stack<Answer>();
      StringTokenizer tok = new StringTokenizer (s, "(),", true);
      while (tok.hasMoreTokens()) {
         String w = tok.nextToken().trim();
```

```
      if (w.equals ("(")) {
         state = 1; // from up
      } else if (w.equals (",")) {
         state = 2; // from left
      } else if (w.equals (")")) {
         state = 3; // from down
         stk.pop();
      } else {
         curr = new Answer (w, null, null);
         switch (state) {
            case 0: {
               root = curr;
               break;
            }
            case 1: {
               last = stk.peek();
               last.setFirstChild (curr);
               break;
            }
            case 2: {
               last = stk.pop();
               last.setNextSibling (curr);
               break;
            }
            default: {
               // do not pop here but after ")"
            }
         } // switch
         stk.push (curr);
      }
   } // next w
   return root;
}

private List<Integer> c = new LinkedList();

private void calc(Answer a) {

            int result = 1;
            if (a.firstChild == null) return;

            Answer answer = a.firstChild;
            while (answer.nextSibling != null){

                  answer = answer.nextSibling;
                  if (answer != null) calc(answer);
                  result++;
            }

      c.add(result);
}

   public int maxWidth() {

         if (this.firstChild == null) return 0;
         calc(this);

      return Collections.max(c);
   }
```

```
   public static void main (String[] param) {
      Answer v = Answer.parseTree ("A(B,C(D,F(K,L,M,N(O)),P))");
      System.out.println (v);
      int n = v.maxWidth();
      System.out.println ("Maximum number of children: " + n); // 4

      v = Answer.parseTree("A(B(C))");
      System.out.println (v);
      n = v.maxWidth();
      System.out.println ("Maximum number of children: " + n); // 1
   }
}
```