Exam: TDDD86

Data Structures, Algorithms and Programming Paradigms

2024-12-19 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 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 $n = 2^p - 1$.

```
int f1(int a[], int lo, int hi){
  for(int i= lo + 1; i < hi; i++){
    if(a[i-1] > a[i]){
      return 0;
    }
  }
  return 1;
}
```

```
void f2(int a[], int lo, int hi){
  for(int i = lo; i < hi-1; i++){
    int s = i;
    int j = i + 1;
    while (j < hi){
      if(a[j] < a[s]){
        s = j;
      }
      j = j + 1;
    }
    int x = a[i];
    a[i] = a[s];
    a[s] = x;
  }
}
```

```
void f3(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 f4(int a[], int lo, int hi, int k){
  while(lo < hi){
    int m = lo + (hi - lo)/2;
    if(a[m] < k){
      lo = m + 1;
    }else if(a[m] > k){
      hi = m;
    }else{
      return m;
    }
  }
  return -1;
}
```

```
int f5(int a[], int lo, int hi, int lv){
  if(lo >= hi){
    return -1;
  }
  for(int i = lo; i < hi; i++){
    if(a[i] == lv){
      return lv;
    }
  }
  int m = lo + (hi - lo)/2;
  int v = f5(a, lo, m, 2*lv);
  if(v == -1){
    return f5(a, m+1, hi, 2*lv);
  }
  return v;
}
```

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 4 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 4 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 8 pts)

Assume linear probing is used (i.e., the probe function is $p(k, i) = i$). In addition, assume we use an array of size 7 with indices 0 to 6. The array is used to implement a hash table where the hash function is given by the following:

Key	Hash value
A	2
B	3
C	4
D	1
E	3
F	5
G	4

In other words, the "home position" of key A is 2 and keys C and G hash both to 4.

3. Give the content of each cell of the table after inserting, starting from an empty table, the sequence C, G, F, E, D, A, B (i.e., inserting first C, then G, then F … and finally B). (2pts if correct, 0 if not answered, -2pts if incorrect).

4. Answer with yes or no (no need for justification). Is there a sequence that results, starting from an empty table, in this table? (2pts if correct, 0 if not answered, -2pts if incorrect).

Index	0	1	2	3	4	5	6
Content	E	F	G	A	B	C	D

5. Answer with yes or no (no need for justification). Is there a sequence that results, starting from an empty table, in this table? (2pts if correct, 0 if not answered, -2pts if incorrect).

Index	0	1	2	3	4	5	6
Content	B	D	A	E	C	G	F

6. Answer with yes or no (no need for justification). Is there a sequence that results, starting from an empty table, in this table? (2pts if correct, 0 if not answered, -2pts if incorrect).

Index	0	1	2	3	4	5	6
Content	G	C	A	D	B	E	F

Problem A3. Binary search trees (min 0 pts, max 8 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.

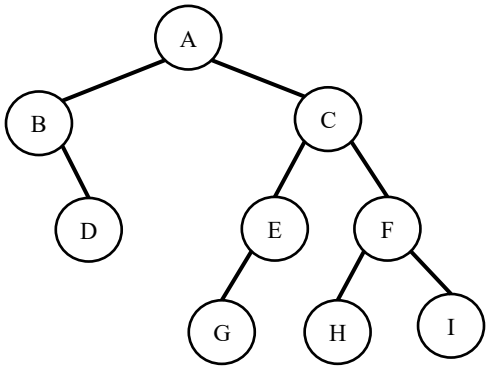

Figure 1. A B / D // C E G /// F H // I //

Consider the sequence of 15 elements:

S: 375, 33, 73, 619, 213, 787, 378, 596, 469, 574, 550, 12, 589, 534, 635

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 375, then 33, then 73 …). Do not try to balance the tree. Call this tree T_1. (2pts if correct, 0 if not answered, -2pts if incorrect).

8. The ordered binary tree obtained in the question above is not perfect. The shape of the tree (e.g., whether it is perfect or not) depends on the order in which the elements are inserted. Give another sequence containing the same 15 elements (but in a different order) that results (when inserting, starting from an empty tree, the elements according to the order given by your proposed sequence) in a perfect binary search tree. (2pts if correct, 0 if not answered, -2pts if incorrect).

9. List the integer values encountered in a **post-order** traversal of T_1 (observe T_1 is the tree you obtain in question A3.7). The question is just about the integer values encountered in a post-order traversal of T_1 (NOT about the sequential representation described in the beginning of the problem) (2pts if correct, 0 if not answered, -2pts if incorrect).

10. Give a sequential representation (see the description of sequential representations of binary trees at the beginning of this problem) of the tree obtained by removing the root node 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. Sorting (min 0 pts, max 4pts)

Consider the array `arr` containing the 15 integers:

arr = [375, 33, 73, 619, 213, 787, 378, 596, 469, 574, 550, 12, 589, 534, 635]

with `arr[0]=375, arr[1]=33, … arr[14]=635.` Consider the quicksort algorithm and suppose we use it to sort the array `arr` above by calling `quicksort(arr,0,14)`. Suppose our implementation of quicksort follows the one described in chapter 11.11 of OpenDSA but where "**int** pivotindex = findpivot(i, j);" randomly returns an index in [i,j] (i.e., findpivot might return any index in {i, i+1, i+2, …j}). Notice that a recursive call on a remaining sequence in the described quicksort algorithm of chapter 11.11 of OpenDSA is only made if the remaining sequence has 2 or more elements (i.e., there are no recursive calls for sequences of one element or less).

Observe that there is a run of the algorithm that results in 7 calls to quicksort, namely the original call `quicksort(arr,0,14)` followed by those resulting from choosing the sequence 33, 213, 378, 534, 574, 596, and 635 as successive pivots. A different choice of

pivots, and the order in which they are chosen, can result in a different number of calls to quicksort despite starting with `quicksort(arr,0,14).`

11. Give a sequence of pivot elements that results, for the array `arr,` in a largest number of calls to `quicksort`. The answer to this question is a sequence of some of the elements in `arr.` (2pts if correct, 0 if not answered, -2pts if incorrect.)

12. Give a sequence of pivot elements that results, for the array `arr,` in a smallest number of calls to `quicksort`. The answer to this question is a sequence of some of the elements in `arr.` (2pts if correct, 0 if not answered, -2pts if incorrect).

Problem A.5: Graphs (min 0 pts, max 4 pts)

13. Give TWO different topological sorts of the directed graph depicted in Figure 2. (2pts if two correct and different sorts, 0 if not answered, -2pts otherwise)

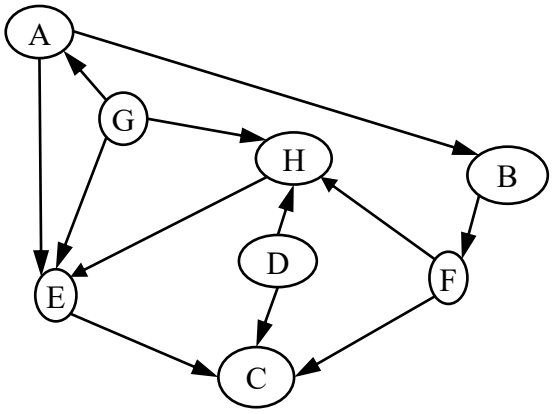

Figure 2. Directed graph for the topological sorting question A5.13

14. If there are strongly connected components in the graph of Figure 3, give the nodes of a maximal (i.e., cannot be extended) strongly connected component, otherwise state there are no strongly connected components. (2pts if correct, 0 if not answered, -2pts if incorrect).

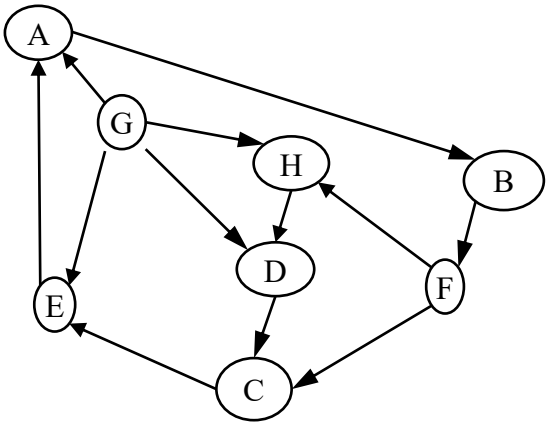

Figure 3. Directed graph for the strongly connected components question A5.14

Part B:

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:

15. Assume the worst-case time complexity of an algorithm is in $O(n^2)$. Does this contradict the existence of a family of inputs, one input for each size n, on which the algorithm takes 5 steps?

16. Assume the worst-case time complexity of an algorithm is in $O(\mathrm{n}^2)$. Does this contradict the possibility that, for any size n, each input of size n requires 5 steps?

17. Assume the worst-case time complexity of an algorithm is in $\Theta(\mathrm{n}^2)$. Does this contradict the existence of a family of inputs, one input for each size n, on which the algorithm takes $t(n)$ steps and where the only fact we know about $t(n)$ is that it belongs to $\Omega(n)$?

18. Assume the worst-case time complexity of an algorithm is in $\Theta(\mathrm{n}^2)$. Does this contradict the existence of a family of inputs, one input for each size n, on which the algorithm takes $t(n)$ steps and where the only fact we know about t(n) is that it belongs to $\Omega(n^2 \log(n))$?

19. Assume the best-case time complexity of an algorithm is in $\Theta(\mathrm{n}^2)$. Does this contradict the existence of a family of inputs, one input for each possible size n, on which the algorithm takes 5n+3 steps?

20. Assume the best-case time complexity of an algorithm is in $\Omega(\mathrm{n}^2)$. Does this contradict the existence of a family of inputs, one input for each size n, on which each input of size n requires $t(n)$ steps and where the only fact we know about t(n) is that it belongs to $O(\mathrm{n})$?

21. Assume the best-case time complexity of an algorithm is in $\Omega(\mathrm{n}^2)$. Does this contradict the possibility that for any possible size n, all inputs of size n require $t(n)$ steps, where the only fact we know about t(n) is that it belongs to $O(\mathrm{n})$?

22. Assume the best-case time complexity of an algorithm is in $\Omega(\mathrm{n}^2)$. Does this contradict the possibility that, for any possible size n, all inputs of size n require $t(n)$ steps where the only fact we know about t(n) is that it belongs to $\Omega(n)$?

Problem B.2 (12 pts):

This problem has two parts: Part I and Part II. Both parts manipulate **`list<T>`** objects, where **`list<T>`** is the doubly linked container defined in the standard library. In your answers, the only allowed **list<T>** methods you may use are those already used in Figure 5. Of course, your code might be different, but you should not use **`list<T>`** methods other than those used in Figure 5. You can assume the following operations to have a constant worst-case time complexity (i.e., worst-case complexity in $\mathcal{O}(1)$):

- "list<int> rslt", "lst.size()"
- "lst1.begin()", "lst1.end()", "it=lst1.begin()", "it1 != it2"
- "rslt.push_back(5)" "(*it1) < (*it2)"
- "return rslt" where rslt is a list<int>

Part I.

The code listed in Figure 5 has a **`main`** method calling a method **`foo`** on a list of integers **`list<int>`**:

23. Give a tight bound $g(n_1, n_2)$ for the worst-case time complexity of the method **`bar(lst1, lst2)`** as a function of the length n_1 of list **`lst1`** and the length n_2 of list **`lst2`**. In other words, give a function $g(n_1, n_2)$ such that the number of steps needed by **`bar(lst1, lst2)`** in the worst case is in $\Theta\big(g(n_1, n_2)\big)$. Justify. (2pts).

24. Give a tight bound f(n) for the worst-case time complexity of the method **`foo(lst)`** as a function of the length n of list **`lst`**. In other words, give a function f(n) such that the number of steps needed by **`foo(lst)`** in the worst case is in $\Theta(f(n))$. Justify (you can assume n is a power of 2). (3pts).

Part II.

We consider the problem of checking, given a list of integers **`lst,`** whether **lst** contains three elements **`a, b`** and **`c`** such that **`a + b + c = 0`**. You can assume the elements in the list **`lst`** are pairwise different and the length of **`lst`** is a power of 2.

25. Propose a "C++ like pseudo code"[1] for a method:

```
bool check (const list<int>& lst)
```

that returns true if and only if the list **`lst`** contains three elements that sum to 0. The method should have $\Theta(n^3)$ worst-case time complexity, where n is the length of the list **`lst`**. Argue why your solution is correct (i.e., why does it solve the problem at hand) and why its worst-case is in $\Theta(n^3)$. (2pts).

[1] Similar to the code in Figure 5, "C++ like pseudo code" should clarify the steps involved in your method. It should be possible to compile with minor changes. Syntax errors such as missing semi-colons are not a problem.

26. Propose a "C++ like pseudo code" for a method:

`bool fasterCheck (const list<int>& lst)`

The method should solve the same problem as above. It should not use hashing. In addition, its worst-case time complexity should be in $\Theta(h(n))$ for some $h(n)$ in $O(n^3)$ but with n^3 not in $O(h(n))$ (intuitively, **`fasterCheck`** solves the same problem as **`check`** but is asymptotically strictly more efficient when it comes to the worst cases). Argue why your solution correctly solve the problem. This question is about giving efficient code and arguing for its correctness. (3pts).

27. Give a tight bound $h(n)$ for the worst-case time complexity of the method **`bool fasterCheck (const list<int>& lst)`** as a function of the length n of list **`lst`**. Explain why the worst-case time complexity of **`fasterCheck`** is in $\Theta(h(n))$ and why $h(n)$ is in $O(n^3)$ but n^3 is not in $O(h(n))$. (2pts).

```cpp
#include <iostream>
#include <list>

using namespace std;

list<int> bar(const list<int>& lst1,
              const list<int>& lst2)
{
  list<int> rslt;
  list<int>::const_iterator it1=lst1.begin();
  list<int>::const_iterator it2=lst2.begin();

  while((it1 != lst1.end())
     && (it2 != lst2.end()))
  {
    if((*it1) < (*it2)){
      rslt.push_back(*it1);
      ++it1;
    }else{
      rslt.push_back(*it2);
      ++it2;
    }
  }
  while(it1 != lst1.end()){
      rslt.push_back(*it1);
      ++it1;
  }
  while(it2 != lst2.end()){
    rslt.push_back(*it2);
    ++it2;
  }
  return rslt;
}
```

```cpp
list<int> foo(const list<int>& input)
{
  if(input.size() < 2){
    return input;
  }

  list<int> lst1;
  list<int>::const_iterator it=input.begin();

  for(int i=0; i < input.size()/2; ++i){
    lst1.push_back(*it);
    ++it;
  }
  list<int> lst2;
  for(; it != input.end(); ++it){
    lst2.push_back(*it);
  }

  list<int> rslt1 = foo(lst1);

  list<int> rslt2 = foo(lst2);

  list<int> rslt = bar(rslt1, rslt2);

  return rslt;
}

int main()
{
  list<int> in = {17, 9, 24, 18, 11, 22, 31, 8};

  list<int> out = foo(in);

  return 0;
}
```

Figure 4. Code for Part I of Problem B.2