Sorting: Homework 2

1. *Generalize the* `SELECT` *algorithm to deal also with repeated values and prove that it still belongs to* $O(n)$.

   To generalize the select algorithm we have seen in order to use it also with arrays with repeated values, we have to change the `PARTITION` function.

   We will call it `SELECT_PARTITION` and it divides the array in three sections:

   - one subarray S containing all the values smaller than the pivot,
   - one subarray P containing all the values equal to the pivot,
   - one subarray G containing all the values greater than the pivot.

   The pseudo-code is the following:

```
def SELECT_PARTITION(A, l, r, p):
    swap(A, l, p)
    (p, i, j) ← (i, i + 1, r)
    while i <= j:
        if A[i] > A[p]:          // if A[i] is greater than the pivot
            swap(A, i, j)        // place it in G
            j ← j - 1            // increase G's size
        else
            if A[i] = A[p]:      // if A[i] is equal to the pivot
                p ← p + 1        // increase P's size
                swap(A, i, p)    // place it in P
                i ← i + 1
            else                 // if A[i] is smaller than the pivot
                i ← i + 1        // A[i] is already in S
            end if
        end if
    end while
    for h in l..p:
        swap(A, h, j)           // place the pivots between S and G
        j ← j - 1
        h ← h + 1
    end for
    return j, i-1
end def
```

   The complexity of this partition algorithm is still $\Theta(n)$, since the `while` is repeated n times and the `for` is repeated in the worst case (when all the elements are equal to the pivot) n times.

   All the other functions and in particular the `SELECT` function remain the same. So we still have that the recurrent relation of the `SELECT` algorithm is

$$T_S(n) = T_S(\lceil n/5 \rceil) + T_S(7n/10 + 6) + \Theta(n)$$

   so its complexity is still $O(n)$.

2. *Download the latest version of the code from*

   https://github.com/albertocasagrande/AD_sorting

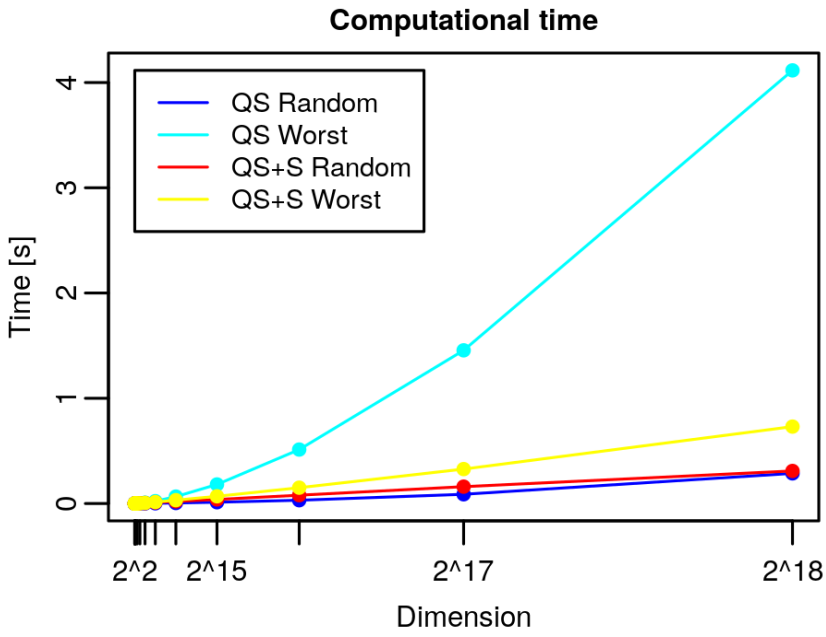
   and

   - *Implement the* `SELECT` *algorithm of Ex. 1.*

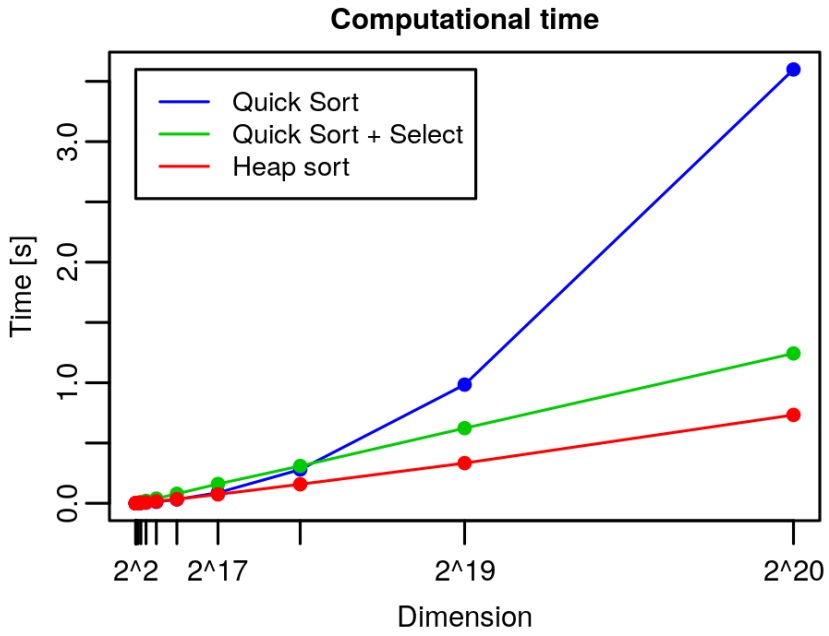
- *Implement a variant of the* `QUICK SORT` *algorithm using above-mentioned* `SELECT` *to identify the best pivot for partitioning.*
- *Draw a curve to represent the relation between the input size and the execution-time of the two variants of* `QUICK SORT` *(i.e, those of Ex. 2 and of Ex. 1 of this file) and discuss about their complexities.*

The solution with the implemented code are the functions `select_index` (with the auxiliary functions `select_pivot` and `select_partition`) and `quick_sort_select` that can be found in the file `select.c` in the folder 05_Sorting.

The following plot shows that the red line of Quick Sort + Select in the *random* case is slightly above, so it's slightly worse, than the blue line of Quick Sort in the *random* case, while the yellow line of Quick Sort + Select in the *worst* case is much more below, so it's much more better, than the light blue line of the Quick Sort in the *worst* case. Thus, while the Quick Sort algorithm is better that the Quick Sort + Select algorithm in the *random* case (since I'm already in a good case, and also performing the Select algorithm only slows down the algorithm), for the *worst* case the Select algorithm greatly improves the performance of Quick Sort.



Increasing the size of the array to be sorted, we can see that the Quick Sort algorithm with Select, for $n > 2^{18}$, is much more better than the Quick Sort algorithm alone also in the *random* case. Anyway, the Heap Sort algorithm is still better than both of them.



We have that the Quick Sort algorithm in the *random* case, with a balanced partition, is $\Theta(n \log n)$, while in the worst case, when the array is already sorted, the partition is not balanced at all, since one part is always empty, and the complexity is $\Theta(n^2)$. The Selection algorithm makes sure that every partition of the array is never unbalanced, so that there is never an empty part. In this way, also the worst case falls into the best case of Quick Sort, and the complexity is $\Theta(n \log n)$ in every case. But as we have seen from the plots, we need to have an array of length greater than 2^{18} to be able to exploit this, otherwise the Select algorithm, for small values of n, is just a burden.

3. *(Ex. 9.3-1 in [1]) 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*?*

   If the input elements are divided into chunks of 7 we have that there will be $\left\lceil \frac{n}{7} \right\rceil$ chunks, there will be $\left\lceil \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil \right\rceil$ m_i (median of the chunk C_i) greater or equal to m (the median of the m_i's), there will be $\left\lceil \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil \right\rceil - 2$ chunks that have at least 3 elements greater then m, there will be at least $4\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil \right\rceil - 2\right) \geq \frac{2}{7}n - 8$ elements that are greater than m. So an upper bound for the number of elements smaller or equal to m is $n - \left(\frac{2}{7}n - 8\right) = \frac{5}{7}n + 8$.

   So the recurrence relation becomes:

$$T_S(n) = T_S\left(\left\lceil \frac{n}{7} \right\rceil\right) + T_S\left(\frac{5}{7}n + 8\right) + \Theta(n)$$

   Substitution Method. Select cn and $c'n$ as representatives of $O(n)$ and $\Theta(n)$ and assume $T_S(m) \leq cm$ for all $m < n$,

$$\begin{aligned} T_S(n) &\leq c\left\lceil \frac{n}{7} \right\rceil + c\left(\frac{5}{7}n + 8\right) + c'n \\ &\leq c\left(\frac{n}{7} + 1\right) + c\left(\frac{5}{7}n + 8\right) + c'n \\ &\leq \frac{6}{7}cn + 9c + c'n \end{aligned}$$

   if $c \geq 20c'$, we have that

$$\begin{aligned} T_S(n) &\leq \frac{6}{7}cn + 9c + \frac{1}{20}cn \\ &\leq \frac{127}{140}cn + 9c \end{aligned}$$

So
$T_S(n) \leq cn \iff \frac{127}{140}cn + 9c \leq cn \iff \frac{127}{140}n + 9 \leq n \iff \frac{13}{140}n \geq 9 \iff n \geq \frac{1260}{13} \iff n \geq 97$
. Hence, $T_S(n) \leq cn$ for $c \geq 20c'$ and $n \geq 327$ so we still have that $T_S(n) \in O(n)$. Thus the algorithm will work in linear time if the input elements are divided into chunks of 7.

If the input elements are divided into chunks of 3 we have that there will be $\left\lceil \frac{n}{3} \right\rceil$ chunks, there will be $\left\lceil \frac{1}{2}\left\lceil \frac{n}{3} \right\rceil \right\rceil$ m_i (median of the chunk C_i) greater or equal to m (the median of the m_i's), there will be $\left\lceil \frac{1}{2}\left\lceil \frac{n}{3} \right\rceil \right\rceil - 2$ chunks that have at least 3 elements greater then m, there will be at least $2\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{3} \right\rceil \right\rceil - 2\right) \geq \frac{2}{6}n - 4 = \frac{1}{3}n - 4$ elements that are greater than m. So an upper bound for the number of elements smaller or equal to m is $n - \left(\frac{1}{3}n - 4\right) = \frac{2}{3}n + 4$.

So the recurrence relation becomes:

$$T_S(n) = T_S\left(\left\lceil \frac{n}{3} \right\rceil\right) + T_S\left(\frac{2}{3}n + 4\right) + \Theta(n)$$

Substitution Method. Select cn and $c'n$ as representatives of $O(n)$ and $\Theta(n)$ and assume $T_S(m) \leq cm$ for all $m < n$,

$$\begin{aligned} T_S(n) &\leq c\left\lceil \frac{n}{3} \right\rceil + c\left(\frac{2}{3}n + 4\right) + c'n \\ &\leq c\left(\frac{n}{3} + 1\right) + c\left(\frac{2}{3}n + 4\right) + c'n \\ &\leq cn + 5c + c'n \end{aligned}$$

if $c \geq 20c'$, we have that

$$\begin{aligned} T_S(n) &\leq cn + 5c + \frac{1}{20}cn \\ &\leq \frac{21}{20}cn + 7c \end{aligned}$$

So $T_S(n) \leq cn \iff \frac{21}{20}cn + 4c \leq cn \iff \frac{21}{20}n + 4 \leq n \iff \frac{1}{20}n \leq -4 \iff n \leq -80$, which is impossible since $n \geq 0$. Hence, $T_S(n) > cn$, so it is not linear: $T_S(n) \notin O(n)$. Thus the algorithm will *not* work in linear time if the input elements are divided into chunks of 3.

4. *(Ex. 9.3-5 in [1] 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.*

   The algorithm will find the median values using the "black-box" algorithm which is $O(n)$, then it will perform a call of the `PARTITION` algorithm (or the `SELECT_PARTITION` algorithm if we have repeated values) that is $O(n)$ in order to have the array partition with all the elements smaller than the median on the left, the median found by the "black-box" algorithm in the right place and all the elements greater than the median on the right. Then it will perform a recursive step on only the sub-array containing i. The code is the following:

```
def BLACK_BOX_SELECT(A, i, l, r):
    m ← BLACK_BOX(A)
    if i = m:
        return A[m]
    end if
    PARTITION(A, l, r, m)
    if i < m
        BLACK_BOX_SELECT(A, i, l , m - 1)
    else
        BLACK_BOX_SELECT(A, i, m + 1, r)
    end if
end def
```

The recurrent relation of the code is $T(n) = T(n/2) + O(n) + O(n) = T(n/2) + O(n)$, and using the recursion tree and choosing cn as representative for $O(n)$ we have that

$$\begin{aligned} T(n) &= T(n/2) + cn = T(n/4) + cn/2 + cn = \dots \\ &= \sum_{i=0}^{\log_2 n} \frac{cn}{2^i} = cn \sum_{i=0}^{\log_2 n} \left(\frac{1}{2}\right)^i = \\ &= cn \frac{1 - (1/2)^{\log_2 n + 1}}{1 - 1/2} = 2cn \left(1 - \frac{1}{2^{\log_2 n + 1}}\right) = \\ &= 2cn \left(1 - \frac{1}{2n}\right) = 2cn - c \in O(n) \end{aligned}$$

so the complexity of the algorithm is linear.

5. *Solve the following recursive equations by using both the recursion tree and the substitution method:*

   - $T_1(n) = 2 * T_1(n/2) + O(n)$

     Using the **recursion tree**, we have that, choosing cn as representative for $O(n)$

     $$\begin{aligned} T_1(n) &= 2 * T_1(n/2) + O(n) \\ &\leq 2 * T_1(n/2) + cn \\ &\leq 2 * (2 * T_1(n/4) + cn/2) + cn \\ &= 4 * T_1(n/4) + 2cn \\ &\leq 4 * (2 * T_1(n/8) + cn/4) + 2cn \\ &= 8 * T_1(n/8) + 3cn \\ &\leq \dots \\ &\leq 2^{\log_2 n} T_1(0) + \log_2 n * cn \\ &= n * 0 + cn \log_2 n \in O(n \log n) \end{aligned}$$

     or directly

     $$T_1(n) = 2 * T_1(n/2) + O(n) \leq \sum_{i=0}^{\log_2 n} 2^i c \frac{n}{2^i} = \sum_{i=0}^{\log_2 n} cn = cn \sum_{i=0}^{\log_2 n} 1 = cn \log_2 n \in O(n \log n)$$

     Using the **substitution method**, we guess that $T_1(n) \in O(n \log n)$. We select a representative for $O(n \log n)$ and $O(n)$, e.g. $cn \log n$ and $c'n$. We assume that $\forall m < n,\ T_1(m) \leq cm \log m$. If this is the case,

     $$\begin{aligned} T_1(n) &= 2 * T_1(n/2) + c'n \\ &\leq 2 * cn/2 \log(n/2) + c'n \\ &= cn(\log n - \log 2) + c'n \\ &= cn \log n - cn \log 2 + c'n \\ &= cn \log n - cn + c'n \end{aligned}$$

     if $c'n - cn \leq 0 \iff c'n \leq cn \iff c \geq c'$ then $T_1(n) \leq cn \log n$. Thus we have proved by induction that $\forall n \in \mathbb{N},\ T_1(n) \leq cn \log n$ for a proper c. So $T_1(n) \in O(n \log n)$.

   - $T_2(n) = T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + \Theta(1)$

     Using the **recursion tree**, we have that, choosing c as representative for $O(1)$:

     $$\begin{aligned} T_2(n) &= T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + \Theta(1) \\ &= T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + c \\ &= (T_2(\lceil \lceil n/2 \rceil / 2 \rceil) + T_2(\lfloor \lceil n/2 \rceil / 2 \rfloor) + c) + (T_2(\lceil \lfloor n/2 \rfloor / 2 \rceil) + T_2(\lfloor \lfloor n/2 \rfloor / 2 \rfloor) + c) + c \end{aligned}$$

     if n is even then we can simply remove the floor and ceiling functions, if n is odd instead, so if $n = m + 1$ with m even, we have that

$$\left\lceil \frac{\lfloor n/2 \rfloor}{2} \right\rceil = \left\lceil \left(\frac{m}{2}+1\right)/2 \right\rceil = \left\lceil \frac{m}{4}+\frac{1}{2} \right\rceil = \left\lceil \frac{m}{4}+\frac{1}{4} \right\rceil = \left\lceil \frac{m+1}{4} \right\rceil = \left\lceil \frac{n}{4} \right\rceil$$
$$\left\lfloor \frac{\lceil n/2 \rceil}{2} \right\rfloor = \left\lfloor \left(\frac{m}{2}+1\right)/2 \right\rfloor = \left\lfloor \frac{m}{4}+\frac{1}{2} \right\rfloor = \left\lceil \frac{m}{4} \right\rceil = \left\lceil \frac{n}{4} \right\rceil$$
$$\left\lceil \frac{\lfloor n/2 \rfloor}{2} \right\rceil = \left\lceil \left(\frac{m}{2}\right)/2 \right\rceil = \left\lceil \frac{m}{4} \right\rceil = \left\lceil \frac{m}{4}+\frac{1}{4} \right\rceil = \left\lceil \frac{m+1}{4} \right\rceil = \left\lceil \frac{n}{4} \right\rceil$$
$$\left\lfloor \frac{\lfloor n/2 \rfloor}{2} \right\rfloor = \left\lfloor \left(\frac{m}{2}\right)/2 \right\rfloor = \left\lfloor \frac{m}{4} \right\rfloor = \left\lfloor \frac{m}{4}+\frac{1}{4} \right\rfloor = \left\lfloor \frac{m+1}{4} \right\rfloor = \left\lfloor \frac{n}{4} \right\rfloor$$

so it follows that

$$\begin{aligned}
T_2(n) &= T_2(\lceil n/4 \rceil) + T_2(\lceil n/4 \rceil) + T_2(\lceil n/4 \rceil) + T_2(\lfloor n/4 \rfloor) + 3c \\
&= 3T_2(\lceil n/4 \rceil) + T_2(\lfloor n/4 \rfloor) + 3c \\
&= 3(T_2(\lceil \lceil n/4 \rceil /2 \rceil) + T_2(\lfloor \lceil n/4 \rceil /2 \rfloor) + c) + (T_2(\lceil \lfloor n/4 \rfloor /2 \rceil) + T_2(\lfloor \lfloor n/4 \rfloor /2 \rfloor) + c) + 3c \\
&= 3(T_2(\lceil n/8 \rceil) + T_2(\lceil n/8 \rceil) + c) + (T_2(\lceil n/8 \rceil) + T_2(\lfloor n/8 \rfloor) + c) + 3c \\
&= 3T_2(\lceil n/8 \rceil) + 3T_2(\lceil n/8 \rceil) + 3c + T_2(\lceil n/8 \rceil) + T_2(\lfloor n/8 \rfloor) + c + 3c \\
&= 7T_2(\lceil n/8 \rceil) + T_2(\lfloor n/8 \rfloor) + 7c \\
&= \dots \\
&= (2^{\log_2 n} - 1)T(0) + T(0) + c(2^{\log_2 n} - 1) \\
&= 0 + 0 + c(n-1) \in O(n)
\end{aligned}$$

Or alternatively

We have that the tree is complete up to the length of the shortest branch (which is the rightmost) → it is the shortest branch, while the left one is the longest.

For sure there exist a power of 2 between n and $2n$, and between $\frac{n}{2}$ and n, so the length of the recursion tree is $\leq \log_2(2n)$ and $\geq \log_2(\frac{n}{2})$.

$$T_2(n) \geq \sum_{i=0}^{\log_2(\frac{n}{2})} c2^i = c \sum_{i=0}^{\log_2(\frac{n}{2})} 2^i = c \cdot \frac{2^{\log_2(\frac{n}{2})+1} - 1}{2-1} = c \cdot \left(2\left(\frac{n}{2}\right) - 1\right) = c(n-1) = cn - c \in \Omega(n)$$
$$T_2(n) \leq \sum_{i=0}^{\log_2(2n)} c'2^i = c' \sum_{i=0}^{\log_2(2n)} 2^i = c' \cdot \frac{2^{\log_2(2n)+1} - 1}{2-1} = c' \cdot (2(2n) - 1) = c'(4n-1) = 4c'n - c' \in O(n)$$

So we have that $T_2(n) \in \Theta(n)$.

Using the **substitution method**, we guess that $T_2 \in O(n)$ and we select the function cn as representative, and we select c' as representative of $\Theta(1)$. Inductive assumption: we assume that $\forall m < n,\ T_2(m) \leq cm$. We want to prove that $\forall n,\ T_2(n) \leq cn$. If this is the case we have that

$$\begin{aligned}
T_2(n) &= T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + \Theta(1) \\
&\leq T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + c' \\
&\leq c\lceil n/2 \rceil + c\lfloor n/2 \rfloor + c' \\
&\leq c(\lceil n/2 \rceil + \lfloor n/2 \rfloor) + c' \\
&\leq cn + c'
\end{aligned}$$

But this is **not** what we wanted to prove, we cannot conclude anything! This doesn't prove $T_2 \in O(n)$ because $cn + c' \not\leq cn$: but we are not able to prove it for a term of lower order. The problem is that we selected the wrong representative for $O(n)$, let us choose $cn - d \in O(n)$. Inductive assumption: we assume that
$\forall m < n,\ T_2(m) \leq cm - d$. We want to prove that $\forall n,\ T_2(n) \leq cn - d$.

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

If $c' - d \leq 0 \iff c' \leq d$, then $T_2(n) \leq cn - d$. Thus we have proved by induction that $\forall n \in \mathbb{N}$, $T_2(n) \leq cn - d$ for proper c and d. So $T_2(n) \in O(n)$.

Now we want to prove that $T_2(n) \in \Omega(n)$. We select the representative $cn \in \Omega(n)$ and the representative $c' \in \Theta(1)$. Inductive assumption: we assume that $\forall m < n,\ T_2(m) \geq cm$. We want to prove that $\forall n,\ T_2(n) \geq cn$.

$$\begin{aligned} T_2(n) &= T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + \Theta(1) \\ &\geq T_2(\lceil n/2 \rceil) + T_2(\lfloor n/2 \rfloor) + c' \\ &\geq c\lceil n/2 \rceil + c\lfloor n/2 \rfloor + c' \\ &\geq c(\lceil n/2 \rceil + \lfloor n/2 \rfloor) + c' \\ &\geq cn + c' \\ &\geq cn \end{aligned}$$

Thus we have that $T_2(n) \geq cn$, so we have proved by induction that $\forall n \in \mathbb{N}$, $T_2(n) \geq cn$ for a proper c. Then $T_2(n) \in \Omega(n)$. Therefore we have proved that $T_2(n) \in \Theta(n)$.

By using substitution method, you can prove that $T_2(n) \in O(n^2)$. This is not wrong, because $O(n) \subseteq O(n^2)$, however this is a curse complexity bound.

- $T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{3}{4}n\right) + \Theta(n)$

  This equation is weird because it is totally unbalanced! All the branches are different in length! Let's call $\alpha = \frac{1}{5}$ and $\beta = \frac{3}{4}$. On the i-th level all the nodes contains $\alpha^j \beta^{i-j}$; the number of nodes on the i-th level having $\alpha^j \beta^{i-j}$ elements to deal with is the number of possible way in which we can select j left moves over i total moves: so $\binom{i}{j}$ moves in total. Thus at each level i we have

  $$\sum_{j=0}^{i} \binom{i}{j} \alpha^j \beta^{i-j} n$$

  So this is the total number of elements we need to deal with at the i-th level of our recursion tree. We have that

  $$\sum_{j=0}^{i} \binom{i}{j} \alpha^j \beta^{i-j} n = (\alpha + \beta)^i \cdot n$$

  To do an over approximation of the complexity, let's suppose that the tree has infinite length. So

  $$T(n) \leq \sum_{i=0}^{\infty} (\alpha + \beta)^i \cdot n = \sum_{i=0}^{\infty} \left(\frac{1}{5} + \frac{3}{4}\right)^i \cdot n = \sum_{i=0}^{\infty} \left(\frac{19}{20}\right)^i \cdot n.$$

  Because of the convergence of geometric series $\left(\frac{19}{20} < 1\right)$ $T(n) \leq \frac{1}{1 - \frac{19}{20}} n = 20n$, so $T(n) \in O(n)$.

  Can we lower bound it? Of course yes, since just the first call of our recursive call costs $\Theta(n)$, so $T(n) \in \Omega(n)$. Thus $T(n) \in \Theta(n)$.

  So just solving this problem we proved a new theorem:

  > **Theorem:** Let α and β be two natural numbers. If $\alpha + \beta < 1$, then $T(n) = T(\alpha n) + T(\beta n) + \Theta(n)$ belongs to $\Theta(n)$.

- $T_3(n) = 3 * T_3(n/2) + O(n)$

  Using the **recursion tree**, we have that, choosing cn as representative for $O(n)$:

$$
\begin{aligned}
T_3(n) &= 3 * T_3(n/2) + O(n) \\
&\leq 3 * T_3(n/2) + cn \\
&\leq 3 * (3 * T_3(n/4) + cn/2) + cn \\
&= 9 * T_3(n/4) + 3/2cn + cn \\
&= 9 * T_3(n/4) + 3/2cn + cn \\
&\leq 9 * (3 * T_3(n/8) + cn/4) + 3/2cn + cn \\
&= 27 * T_3(n/8) + 9/4cn + 3/2cn + cn \\
&\leq 27 * (3 * T_3(n/16) + cn/8) + 9/4cn + 3/2cn + cn \\
&= 81 * T_3(n/16) + 27/8cn + 9/4cn + 3/2cn + cn \\
&\leq \dots \\
&\leq 3^{\log_2 n} \cdot T_3(0) + \sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i \cdot cn \\
&= 0 + cn \cdot \frac{(3/2)^{(\log_2 n)+1} - 1}{3/2 - 1} = cn \cdot \frac{\frac{3^{(\log_2 n)+1} - 2^{(\log_2 n)+1}}{2^{(\log_2 n)+1}}}{\frac{3-2}{2}} \\
&= 2cn \cdot \frac{3 \cdot 3^{\log_2 3 \cdot \log_3 n} - 2n}{2n} = c(3 \cdot (3^{\log_3 n})^{\log_2 3} - 2n) \\
&= c(3 \cdot n^{\log_2 3} - 2n) = 3cn^{\log_2 3} - 2cn \in O(n^{\log_2 3})
\end{aligned}
$$

because since $\log_b n = \log_b a \cdot \log_a n$ we have that $\log_2 n = \log_2 3 \cdot \log_3 n$.

Or directly

$$
\begin{aligned}
T_3(n) = 3 * T_3(n/2) + O(n) &\leq \sum_{i=0}^{\log_2 n} 3^i c\frac{n}{2^i} = cn \sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i = cn \cdot \frac{(3/2)^{(\log_2 n)+1} - 1}{3/2 - 1} \\
&= cn \cdot \frac{\frac{3^{(\log_2 n)+1} - 2^{(\log_2 n)+1}}{2^{(\log_2 n)+1}}}{\frac{3-2}{2}} = 2cn \cdot \frac{3 \cdot 3^{\log_2 3 \cdot \log_3 n} - 2n}{2n} = c(3 \cdot (3^{\log_3 n})^{\log_2 3} - 2n) \\
&= c(3 \cdot n^{\log_2 3} - 2n) = 3cn^{\log_2 3} - 2cn \in O(n^{\log_2 3})
\end{aligned}
$$

Using the **substitution method**, we guess that $T_3 \in O(n^{\log_2 3})$ and we select the function $cn^{\log_2 3} - dn$ as representative, and we select $c'n$ as representative of $O(n)$. Inductive assumption: we assume that $\forall m < n,\ T_3(m) \leq cm^{\log_2 3} - dm$. We want to prove that $\forall n,\ T_3(n) \leq cn^{\log_2 3} - dn$. If this is the case we have that

$$
\begin{aligned}
T_3(n) &= 3 * T_3(n/2) + O(n) \\
&\leq 3\left(c\left(\frac{n}{2}\right)^{\log_2 3} - d\frac{n}{2}\right) + c'n \\
&= 3c\frac{n^{\log_2 3}}{2^{\log_2 3}} - n\left(\frac{3}{2}d - c'\right) \\
&= 3c\frac{n^{\log_2 3}}{3} - n\left(\frac{3d - 2c'}{2}\right) \\
&= cn^{\log_2 3} - \frac{3d - 2c'}{2}n
\end{aligned}
$$

if $\frac{3d-2c'}{2} \leq d \iff 3d - 2c' \leq 2d \iff d \leq 2c'$, then $T_3(n) \leq cn^{\log_2 3} - dn$. Thus we have proved by induction that $\forall n \in \mathbb{N},\ T_3(n) \leq cn^{\log_2 3} - dn$ for a proper c and d. So $T_3(n) \in O(n^{\log_2 3})$.

- $T_4(n) = 7 * T_4(n/2) + \Theta(n^2)$

  Using the **recursion tree**, we have that, choosing cn^2 as representative for $\Theta(n^2)$:

$$
\begin{aligned}
T_4(n) &= 7 * T_4(n/2) + \Theta(n^2) \\
&= 7 * T_4(n/2) + cn^2 \\
&= 7 * \left(7 * T_4\left(\frac{n}{4}\right) + c\left(\frac{n}{2}\right)^2\right) + cn^2 \\
&= 49 * T_4\left(\frac{n}{4}\right) + 7c\left(\frac{n}{2}\right)^2 + cn^2 \\
&= 49 * \left(7 * T_4\left(\frac{n}{8}\right) + c\left(\frac{n}{4}\right)^2\right) + 7c\left(\frac{n}{2}\right)^2 + cn^2 \\
&= 343 * T_4\left(\frac{n}{8}\right) + 49c\left(\frac{n}{4}\right)^2 + 7c\left(\frac{n}{2}\right)^2 + cn^2 \\
&= \dots \\
&= 7^{\log_2 n} T_4(0) + \sum_{i=0}^{\log_2 n} 7^i \cdot c\left(\frac{n}{2^i}\right)^2 \\
&= 0 + cn^2 \sum_{i=0}^{\log_2 n} \left(\frac{7}{4}\right)^i = cn^2 \cdot \frac{(7/4)^{(\log_2 n)+1} - 1}{7/4 - 1} \\
&= cn^2 \cdot \frac{\frac{7^{(\log_2 n)+1} - 4^{(\log_2 n)+1}}{4^{(\log_2 n)+1}}}{\frac{7-4}{4}} = \frac{4}{3}cn^2 \cdot \frac{7 \cdot 7^{\log_2 7 \cdot \log_7 n} - 4 \cdot 4^{\log_2 n}}{4 \cdot 4^{\log_2 n}} \\
&= \frac{4}{3}cn^2 \cdot \frac{7 \cdot (7^{\log_7 n})^{\log_2 7} - 4 \cdot (2^{\log_2 n})^2}{4 \cdot (2^{\log_2 n})^2} = \frac{4}{3}cn^2 \cdot \frac{7 \cdot n^{\log_2 7} - 4n^2}{4n^2} \\
&= \frac{7}{3}cn^{\log_2 7} - \frac{4}{3}cn^2 \in O(n^{\log_2 7})
\end{aligned}
$$

Or directly

$$
\begin{aligned}
T_4(n) = 7 * T_4(n/2) + \Theta(n^2) &\leq \sum_{i=0}^{\log_2 n} 7^i c\left(\frac{n}{2^i}\right)^2 = cn^2 \sum_{i=0}^{\log_2 n} \left(\frac{7}{4}\right)^i = cn^2 \cdot \frac{(7/4)^{(\log_2 n)+1} - 1}{7/4 - 1} \\
&= cn^2 \cdot \frac{\frac{7^{(\log_2 n)+1} - 4^{(\log_2 n)+1}}{4^{(\log_2 n)+1}}}{\frac{7-4}{4}} = \frac{4}{3}cn^2 \cdot \frac{7 \cdot 7^{\log_2 7 \cdot \log_7 n} - 4 \cdot 4^{\log_2 n}}{4 \cdot 4^{\log_2 n}} \\
&= \frac{4}{3}cn^2 \cdot \frac{7 \cdot (7^{\log_7 n})^{\log_2 7} - 4 \cdot (2^{\log_2 n})^2}{4 \cdot (2^{\log_2 n})^2} = \frac{4}{3}cn^2 \cdot \frac{7 \cdot n^{\log_2 7} - 4n^2}{4n^2} \\
&= \frac{7}{3}cn^{\log_2 7} - \frac{4}{3}cn^2 \in O(n^{\log_2 7})
\end{aligned}
$$

Besides, we have that

$$
\begin{aligned}
T_4(n) = 7 * T_4(n/2) + \Theta(n^2) &\geq \sum_{i=0}^{\log_2 n} 7^i c\left(\frac{n}{2^i}\right)^2 = cn^2 \sum_{i=0}^{\log_2 n} \left(\frac{7}{4}\right)^i = cn^2 \cdot \frac{(7/4)^{(\log_2 n)+1} - 1}{7/4 - 1} \\
&= cn^2 \cdot \frac{\frac{7^{(\log_2 n)+1} - 4^{(\log_2 n)+1}}{4^{(\log_2 n)+1}}}{\frac{7-4}{4}} = \frac{4}{3}cn^2 \cdot \frac{7 \cdot 7^{\log_2 7 \cdot \log_7 n} - 4 \cdot 4^{\log_2 n}}{4 \cdot 4^{\log_2 n}} \\
&= \frac{4}{3}cn^2 \cdot \frac{7 \cdot (7^{\log_7 n})^{\log_2 7} - 4 \cdot (2^{\log_2 n})^2}{4 \cdot (2^{\log_2 n})^2} = \frac{4}{3}cn^2 \cdot \frac{7 \cdot n^{\log_2 7} - 4n^2}{4n^2} \\
&= \frac{7}{3}cn^{\log_2 7} - \frac{4}{3}cn^2 \in \Omega(n^{\log_2 7})
\end{aligned}
$$

So we have that $T_4(n) \in \Theta(n^{\log_2 7})$.

Using the **substitution method**, we guess that $T_4 \in O(n^{\log_2 7})$ and we select the function $cn^{\log_2 7} - dn^2$ as representative, and we select $c'n^2$ as representative of $\Theta(n^2)$. Inductive assumption: we assume that $\forall m < n,\ T_3(m) \leq cm^{\log_2 7} - dm^2$. We want to prove that $\forall n,\ T_4(n) \leq cn^{\log_2 7} - dn^2$. If this is the case we have that

$$
\begin{aligned}
T_4(n) &= 7 * T_4(n/2) + \Theta(n^2) \\
&\leq 7\left(c\Big(\frac{n}{2}\Big)^{\log_2 7} - d\Big(\frac{n}{2}\Big)^2\right) + c'n^2 \\
&= 7c\frac{n^{\log_2 7}}{2^{\log_2 7}} - 7d\frac{n^2}{4} + c'n^2 \\
&= 7c\frac{n^{\log_2 7}}{7} - n^2\left(\frac{7}{4}d - c'\right) \\
&= cn^{\log_2 7} - \frac{7d-4c'}{4}n^2
\end{aligned}
$$

if $\frac{7d-4c'}{4} \leq d \iff 7d - 4c' \leq 4d \iff 3d \leq 4c' \iff d \leq \frac{4}{3}c'$, then $T_4(n) \leq cn^{\log_2 7} - dn^2$. Thus we have proved by induction that $\forall n \in \mathbb{N},\ T_3(n) \leq cn^{\log_2 7} - dn^2$ for a proper c and d. So $T_3(n) \in O(n^{\log_2 7})$.

Now we want to prove that $T_4(n) \in \Omega(n^{\log_2 7})$. We select the representative $cn^{\log_2 7} \in \Omega(n^{\log_2 7})$ and the representative $c'n^2 \in \Theta(n^2)$. Inductive assumption: we assume that $\forall m < n,\ T_4(m) \geq cm^{\log_2 7}$. We want to prove that $\forall n,\ T_4(n) \geq cn^{\log_2 7}$.

$$
\begin{aligned}
T_4(n) &= 7 * T_4(n/2) + \Theta(n^2) \\
&\geq 7c\Big(\frac{n}{2}\Big)^{\log_2 7} + c'n^2 \\
&= 7c\frac{n^{\log_2 7}}{2^{\log_2 7}} + c'n^2 \\
&= 7c\frac{n^{\log_2 7}}{7} + c'n^2 \\
&= cn^{\log_2 7} + c'n^2 \\
&\geq cn^{\log_2 7}
\end{aligned}
$$

Thus we have that $T_4(n) \geq cn^{\log_2 7}$, so we have proved by induction that $\forall n \in \mathbb{N},\ T_3(n) \geq cn^{\log_2 7} - dn^2$ for a proper c. Then $T_4(n) \in \Omega(n^{\log_2 7})$. Therefore we have proved that $T_4(n) \in \Theta(n^{\log_2 7})$.

References

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press. MIT Press, 2009.