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

Victor Vangkilde Jørgensen - kft410

February 5, 2025

Contents

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

```
for i := 1 upto n {
    B[i] := 1
    for j := 1 upto i {
        B[i] := B[i] * A[j]
    }
}
```

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?

Algoritmet ændrer hvert element i array'en B til produktet af alle elementer i A op til samme nummer element.

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

```
j := n
good := TRUE
while (j > 1 and good)
    i := j - 1
    while (i >= 1 and good)
        if (A[i] > A[j])
            good := FALSE
        i := i - 1
    j := j - 1
if (good)
    return "success"
else
    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?

Algoritmet retunerer "success" eller "failure", afhængigt af om alle elementers værdi i array'en A øges i tagt med elementnummeret, eller har samme værdi. Når "success" er retuneret, ved vi, at array'en A kun indeholder elementer, der har samme eller højere værdi end elementet før for hvert element i A. Sagt med andre ord er elle elementer i A sorteret fra laveste til højeste værdi, hvis "success" er retuneret.

Når "failure" er retuneret, er der et eller flere elementer, hvis værdi ikke øges i tagt med elementnummeret eller ikke er den samme. Med andre ord er array'en ikke sorteret.

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.

```
for i := 1 upto n {
    if (A[i] < 1 or A[i] > n)
        return "failure"
}
i := 1
found := -1
while (i <= n and found < 0) {
    for j := 1 upto n {
        B[j] := false
    }
    j := 1
    while (B[j] == false) {
        B[j] := true
        j := A[j]
    }
    if (A[A[j]] == j)
        found := j
    i := i + 1
}
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?**

Algoritmet søgerer først for, at alle elementer i array'en A kun indeholder værdier der er imellem 1 og længden (antallet af elementer) af array'en.

Kort sagt retunerer algoritmet "failure" hvis A har elementer, der er mindre end 1, eller værdier, som er større end nummeret af elementer. Hvis algoritmet ikke retunerer "failure", retunerer den -1 hvis der ikke er nogle elementer, der har samme værdi som dens index, ellers retunerer den pladsen af elementer, der har samme værdi som dens index.

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.

```
search (A, lo, hi)
    if (A[lo] >= A[hi])
        return "failure"
    else if (lo + 1 == hi)
        return lo
    else
        mid = floor ((lo + hi) / 2)
        if (A[mid] > A[lo])
            search (A, lo, mid)
        else
            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 some- thing 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?