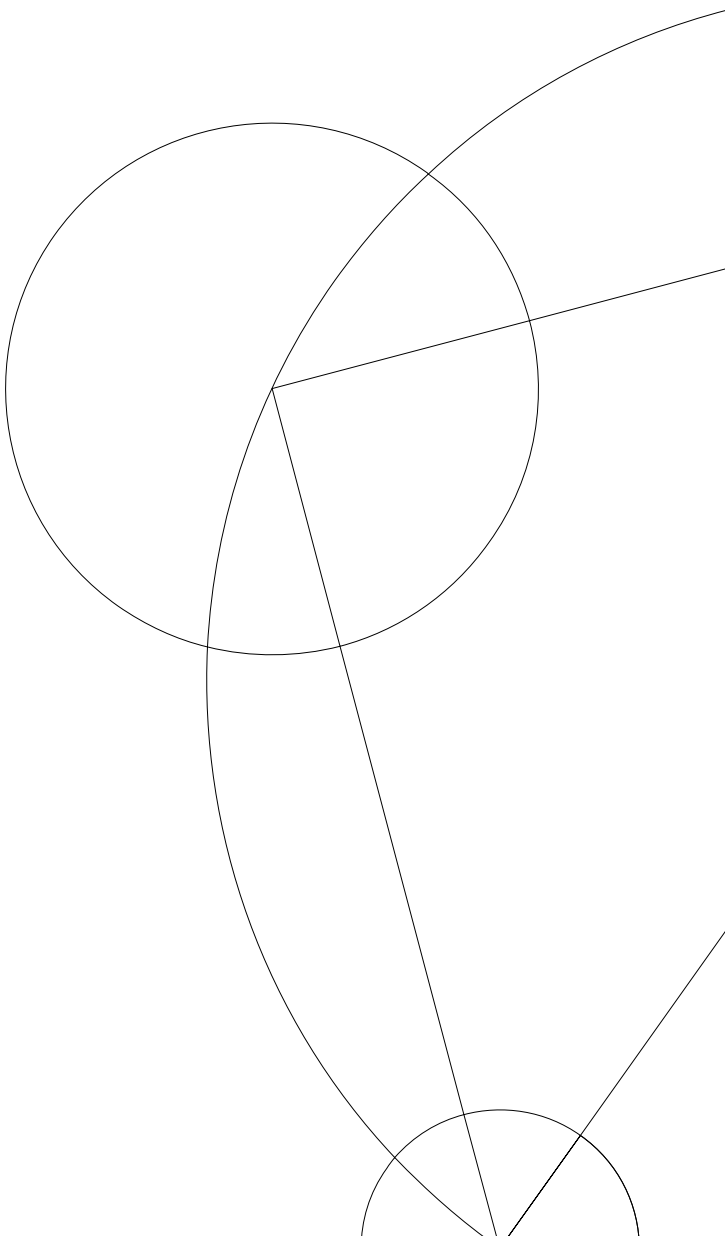
KØBENHAVNS UNIVERSITET

Assignment 1

Alex (zhc522), Adit (hjg708), Usama (mhw630), Nikolaj (bxz911)

AD 2023



10. april 2023

Exercise 2

Task 1

The task is to give an asymptotic upper bound for $p(n)$ for each of the following pricing schemes.

1. $p(n) = 8p(n/2) + n^2$

For this recurrence, we have $a = 8$, $b = 2$ and $f(n) = n^2$, we therefore have that $n^{log_b(a)} = n^{log_2(8)} = \Theta(n^3)$. Since $f(n) = O(n^{log_2(8-\varepsilon)})$, where $\varepsilon = 1$, for this we can apply case 1 of master method. We conclude that the solution is

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

2. $p(n) = 8p(n/4) + n^3$

For this recurrence, we have $a = 8$, $b = 4$ and $f(n) = n^3$, we therefore have that $n^{log_b(a)} = n^{log_4(8)} = \Theta(n^{1.5})$. Since $f(n) = \Omega(n^{log_4(8+\varepsilon)}$, where $\varepsilon = 1$, case 3 applies if we can show that the regularity condition holds for $f(n)$. For sufficiently large n, we have that $af(n/b) = 8(n/4)^4 \cdot n^3 = c \cdot f(n)$ for $c = 3/4$. Consequently, by case 3, the solution to the recurrence is

$$p(n) = \Theta(f(n) = \Theta(n^3)$$

3. $p(n) = 10p(n/9) + n \cdot log_2(n)$

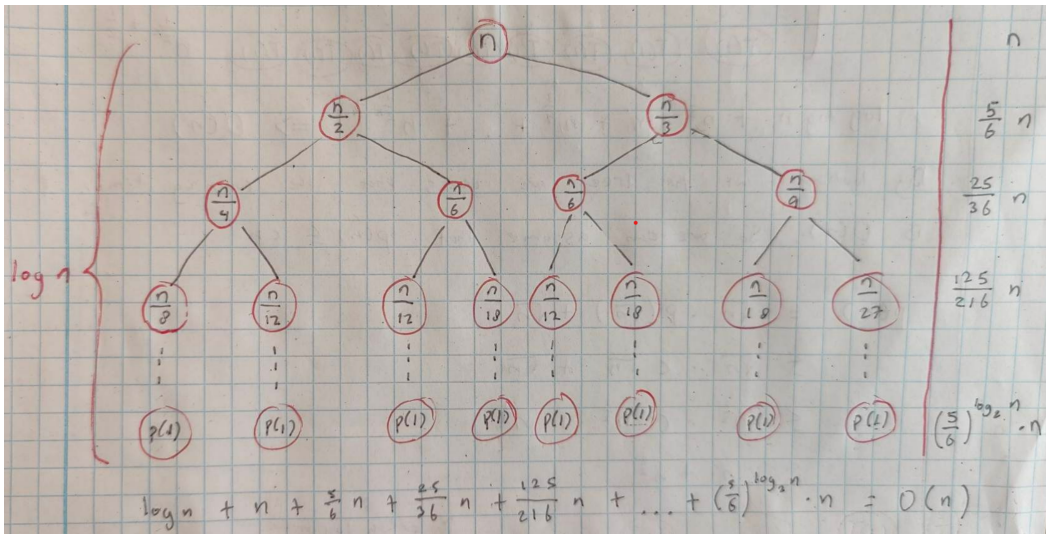
For this recurrence, we have $a = 10$, $b = 9$ and $f(n) = n \cdot log_2(n)$, we therefore have that $n^{log_b(a)} = n^{log_9(10)} = O(n^{1.0480})$. Since $f(n) = O(n^{log_9(10-\varepsilon)})$, as long as $\varepsilon < 1$. Which means that for a large enough n we can use case 1 of master method. We conclude that the solution is therefore

$$p(n) = \Theta(n^{log_b(a)}) = \Theta(n^{log_9(10)})$$

Task 2

1. $p(n) = p(n/2) + p(n/3) + n$

Recursion tree:



We can write that the cost of each level of the recursion tree is:

level 0: $\left(\frac{5}{6}\right)^0 \cdot n = n$

level 1: $\left(\frac{5}{6}\right)^1 \cdot n = \frac{5}{6}n$

level 2: $\left(\frac{5}{6}\right)^2 \cdot n = \frac{25}{36}n$

level 3: $\left(\frac{5}{6}\right)^3 \cdot n = \frac{125}{216}n$

level $log_2(n)$: $\left(\frac{5}{6}\right)^{log_2(n)} \cdot n$

The depth of the tree is $log\, n$ when a recursion tree is divided by 2 for every nodes, it will take $log\, n$ times before it reached 1. Therefore will the height of the tree be $log\, n$

The highest cost on the tree is n which is on the first node/root. We can therefore state that the total cost of the tree is $O(n)$.

From the recursion tree we can assume that $p(n) \leq cn$

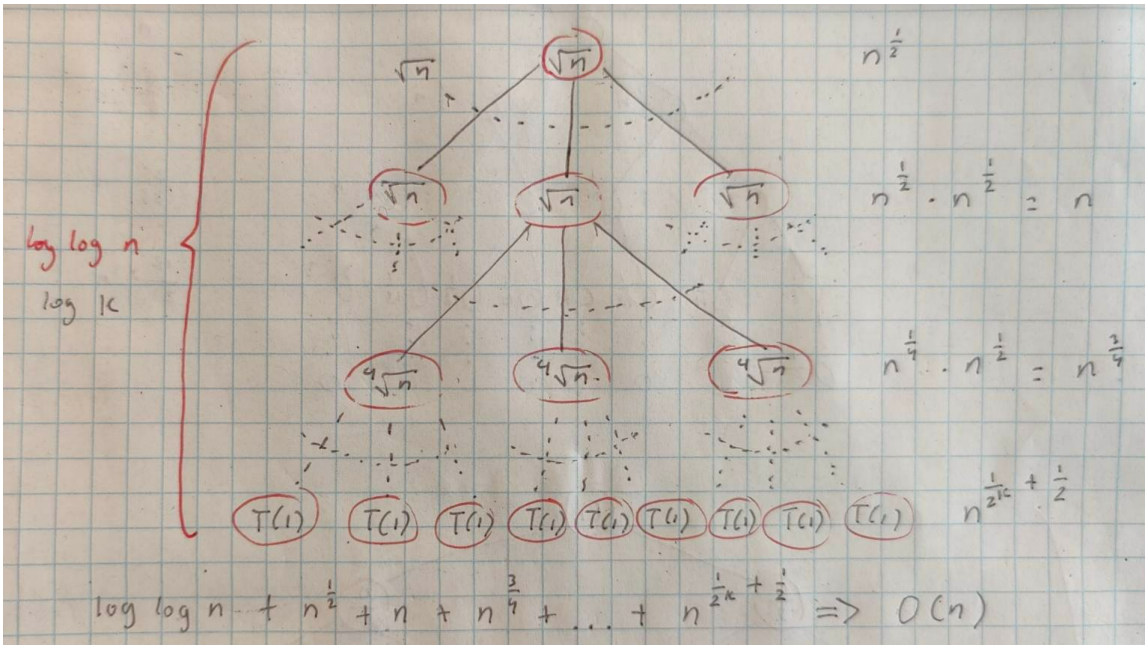
$$\begin{aligned} p(n) &\leq c\left(\frac{n}{2}\right) + c\left(\frac{n}{3}\right) + n \\ &= \frac{cn}{2} + \frac{cn}{3} + n \\ &= \frac{3cn}{6} + \frac{2cn}{6} + n \\ &= n\left(\frac{3c}{6} + \frac{2c}{6} + 1\right) \\ &= n\left(c\frac{5}{6} + 1\right) \qquad \text{for } c \geq 6, \text{ as } c\frac{5}{6} + 1 \leq c \\ &\leq cn \end{aligned}$$

The solution is therefore

$$p(n) = O(cn) = O(n)$$

2. $p(n) = \sqrt{n} \cdot p(\sqrt{n}) + \sqrt{n}$

Recursion tree:



Recursion tree shows that the asymptotic bound cost of the tree is $O(n)$.

From the recursion tree we can assume that $p(n) \leq cn$

$$\begin{aligned} p(n) &= \sqrt{n} \cdot p(\sqrt{n}) + \sqrt{n} \\ &\leq n^{1/2} \cdot p(n^{1/2}) + n^{1/2} \\ &\leq n^{1/2}(c \cdot n^{1/2}) + n^{1/2} \\ &= cn + n^{1/2} \\ &= cn + \sqrt{n} \end{aligned}$$

With this we cannot prove our initial assumption of $p(n) \leq cn$. We try again after subtracting a lower-bound term from our initial assumption. And as we have and extra $\sqrt{n}$ in our final step, we subtract a constant b from our initial, as the $p(\sqrt{n})$ is multiplied with $\sqrt{n}$.
Our new assumption would be $p(n) \leq cn - b$.

$$\begin{aligned} p(n) &= \sqrt{n} \cdot p(\sqrt{n}) + \sqrt{n} \\ &\leq \sqrt{n}(c \cdot \sqrt{n} - b) + \sqrt{n} \\ &= cn - b\sqrt{n} + \sqrt{n} \\ &= cn - (b-1)\sqrt{n} \qquad \text{for } b > 0 \\ &\leq cn - b \end{aligned}$$

This holds as long as $b > 0$.
With this we can say that the assumption of $p(n) \leq cn - b$ holds which means that the solution is $p(n) \leq cn - b$ with the asymptotic bound of $O(n)$.

$$p(n) = O(n)$$

Exercise 3

Task 3

Pseudocode for introsort:
In the pseudocode, A is the array introduced in the assignment, i and j are integers deciding where the subrange starts and ends respectively and counter is the recursion depth.

```
1  Introsort(A, i, j, counter):
2  c = 16 or 32
3  // Base cases
4  if j - i < c:
5      InsertionSort(A, i, j)
6      return
7  if counter > 2 * log n:
8      HeapSort(A, i, j)
9      return
10 // Recursive cases
11 // Randomized - Partition
12 pivotPoint = Ramdomized-Partition(A, p, r) (from clrs page 179)
13 // Recursively sort left and right subarrays
14 counter += 1
15 Introsort(A, i, pivotPoint - 1, counter)
16 Introsort(A, pivotPoint + 1, j, counter)
```

Task 4

Quicksort has a worst-case time complexity of $O(n^2)$. However, this is mitigated by the fact that the algorithm is only performed recursively for a limited number of levels (up to $2log(n)$). Thus, we can say that the overall running time of quicksort is $O(n\,log(n))$.

Heapsort has a worst-case time complexity of $O(n\,log(n))$, while insertion sort has a worst-case time complexity of $O(n^2)$. However, these insertion sort is only used when the subrange is small ($j - i < c$, where c is some constant, typically 16 or 32), so for a nearly sorted array, insertion sort have a negligible asymptotic running time. Thus, the overall running time of these two sorts is $O(n\,log(n))$, which is primarily from heapsort.

Therefore, the overall running time of introsort is $O(n\,log(n))$.

Task 5

Heap sort is an in-place sorting algorithm, meaning that it sorts within its own array. On the other hand, merge sort requires an auxiliary array that it uses for sorting. Therefore, merge sort requires more memory than heap sort.

Task 6

Insertion sort only have the O(n^2) runtime when the array is reverse sorted. So if the array is already nearly sorted, insertion sort will not need to move the elements in the array around much, and therefore won't be close to the O(n^2) runtime. Therefore it is applicable to use insertion sort to a nearly sorted array.