Sorting: homework 1 2

Eros Fabrici

Course of AA 2019-2020

Ex. 1

Generalise the SELECT algorithm to deal also with repeated values and prove that it still belongs to O(n). To generalise the SELECT algorithm, a new version of partition was implemented (can be found in `select.c`) which partitions the array by also placing the values equal to the pivot right before it. The algorithm will return the position of the last repetition of the pivot and the position of the first element before the repeated values of the pivot. Following the pseudo code.

```
function THREE_WAY_PARTITION(A, p, l, r)
    SWAP(A[p], A[l])
    p ← l
    l ← l + 1
    repetitions ← 0
    while l < r do
        if A[l] < A[p] then
            SWAP(A[l], A[p − repetitions])
            p ← l
            l ← l + 1
        else if A[l] > A[p] then
            SWAP(A[l], A[r])
            r ← r − 1
        else
            p ← l
            l ← l + 1
            repetitions ← repetitions + 1
        end if
    end while
    SWAP(A[p], A[r])
    bounds.first ← r − repetitions
    bounds.second ← r
    return bounds
end function
```

It is easy to deduce that the complexity does not change with respect to the original partition array, namely $\mathcal{O}(n)$. Now we can show the pseudo code of select.

function SELECT(A, l, r, i)
  if $r - l \leq 10$ **then**
    QUICK_SORT(A, l, r)
    return i
  end if
  $pivot \leftarrow$ SELECT_PIVOT(A, l, r)
  $bounds \leftarrow$ THREE_WAY_PARTITION$(A, pivot, l, r)$
  if $i < bounds.first$ **then**
    return SELECT$(A, l, bounds.first - 1, i)$
  else if $i > bounds.second$ **then**
    return SELECT$(A, bounds.second, r - bounds.second - 1, i)$
  else
    return i
  end if
end function

The procedure `select_pivot` is the same as the one seen during the lectures. Compared to the original algorithm, in this version there are the same number of `select` recursive calls, consequently the complexity for the algorithm is the same, namely $\mathcal{O}(n)$.

Ex. 2

Figure 2 shows clearly that quick sort with select algorithm performs worse that the generic quick sort for small inputs, while Figure 3 shows that for larger input sizes, quick sort with select performs better.

The quick sort + select algorithm uses a 3-partition algorithm and can be found in the `select.c` file.

Ex. 3

- HEAP SORT on a array A whose length is n takes $O(n)$.
  HEAP SORT first builds an heap on the given array which cost is $\theta(n)$ and then execute EXTRACT MIN (which costs $\mathcal{O}(log(n))$) on all elements except for the first one, thus we have

$$\theta(n) + \sum_{i=2}^{n} \mathcal{O}(log(i)) \leq \mathcal{O}(n) + \mathcal{O}(\sum_{i=2}^{n} log(n)) = \mathcal{O}(n\ log(n))$$

  Therefore, on average, HEAP SORT does not take $\mathcal{O}(n)$. However, EXTRACT MIN on a best case scenario, namely when all the keys in Heap are equal, takes $\theta(1)$, thus in this case we have total complexity of $\theta(n)$ which is a lower bound for our problem.

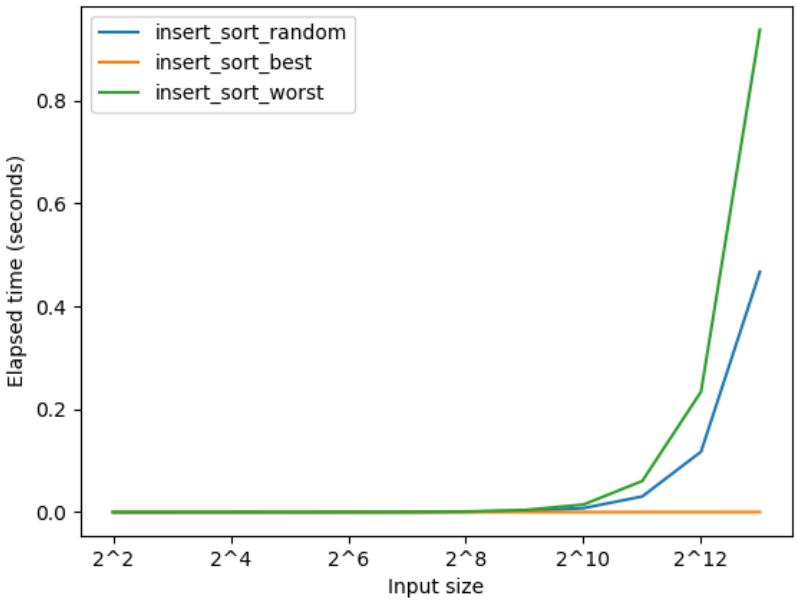

Figure 1: Insertion Sort times

- HEAP SORT on a array A whose length is n takes $\Omega(n)$.
  As stated in the previous question, HEAP SORT time complexity on a best case scenario is

  $$T_{BC}(n) \in \theta(n) \iff T_{BC}(n) \in \mathcal{O}(n) \cap \Omega(n)$$

  thus the statement is true.

- What is the worst case complexity for HEAP SORT?
  The worst case takes place when at each call of EXTRACT MIN will take exactly $\theta(log(i))$ and this happens when HEAPIFY "traverses" a path from the root to a leaf. Therefore the complexity is

  $$T_{WC}(n) \in \theta(n\ log(n))$$

- QUICK SORT on a array A whose length is n takes time $\mathcal{O}(n^3)$
  True. As QUICK SORT worst case is $\mathcal{O}(n^2)$ we have that $\mathcal{O}(n^2) \subset \mathcal{O}(n^3)$.

- What is the complexity of QUICK SORT?
  On average and best case, it takes $\theta(n\ log(n))$, while on worst case it takes $\theta(n^2)$, which takes place when the pivot is $\geq$ or $\leq$ that all the other elements in the array.

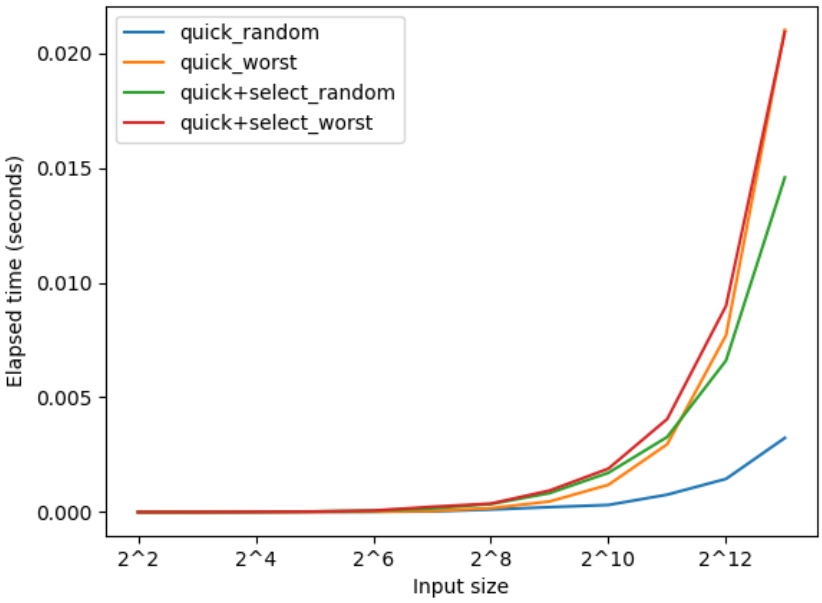

Figure 2: Quick sort times

- BUBBLE SORT on a array A whose length is n takes time $\Omega(n)$.
  True. As BUBBLE SORT time complexity is $\theta(n^2)$ then

$$\theta(n^2) \subset \Omega(n)$$

- What is the complexity of BUBBLE SORT?
  According to what we replied to the previous question it is $\theta(n^2)$.

- In the algorithm SELECT, the input elements are divided into chunks of 5. Will the algorithm work in linear time if they are divided into chunks of 7? What about chunks of 3?
  It will still work if they are divided into groups of 7, because we will still know that the median of medians is less than at least 4 elements from half of the $\lceil n/7 \rceil$ groups, so, it is greater than roughly $4n/14$ of the elements. We guess $T(n) < cn$ for $n < k$. Then, for $m \geq k$ we have

$$\begin{aligned} T(M) &\leq T(m/7) + T(10m/14) + \mathcal{O}(m) \\ &\leq cm(1/7 + 10/14) + \mathcal{O}(m) \end{aligned}$$

  therefore, as long as we have that the constant hidden in the big-O notation is less than $c/7$, we have the desired result.

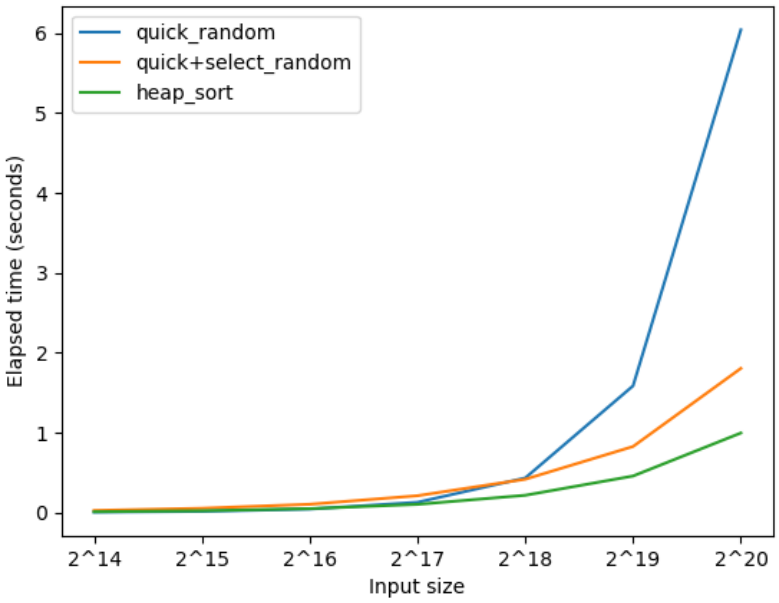

Figure 3: Quicksort vs heapsort

As regards the case in which we have groups of 3, we have that the recurrence we are able to get is $T(n) = T(\lceil n/3 \rceil) + T(4n/6) + \mathcal{O}(n) \geq T(n/3) + T(2n/3) + \mathcal{O}(n)$ thus,

$$\begin{aligned} T(m) &\geq c(m/3)log(m/3) + c(2m/3)log(2m/3) + \mathcal{O}(m) \\ &\geq cm\ log(m) + O(m) \end{aligned}$$

which grows quicker than linear.

- Suppose that you have a "black-box" worst-case linear- time subroutine to get the position in A of the value that would be in position $n/2$ if A was sorted. Give a simple, linear-time algorithm that solves the selection problem for an arbitrary position i.
  To use it, just find the median, partition the array based on that median.

  1. If i is less than half the length of the original array, apply the algorithm recursively on the first half.
  2. If i is half the length of the array, return the element coming from the median finding black box.
  3. If i is more than half the length of the array, subtract half the length of the array, and then invoke the algorithm on the second half of the array.

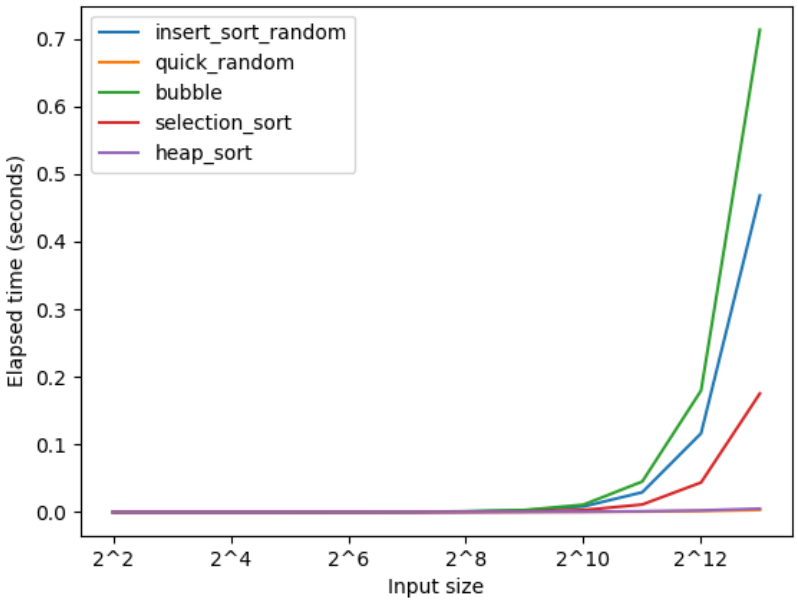

Figure 4: Benchmark

Ex. 4

Solve the following equation

$$T(n) = \begin{cases} \theta(1) & \text{if } n = 32 \\ 3T(\frac{n}{4}) + \theta(n^{3/2}) & \text{otherwise} \end{cases}$$

Solution

Figure 5 represents the recursion tree of the above equation.

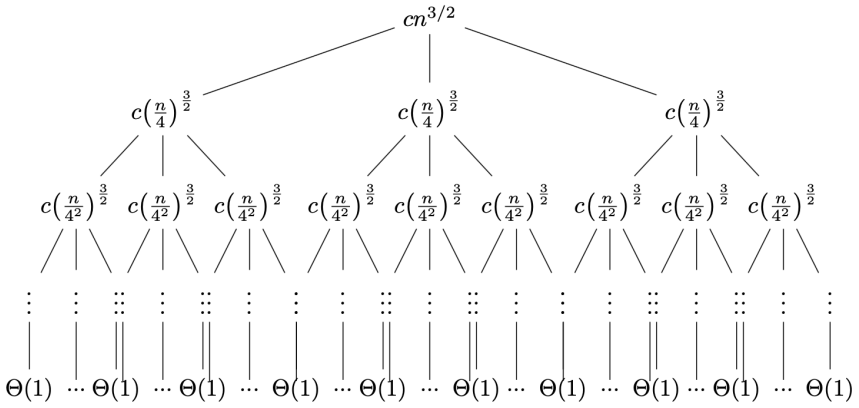

Figure 5: Recursion tree

$$
\begin{aligned}
T(n) &= cn^{3/2} + \frac{3}{4^{3/2}} cn^{3/2} + ... + \left(\frac{3}{4^{3/2}}\right)^{log_4(n/32)-1} cn^{3/2} + \theta(n^{log_4(3)}) \\
&= cn^{3/2} \sum_{i=0}^{log_4(n/32)-1} \left(\frac{3}{8}\right)^i + \theta(n^{log_4(3)}) \\
&\leq cn^{3/2} \sum_{i=0}^{\infty} \left(\frac{3}{8}\right)^i + \theta(n^{log_4(3)}) \\
&= \frac{3}{5} cn^{3/2} + \theta(n^{log_4(3)}) \in \mathcal{O}(n^{3/2})
\end{aligned}
$$

We have then that $T(n) \in \mathcal{O}(n^{3/2})$.

Ex. 5

Solve the following recurrences with both the recursion tree and the substitution method.

- $T_1(n) = 2T_1(n/2) + \mathcal{O}(n)$
  Figure 6 shows the recursion tree from which we have the following

  $$\sum_{i=0}^{\log n} cn = cn \log n$$

  thus, $T_1(n) \in \mathcal{O}(n \log n)$.

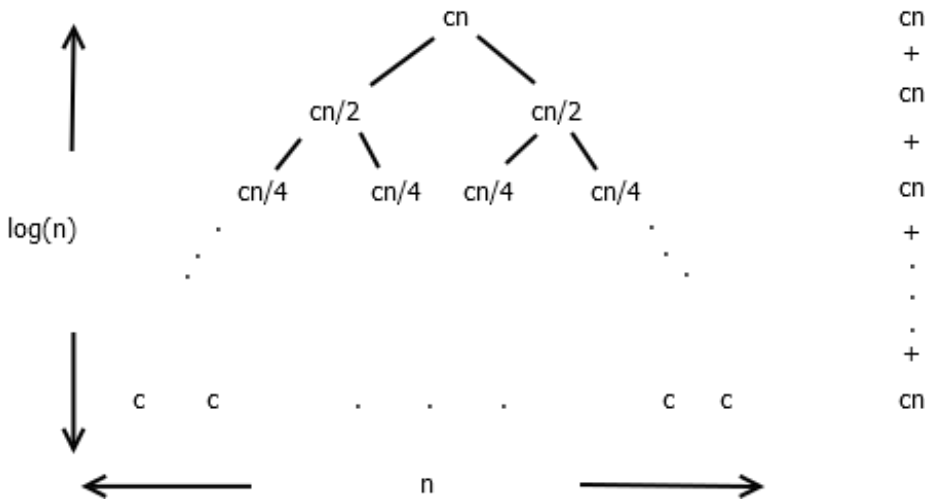

Figure 6: T_1 recursion tree

With the substitution method we assume that $T_1(n) \in \mathcal{O}(n \log n)$. Let $c > 0$ and $c' > 0$ two constants. We pick $cn \log n$ as candidate for $\mathcal{O}(n \log n)$ and $c'n$ for $\mathcal{O}(n)$.

$$\begin{aligned} T(n) &\leq 2c\frac{n}{2}\log\frac{n}{2} + c'n \\ &= cn\log\frac{n}{2} - cn + c'n \end{aligned}$$

Now when $c' \leq c$ we have that $T_1(n) \leq cn \log \frac{n}{2} \iff T_1(n) \in \mathcal{O}(n \log n)$.

- $T_2(n) = T_2(\lceil \frac{n}{2} \rceil) + T_2(\lfloor \frac{n}{2} \rfloor) + \theta(1)$
  As shown in Figure 7, the tree is not a full tree, namely not all the paths from the root to the leaves have the same length. The left-most path is the one identifying all the recursive calls of the the term $T_2(\lceil \frac{n}{2} \rceil$ and it is the longest path in the tree, while the shortest path is the right-most, which refers to the other recursive term in the equation.

  Now, let us assume that n is a power of two, then the smallest power of two greater of equal than $\log n$ is $\log 2n$, thus we can say that the left-most path is $\leq \log 2n$. Similarly, the largest power of two smaller than $\log n$ is $\log \frac{n}{2}$, thus the right-most path is $\geq \log \frac{n}{2}$. We now proceed with the

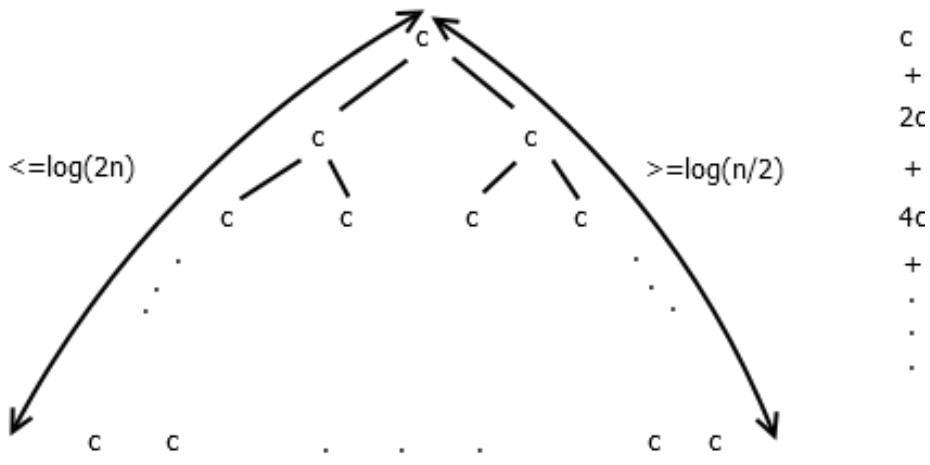

Figure 7: T_2 recursion tree

proof that $T_2(n) \leq \sum_{i=0}^{\log 2n} c2^i$ & $T_2(n) \geq \sum_{i=0}^{\log \frac{n}{2}} c2^i$.

$$\begin{aligned} T_2(n) &\leq \sum_{i=0}^{\log 2n} c2^i \\ &= c(2^{\log(2n)+1} - 1) \\ &= c2^{\log n+2} - c \\ &= 4cn - c \end{aligned}$$

Thus $T_2(n) \in \mathcal{O}(n)$. Now

$$\begin{aligned} T_2(n) &\geq \sum_{i=0}^{\log \frac{n}{2}} c2^i \\ &= c2^{\log n - \log 2 + 1} - c \\ &= cn - c \end{aligned}$$

therefore $T_2(n) \in \Omega(n)$. Finally we have that $T_2(n) \in \theta(n)$.

Now, we will prove this with the substitution method, thus we assume $T_2(n) \in \mathcal{O}(n)$ and let $c > 0$ and $d > 0$ s.t. the candidate function for $\mathcal{O}(n)$ is $cn - d$, while the candidate for $\theta(1)$ is 1. Let us assume also that $\forall m < n\ T(m) \leq cm - d$, now

$$\begin{aligned} T_2(n) &\leq c\lceil \frac{n}{2} \rceil - d + c\lfloor \frac{n}{2} \rfloor - d + 1 \\ &= cn - 2d + 1 \end{aligned}$$

now if $1 - d \leq 0 \implies T_2(n) \leq cn - d$, thus $T_2(n) \in \mathcal{O}(n)$.

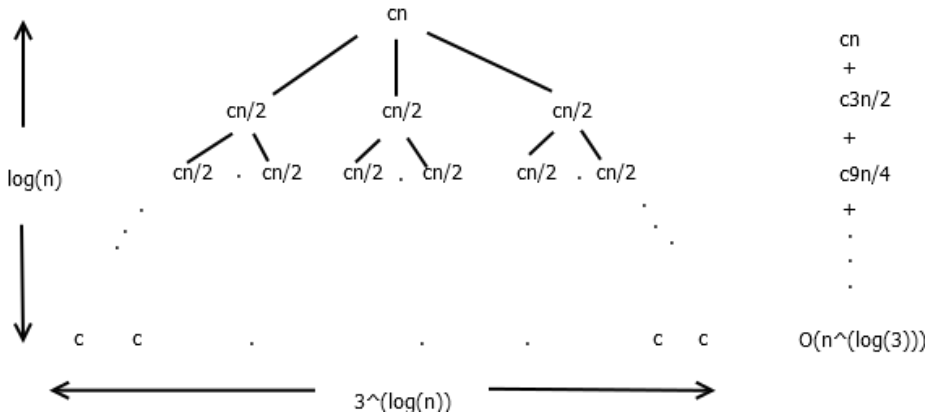

Figure 8: T_3 recursion tree

Let's prove $T_2(n) \in \Omega(n)$.

$$\begin{aligned} T_2(n) &\geq c\lceil \frac{n}{2} \rceil + c\lfloor \frac{n}{2} \rfloor \\ &= cn + 1 \\ &\geq cn \implies T_2(n) \in \Omega(n) \end{aligned}$$

Concluding, $T_2(n) \in \mathcal{O}(n) \ \& \ T_2(n) \in \Omega(n) \iff T_2(n) \in \theta(n)$.

- $T_3(n) = 3T_3(n/2) + \mathcal{O}(n)$

Figure 8 shows the recursion tree of the equation. We have,

$$\begin{aligned} T_3(n) &= cn \sum_{i=0}^{\log(n-1)} (\frac{3}{2})^i + \mathcal{O}(n^{\log 3}) \\ &= cn\Big(\frac{(3/2)^{\log n} - 1}{(3/2) - 1}\Big) + \mathcal{O}(n^{\log 3}) \\ &= 2cn((3/2)^{\log n} - 1) + \mathcal{O}(n^{\log 3}) \\ &= 2c(n^{\log 3 - 1 + 1} - n) + \mathcal{O}(n^{\log 3}) \\ &= 2c(n^{\log 3} - n) + \mathcal{O}(n^{\log 3}) \end{aligned}$$

thus $T_3(n) \in \mathcal{O}(n^{\log 3})$.

Let's prove this with substitution. We assume $T_3(n) \in \mathcal{O}(n^{\log 3})$ and let $c > 0$ and $c' > 0$ two constants for the candidate functions $cn^{\log 3}$ and $c'n$

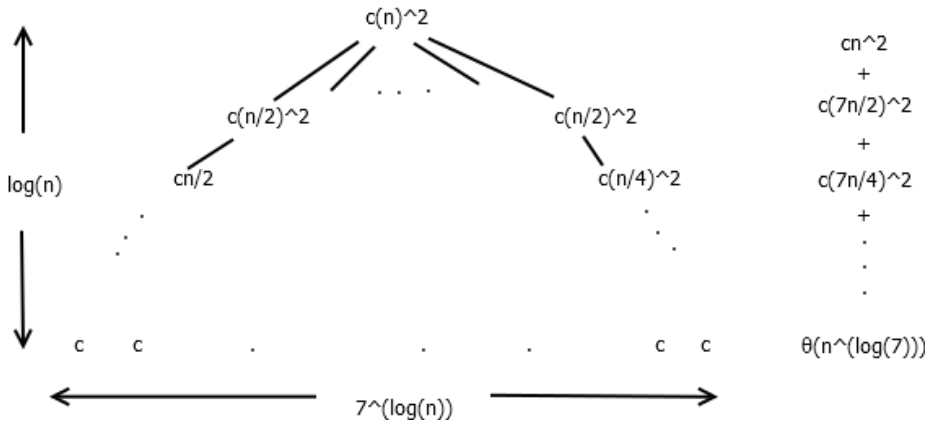

Figure 9: T_4 recursion tree

s.t.

$$\begin{aligned} T_3(n) &\leq 3c\left(\frac{n}{2}\right)^{\log 3} + c'n \\ &= cn^{\log 3} + c'n \end{aligned}$$

we are stuck here, thus we update our guess: $cn^{\log 3} - dn$

$$\begin{aligned} T_3(n) &\leq 3c(n/2)^{\log 3} - \frac{dn}{2} + c'n \\ &= cn^{\log 3} - \frac{dn}{2} + c'n \end{aligned}$$

now for $d \geq 2c'$ we have that $T_3(n) \in \mathcal{O}(n^{\log 3})$.

- $T_4(n) = 7T_4(n/2) + \theta(n^2)$
  From the recursion tree shown in Figure 9 we can obtain

$$
\begin{aligned}
T_4(n) &= cn^2 \sum_{i=0}^{\log(n-1)} \frac{7^i}{(2^i)^2} + \theta(n^{\log 7}) \\
&= cn^2 \sum_{i=0}^{\log(n-1)} \left(\frac{7}{4}\right)^i + \theta(n^{\log 7}) \\
&= cn^2 \Big(\frac{(7/4)^{\log n} - 1}{(7/4) - 1}\Big) + \theta(n^{\log 7}) \\
&= \frac{4}{3} cn^2((7/4)^{\log n} - 1) + \theta(n^{\log 7}) \\
&= \frac{4}{3} cn^2(n^{\log(7/4)} - 1) + \theta(n^{\log 7}) \\
&= \frac{4}{3} c(n^{\log 7 - 2 + 2} - n^2) + \theta(n^{\log 7}) \\
&= \frac{4}{3} c(n^{\log 7} - n^2) + \theta(n^{\log 7}) \in \theta(n^{\log 7})
\end{aligned}
$$

thus $T_4(n) \in \theta(n^{\log 7})$.

As regards the substitution, we start with our guess which is $T_4(n) \in \theta(n^{\log 7})$ and let $c > 0$ and $c' > 0$ s.t. our candidate functions are $cn^{\log 7} - dn^2$ and $c'n^2$.

$$
\begin{aligned}
T_4(n) &\leq 7c(\frac{n}{2})^{\log 7} - d(\frac{n}{2})^2 + c'n^2 \\
&= cn^{\log 7} - \frac{d}{4} n^2 + c'n^2
\end{aligned}
$$

For $d \geq 4c'$ we have that $T_4(n) \in \mathcal{O}(n^{\log 7})$. Now let's focus on the lower bound, for which we change our candidate function to $cn^{\log 7}$

$$
\begin{aligned}
T_4(n) &\geq 7c(\frac{n}{2})^{\log 7} + c'n^2 \\
&= cn^{\log 7} + c'n^2 \\
&\geq cn^{\log 7}
\end{aligned}
$$

thus we have our lower bound $T_4(n) \in \Omega(n^{\log 7})$ and we have proved that $T_4(n) \in \theta(n^{\log 7})$.