Algorithms and Datastructures assignment 1

Thomas Broby Nielsen (xlq119)
Tobias Overgaard (vqg954)
Christian Buchter (zvc154)

8. maj 2015

Indhold

1 Task 1

1)
$p(n) = 8p(n/2) + n^2$
$a = 8$, $b = 2$, $f(n) = n^2$ and $n^{log_b a} = n^{log_2 8} = n^3$
since $n^{log_2 8} = n^3$ some constant $\epsilon > 0$ exists so $n^2 = O(n^{log_2(8-\epsilon)})$ is true.
We use case 1 of the master theorem
Thus $p(n) = \Theta(n^3)$

2)
$p(n) = 8p(n/4) + n^3$
$a = 8$, $b = 4$, $f(n) = n^3$ and $n^{log_b a} = n^{log_4 8} = n^{3/2}$
since $n^{log_4 8} = n^3$ some constant $\epsilon > 0$ exists so $n^3 = \Omega(n^{log_4(8+\epsilon)})$ is true.
We use case 3 of the master theorem
And $8\frac{n^3}{4^3} \leq cn^3$ and c < 1 =
$8\frac{n^3}{64} \leq cn^3$ and c < 1 =
$\frac{n^3}{8} \leq cn^3$ and c < 1 =
$\frac{1}{4}n \leq cn^3$ and c < 1 =
$\frac{1}{4} \leq c$
Thus $p(n) = \Theta(n^3)$

3)
$p(n)10p(n/9) + nlog_2 n$
$a = 10$, $b = 9$, $f(n) = nlog_2 n$ and $n^{log_b a} = n^{log_9 10} = n^{1.04795}$
Since $n^{log_9 10}$ is an exponential expression with a power > 1, and $nlog_2 n$ is a logarithmic expression, $n^{log_9 10}$ will eventually be larger than $nlog_2 n$ for a big enough n
Thus $nlog_2 n = O(n^{log_9 10-\epsilon})$ is true for a small enough ϵ and a big enough n.
$p(n) = \Theta(n^{log_9 10})$.

2 task 2

we use the substitution method to find the runtimes of the two pricing schemes:
1)
given:

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

we make the guess $p(n) = \Theta(n)$
we start by proving $p(n) = O(n)$:

$$p(n) \leq \frac{cn}{2} + \frac{cn}{3} + n = \frac{5cn}{6} + n \leq cn \quad \text{for } c \geq 6$$

for $c < 6$ the inequalities can be flipped, and we have $p(n) = \Omega(n)$
thus we have proven $p(n) = \Theta(n)$
2)
given:

$$p(n) = \sqrt{n} * p(\sqrt{n}) + \sqrt{n}$$

we make the guess $p(n) = \Theta(n)$
we start by proving $p(n) = \Omega(n)$:

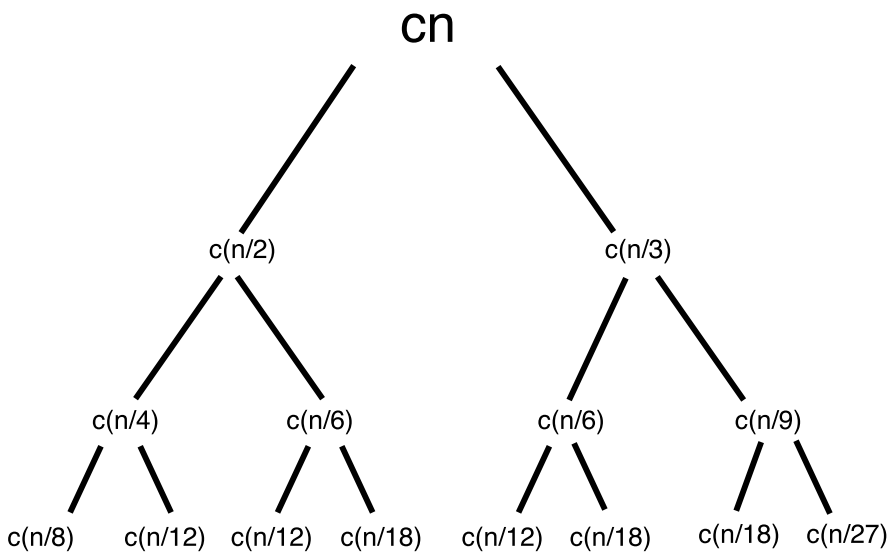
$$p(n) \geq \sqrt{n} * c\sqrt{n} + \sqrt{n} = cn + \sqrt{n} \geq cn$$

we now assume $p(n) \geq cn - 2$, which makes $p(n) = O(n)$ and prove this.

$$p(n) \leq \sqrt{n} * (c\sqrt{n} - 2) + \sqrt{n} = cn - 2\sqrt{n} + \sqrt{n} = cn - \sqrt{n} \leq cn - 2 \quad for n \geq 4$$

thus we have proven $p(n) = \Theta(n)$

From the function given to us in this task ($p(n) = p(\frac{n}{2})p(\frac{n}{3}) + n$), we have constructed the following tree to show the growth of the function. From



Figur 1: Diagram over the tree structur of the $p(n) = p(\frac{n}{2}) + p(\frac{n}{3}) + n)$ function.

this we have made the edjucated guess that our recursive function $p(n) = p(\frac{n}{2})p(\frac{n}{3}) + n$ has the running time of $O(n \cdot lgn)$.

3 Task 3

Bellow we have written the pseudo-code for Introsort. The functions Heapsort, Insertionsort, and Randomized-partition is from CLRS

```
1 int RecursionDepth = 0
2 int c = 16 or 32
3
4 IntroSort (A, i, j,)
5    if j - i > c
6       if recursionDepth > 2log(j-i)
7         HeapSort(A[i,j])
8       else
9            RecursionDepth++
10           q = RandomizedPartition (A, i, j)
11           IntroSort(A, i, q-1)
12           IntroSort(A, q+1, j)
13   else
14        Insertion
Sort(A[i,j])
```

4 Task 4

Introsort has a worst case runtime of O(n log n) because Quicksort[?] (which has a worst case runtime of O(n2)) is replaced with Heapsort[?] (which has a worst runtime of O(n log n) if it's recursiondepth becomes too big. thus the worst case will never happen. InsertionSort[?] also has a worst case runtime of O(n2), but it is used on an almost already sorted array which it is very efficient at.

5 Task 5

unlike merge sort, heapsort sorts in place: only a constant number of array elements are stored outside the input array at any time. This makes it more memory efficient.

6 Task 6

Insertion sort has a best case runtime of O(n) making it potentially much faster than the other sorting algorithms. the best case for Insertion sort is when the array is already sorted, so letting it sort an already almost sorted array will give a runtime close to its best case.