

Københavns Universitet
Introduktion til diskret matematik og algoritmer -
Problem set 1

Victor Vangkilde Jørgensen - kft410

February 4, 2025

Contents

1	Question 1	3
1.a	3
1.b	3
1.c	3
2	Question 2	3
2.a	4
2.b	4
2.c	4
3	Question 3	4
3.a	5
3.b	5
3.c	5
4	Question 4	5
4.a	5
4.b	5
4.c	5

1 Question 1 - In the following snippet of code A and B are arrays indexed from 1 to n that contain numbers.

```
1 for i := 1 upto n {
2   B[i] := 1
3   for j := 1 upto i {
4     B[i] := B[i] * A[j]
5   }
6 }
```

- 1.a Explain in plain language what the algorithm above does. In particular, what is the meaning of the entries B [i] after the algorithm has terminated?
 - 1.b Provide an asymptotic analysis of the running time as a function of the array size n. (That is, state how the worst-case running time scales with n, focusing only on the highest-order term, and ignoring the constant factor in front of this term.)
 - 1.c Can you improve the code to run faster while retaining the same functionality? How much faster can you get the algorithm to run? Analyse the time complexity of your new algorithm. Can you prove that it is asymptotically optimal? (That is, that no algorithm solving this problem can run faster except possibly for a constant factor in the highest-order term or improvements in lower-order terms.)
- 2 Question 2 - In the following snippet of code A is an array indexed from 1 to n that contain elements that can be compared

```
1 j := n
2 good := TRUE
3 while (j > 1 and good)
4   i := j - 1
5   while (i >= 1 and good)
6     if (A[i] > A[j])
7       good := FALSE
8     i := i - 1
9   j := j - 1
10 if (good)
11   return "success"
12 else
13   return "failure"
```

-
- 2.a Explain in plain language what the algorithm above does. In particular, what do we know about the array A when "success" or "failure" is returned, respectively?
- 2.b Provide an asymptotic analysis of the running time as a function of the array size n. (That is, state how the worst-case running time scales with n, focusing only on the highest-order term, and ignoring the constant factor in front of this term.)
- 2.c Can you improve the code to run faster while retaining the same functionality? How much faster can you get the algorithm to run? Analyse the time complexity of your new algorithm. Can you prove that it is asymptotically optimal? (That is, that no algorithm solving this problem can run faster except possibly for a constant factor in the highest-order term or improvements in lower-order terms.)
- 3 Question 3 - In the following snippet of code A is an array indexed from 1 to n that contains integers, and B is an auxiliary array, also indexed from 1 to n, that is meant to contain Boolean values.

```
1  for i := 1 upto n {
2      if (A[i] < 1 or A[i] > n)
3          return "failure"
4  }
5  i := 1
6  found := -1
7  while (i <= n and found < 0) {
8      for j := 1 upto n {
9          B[j] := false
10     }
11     j := 1
12     while (B[j] == false) {
13         B[j] := true
14         j := A[j]
15     }
16     if (A[A[j]] == j)
17         found := j
18     i := i + 1
19 }
20 return found
```

-
- 3.a Explain in plain language what the algorithm above does. In particular, when does it return a positive value, and, if it does, what is the meaning of this value?
- 3.b Provide an asymptotic analysis of the running time as a function of the array size n . (That is, state how the worst-case running time scales with n , focusing only on the highest-order term, and ignoring the constant factor in front of this term.)
- 3.c Can you improve the code to run faster while retaining the same functionality? How much faster can you get the algorithm to run? Analyse the time complexity of your new algorithm. Can you prove that it is asymptotically optimal?
- 4 Question 4 - In the following snippet of code A is an array indexed from 1 to $n > 2$ containing integers.

```
1 search (A, lo, hi)
2     if (A[lo] >= A[hi])
3         return "failure"
4     else if (lo + 1 == hi)
5         return lo
6     else
7         mid = floor ((lo + hi) / 2)
8         if (A[mid] > A[lo])
9             search (A, lo mid)
10        else
11            search (A, mid, hi)
```

The first call to the algorithm is `search (A, 1, n)`, where n is whatever size (at least 2) the array has.

- 4.a Explain in plain language what the algorithm above does. If the algorithm returns something other than "failure", then what is the meaning of the value returned?
- 4.b Provide an asymptotic analysis of the running time as a function of the array size n .
- 4.c Could it be the case that recursive calls of the algorithm also return "failure", or would it be sufficient to check just once before making the first recursive call? If we get the additional guarantee that all elements in the array are distinct, could we remove the "failure" check completely, since we would be guaranteed to never have this answer returned anyway? What about if we get the additional guarantee that the array is sorted in increasing order? What if both of these extra guarantees apply?