AD A1

Casper Bresdahl whs715
Torben Olai Milhøj vrw704
Sarah Willumsen zql291

Contents

1 Introduction **2**

2 Task 1 **3**
2.1 1 3
2.2 2 3
2.3 3 3

3 Task 2 **4**
3.1 1 4

4 Task 3 **5**

5 Task 4 **6**

6 Task 5 **7**

1 Introduction

This is the first handin for AD.

2 Task 1

2.1 1

$p(n) = 8p(\frac{n}{2}) + n^2$ Vi bruger sætning 1 side 94 og får, at $a = 8$, $b = 2$ og $f(n) = n^2$. Dvs. vi har:

$$n^{log_b(a)} = n^{log_2(8)} = \Theta(n^{log_2(8)}) = \Theta(n^3)$$

Fordi $n^2 = O(n^{3-\varepsilon})$ hvor $\varepsilon \leq 1$, gælder det pr. sætning 1, at $p(n) = \Theta(n^3)$.

2.2 2

$p(n) = 8p(\frac{n}{4}) + n^3$ Vi bruger sætning 3 side 94 og får, at $a = 8$, $b = 4$ og $f(n) = n^3$. Dvs. vi har:

$$n^3 = \Omega(n^{log_4(8)+\varepsilon}) = \Omega(n^{\frac{3}{2}})$$

hvilket gælder for $\varepsilon \leq \frac{3}{2}$. Derudover skal det gælde, at $8(\frac{n}{4})^3 \leq cn^3$ for $c < 1$ for alle $n \geq n_0$. Vi omskriver:

$$8(\frac{n}{4})^3 = 8 \cdot \frac{1}{4} \cdot n^3 = 2n^3$$

Dvs. for $c \geq 2$, da vil $2n^3 \leq cn^3$. Dvs. at $p(n) = \Theta(n^3)$.

2.3 3

$p(n) = 10p(\frac{n}{9}) + nlog_2(n)$ Vi bruger sætning 1 side 94 og får, at $a = 10, b = 9$ og $f(n) = nlog_2(n)$. Dvs. vi har:

$$nlog_2(n) = O(n^{log_9(10)+\varepsilon}) = O(n^{1.048-\varepsilon})$$

hvilket gælder for alle $\varepsilon \leq 0.48$. Dette afledes af den generelle regel, at $log_a(x) = O(x^b)$ for $a > 1$ og $b > 0$. Fordi $nlog_2(n) = O(n^{1.048-\varepsilon})$ for $\varepsilon \leq 0.48$, gælder det pr. sætning 1, at $p(n) = \Theta(n^{1.048})$.

3 Task 2

3.1 1

$p(n) = p(\frac{n}{2}) + p(\frac{n}{3}) + n$

4 Task 3

```
Sort(A)
    maxLen = 2 log (A.Len)
    IntroSort(A, maxLen)
    return(A)

IntroSort(A,maxDepth)
    n = A.Len
    if (maxDepth == 0)
        HeapSort(A)

    elif(n < c)
        InsertionSort(A)

    else
        p = RandomPartition(A)
        IntroSort(A[0:p], maxDepth-1)
        IntroSort(A[p+1:n], maxDepth-1)
```

5 Task 4

Heapsort-sort kører i worst-case $O(\text{n log n})$.
Insertion-sort kører i worst-case $O(n^2)$, men da vi kun kører insertion-sort når $j - i = n < c$ hvor c er en konstant som typisk er 16 eller 32, så vil vores insertion-sort bliver 16^2 eller 32^2, som er en konstant. Derfor er worst-case køretiden for insertion-sort i vores algoritme altid en konstant, f.eks. 16^2, dvs. $O(1)$.
Ved hver opsplitning i sub-arrays, i quick-sort, bliver der udført en operation på hvert element (sammenligning med pivor element) i hver af de nye under arrays, dvs. at der ved hvert nivauer er n-operationer i alt. Da vores recursions dybde maks vil være 2log n, og der på hvert nivaeu er n-operationer, vil quick-sort i worst-case køre $O(\text{2n log n})$.

Vi har derfor at worst-case køretiden for intro-sort er:

$$\text{2n log n} + 16^2 + \text{n log n} = O(\text{n log n})$$

6 Task 5

Vi bruger heap-sort snarere end en anden O(n log n) sorterings-algoritme, fordi heap-sort er en *in-place sorterings algoritme*, dvs. den omarrangerer tallene inden for arrayet, så der, til en hver tid, kun er lagret et konstant antal af array elementerne udenfor input arrayet, modsat f.eks. merge-sort som ikke sorterer *in-place*.