Exam: TDDD86

Data Structures, Algorithms and Programming Paradigms

2026-03-17 kl: 08-12

On-call (jour): Ahmed Rezine (tel: 1938)

Specific instructions for the computer exams:

- In summary: you log in with your LiU-ID and your private password. You can only save files in the desktop. We might leave files for you in the read-only "given_files" folder (e.g., lecture slides). **You will use the "student chat client", or "student client" to receive information during the exam, to ask questions and to submit your solution.** More details in the "EXAM_README.pdf" under "given_files".

- Your "student client" should start automatically, if you close it and need to start it again, double click on the "fish icon" on your desktop.

- Submit one file with all your answers. The document should only contain text with a .txt suffix (e.g., answers.txt). The document should not contain drawings or pictures. We will only look at the last submitted file.

- The questions are formulated so that you can answer with any text editor (e.g., vi, emacs, gedit, etc).

- You can access OpenDSA using chromium. The start page will list available links.

General instructions:

- You may answer in either English or Swedish.

- If in doubt about a question, write down your interpretation and assumptions.

- The exam is divided into two parts:
  - Part A with a maximum of 34 pts.
  - Part B with a maximum of 20 pts.

- Grading:
  - Grade 3 requires at least 20 pts exclusively from Part A.
  - Grade 4 requires grade 3 is secured and at least 8 pts from Part B.
  - Grade 5 requires grade 3 is secured and at least 12 pts from Part B.

Part A

Problem A.1: Asymptotic execution time (min 0 pts, max 10 pts)

Consider the five methods `f1, f2, f3, f4, f5` and the nine complexity classes (A)-(I) depicted below. Assume the manipulated arrays are large enough in the five methods. The asymptotic analysis is to be carried out with respect to $n = hi - lo$ for each one of the five methods. If it simplifies your reasoning when analyzing the asymptotic time complexity of the methods, you can restrict the analysis to sizes of the form $n = 2^p$ or $n = 2^p - 1$ for some natural number p.

```
void f1(int a[], int lo, int hi ){
  for(int i = lo + 1; i < hi; i++){
    int j = i;
    int x = a[i];
    while ((j > lo) && (a[j-1] < x)){
      a[j] = a[j-1];
      j = j-1;
    }
    a[j] = x;
  }
}
```

```
int f2(int a[], int lo, int hi){
   for(int i = lo; i <= hi; i++){
      for(int j = i + 1; j <= i + 100; j++){
         if(a[i] == a[j]){
            return 1;
         }
      }
   }
   return 0;
}
```

```
int f3(int a[], int lo, int hi){
  for(int i= lo + 1; i < hi; i++){
    if(a[i-1] < a[i]){
      return 0;
    }
  }
  return 1;
}
```

```
int f4(int count, int a[], int lo, int hi){
   for(int j = lo; j <= hi; j++){
     if(!f4(count + a[lo], a, lo + 1, hi)){
        return f4(count, a, lo + 1, hi);
     }
  }
     return (count == 0);
}
```

```
int f5(int x, int a[], int lo, int hi){
  if(lo == hi) {
     return (a[lo] < 0);
  }
  int m = lo + (hi - lo)/2;
  if (f5(x, a, lo, m))
      return 1;
  return f5(x, a, m + 1, hi);
}
```

Complexity classes:

(A) $\Theta(1)$	(D) $\Theta(n \log n)$	(G) $\Theta(2^n)$
(B) $\Theta(\log n)$	(E) $\Theta(n^2)$	(H) $\Theta(3^n)$
(C) $\Theta(n)$	(F) $\Theta(n^3)$	(I) $\Theta(n!)$

1. For each one of the 5 methods above, give (without justification!) the complexity class among the classes (A-I) that best matches its asymptotic **worst-case** execution time. (For each method, 1pts if correct, 0 if not answered, -1pts if incorrect.)

2. For each one of the 5 methods above, give (without justification!) the complexity class among the classes (A-I) that best matches its asymptotic **best-case** execution time. (For each method, 1pts if correct, 0 if not answered, -1pts if incorrect.)

Problem A.2: Hashing and conflict resolution (min 0 pts, max 10 pts)

Consider the 5 elements sequence seq = < 54, 13, 12, 23, 33 >

We will assume closed hashing and use the hash function $h(k) = k$ % 10 to compute the "home" (or "default") slot of key k. We consider three different collision resolution approaches.

For each approach, we will insert all elements of seq (one after the other) starting from an empty hash table "T". Indices of the table range from 0 to 9 (i.e., the table is of capacity 10). Initially all cells T[0], … T[9] are marked empty.

Give the final position (as a index between 0 and 9) of the element 33 in the table T after adding all elements of seq according to the above description:

3. Case 1: What is the final position of 33 if we use linear probing with probe function $p(k, i) = i$. (2pts if correct, 0 if not answered, -2 if incorrect).

4. Case 2: What is the final position of 33 if we use linear probing with probe function $p(k, i) = 7i$. (2pts if correct, 0 if not answered, -2 if incorrect).

5. Case 3: What is the final position of 33 if we use quadratic probing with probe function $p(k, i) = (i + 1)^2$. (2pts if correct, 0 if not answered, -2 if incorrect).

6. Recall hashing can suffer from primary and secondary clustering during collusion resolution. Explain each one of the two clusterings and state, for each one of the three cases above, if the case may suffer from primary clustering, secondary clustering or both. (4pts if correct, 0 if not answered or partly incorrect).

Problem A3. Binary search trees (min 0 pts, max 4 pts)

Recall that binary trees can be represented sequentially. We adopt the approach described in 8.3.1 in OpenDSA. For instance, the binary tree in Figure 1 can be sequentially represented using the sequence: "A B / D // C E G /// F H // I //". The symbol "/" is used to represent a "null" child. Do not draw your trees! Use this approach instead.

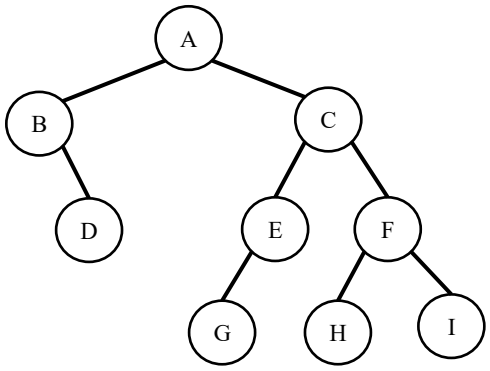

Figure 1. A B / D // C E G /// F H // I //

Consider the sequence of 15 elements:

$$S=<8, 15, 9, 1, 7, 4, 6, 12, 13, 3, 2, 10, 11, 14, 5>$$

7. Give a sequential representation (see the description of sequential representations of binary trees at the beginning of this problem) of the final binary search tree obtained by starting from an empty tree and inserting all the integers of the sequence S one after the other (i.e., first insert 8, then 15, then 9 …). Do not try to balance the tree. Call this tree T_1. (2pts if correct, 0 if not answered, -2pts if incorrect).

8. Give a sequential representation (see the description of sequential representations of binary trees at the beginning of this problem) of the binary search tree obtained by removing the node with value 4 from tree T_1 (observe T_1 is the tree you obtain in question A3.7). Write your assumptions in case you make choices. (2pts if correct, 0 if not answered, -2pts if incorrect).

Problem A4. Complete Binary Trees and Heaps (min 0 pts, max 4pts)

Consider the set of 7 integers H= {1, 2, 3, 4, 5, 6, 7}. Recall that priority queues can be efficiently implemented using complete binary trees. This results in so called "heaps". We consider **max-heaps** in the remaining part of this problem.

9. Recall that a complete binary tree capturing a max-heap can be represented using an array of the same size as the number of elements in the max-heap. You used such a representation for a min-heap in lab 6. There might be several different

complete binary trees representing a max-priority queue containing the elements in the set H above. Choose one such tree and give the sequence of elements appearing in its array representation. Just give the sequence of elements in the array representation (no need to give the tree). Your sequence should contain |H|=7 elements appearing in increasing order of their indices in the array representation. (2pts if correct, 0 if not answered, -2 if incorrect).

10. Assume the max-heap (together with its complete binary tree and array representation) you chose in the previous question. Give the sequence of elements appearing in the array representation of the max-heap obtained by inserting the element 8 to the max-heap. (2pts if correct, 0 if not answered, -2 if incorrect).

Problem A.5: Graphs (min 0 pts, max 6 pts)

11. Does the graph depicted in Figure 2 admit a topological sort of its nodes? If your answer is yes, give such a topological sort, otherwise, argue why it does not admit a topological sort. (2pts if correct, 0 if not answered, -2 if incorrect).

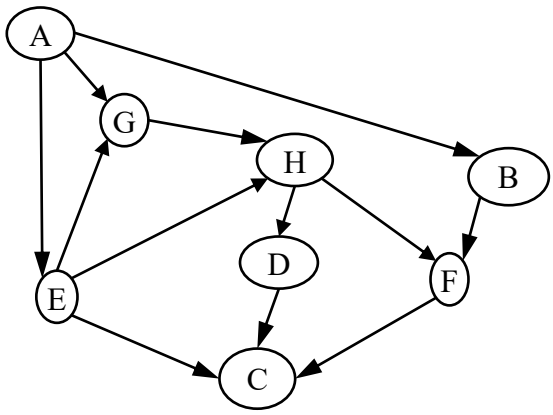

Figure 2. Directed graph for question 12

12. Consider the graph in Figure 3. It represents eight places from A to H together with amounts of energy in Watt hour (Wh) to get from one place to the other. Observe the edges are directed and weighted. We will identify the edges with the nodes at their extremities. For instance, the edge A-B captures that you can get directly from place A to place B using 2850 Wh. There is no edge B-A (since the edges are directed).

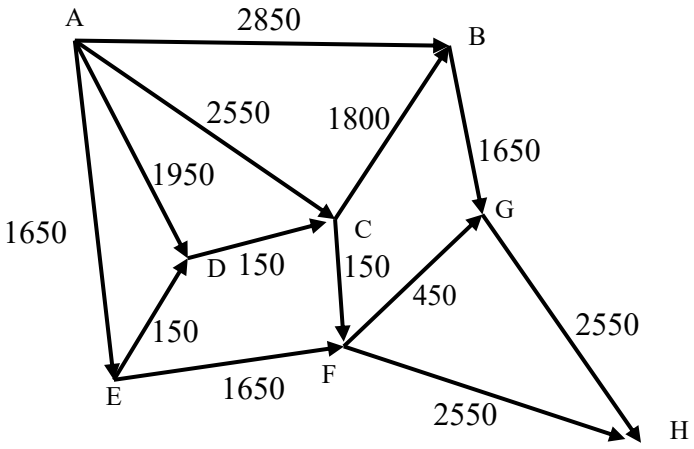

Figure 3: Energy consumption in Wh for an electrical car for question 13

Dijkstra's algorithm can be used to find paths with minimal costs (here energy) from a single source to all other nodes in a weighted graph. Assume the single source is A in Figure 3. The algorithm manipulates a priority queue. Initially, the priority queue only contains the pair (A, 0). At each iteration of the loop, the algorithm picks (and removes) an element from the priority queue and might add elements to the priority queue and/or update values associated to elements already in the priority queue. Use alphabetical order to break ties when picking from the queue. Give the pairs contained in the priority queue at the beginning of each iteration of Dijkstra's algorithm. For instance, at the beginning of the first iteration, the priority queue only contains the pair (A, 0). For each priority queue, just give the pairs (node, value) in the priority queue. The algorithm terminates when the priority queue is empty. You do not have to represent the priority queue as a heap as in questions 9-10 above, just give the pairs contained in the queue at the start of each iteration. (4pts).

Part B:

Problem B.1 (minimum 0pts, maximum 8 pts)

We adopt the same approach as the one used in problem A3 (described in Figure 1) above to sequentially represent binary trees.

AVL trees. Recall the height of a binary tree is the length of the longest paths from a root to a leaf. To avoid confusion, we will define the length of a path containing n nodes, for $n > 0$, to be $(n - 1)$. Hence, the height of a binary tree containing a single node is 0.

13. Can you have an AVL tree of height 3 containing all the 7 elements of the set {3, 5, 7, 11, 13, 17, 19}? If yes give a sequential representation of such an AVL tree, clearly justify otherwise. (2pts if correct, 0 if not answered, -2pts if incorrect).

14. Can you have an AVL tree of height 4 containing all the 12 elements of the set {3,

5, 7, 11, 13, 17, 19, 23, 29, 31, 37}? If yes give a sequential representation of such an AVL tree, clearly justify otherwise. (3pts if correct, 0 if not answered, -3pts if incorrect).

15. Can you obtain an AVL tree of height 5 using all the 19 elements of the set {3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 39, 41, 43, 47, 51, 53, 59}? If yes give a sequential representation of such a tree, clearly justify otherwise. (3pts if correct, 0 if not answered, -3pts if incorrect).

Problem B.2 (min 0pts, max 8pts)

Answer with yes or no (no need for justification). For each answer, 1pts if correct, 0 if not answered, -1pts if incorrect:

16. Assume the best-case time complexity of an algorithm is in $\Omega(n^3)$. Does this mean it is impossible to have a family of inputs, one input for each size n, on which each input of size n uses $t(n)$ steps with t(n) belongs to O(n)?

17. Assume the worst-case time complexity of an algorithm is in $\Theta(n^3)$. Does this mean it is impossible to have a family of inputs, one input for each size n, on which the algorithm takes $t(n)$ steps with t(n) belongs to $\Omega(n^3 \log(n))$?

18. Assume the best-case time complexity of an algorithm is in $\Theta(n^3)$. Does this mean it is impossible to have a family of inputs, one input for each possible size n, on which the algorithm takes $n/2$ steps?

19. Assume the worst-case time complexity of an algorithm is in $\Theta(n^3)$. Does this mean it is impossible to have a family of inputs, one input for each size n, on which the algorithm takes $t(n)$ steps with $t(n)$ belongs to $\Omega(n)$?

20. Assume the worst-case time complexity of an algorithm is in $O(n^3)$. Does this mean it is impossible to have a family of inputs, one input for each size n, on which the algorithm takes 100 steps?

21. Assume the worst-case time complexity of an algorithm is in $\Omega(n^3)$. Does this mean it is impossible that all inputs, for any size n, use $n/2$ steps?

22. Assume the best-case time complexity of an algorithm is in $\Omega(n^3)$. Does this mean it is impossible that all inputs, for any size n, require $t(n)$ steps with t(n) belongs to $\Omega(n)$?

23. Assume the best-case time complexity of an algorithm is in $\Omega(n^3)$. Does this mean it is impossible that all inputs, for any input size n, require $t(n)$ steps with t(n) belongs to O(n)?

Problem B.3 (4 pts):

Consider the method `foo` listed at the end of this problem.

24. Let **foo_steps(n)** be the number of steps executed in the worst case by a call to `foo` when the argument equals **n**. State a recurrence definition of **foo_steps(n)** in terms of **foo_steps(n-1)** and **foo_steps(n-2)**. (1pts).

25. Observe **foo_steps(n-1)** monotonically increases. Give, and explain how you can derive an exponential asymptotic lower bound **f(n)** for **foo_steps(n)**. (3pts).

```
unsigned int foo(unsigned int n){
if (n < 2)
    return n;
  return foo(n - 1) + foo(n-2);
}
```